Universidad de Valladolid

Escuela de Ingeniería Informática

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

(Mención en Ingeniería de Software)

Dashboard gráfico para la gestión de usuarios en videojuego de entrenamiento

Autor:

D. David Sastre Conde

Tutor:

David Escudero Mancebo

Resumen

Este trabajo de Fin de Grado consiste en el desarrollo de una aplicación de monitorización de usuarios en un entorno de realidad virtual a través de un panel de control gráfico. La aplicación está destinada a utilizarse como herramienta de apoyo para la formación y/o entrenamiento de usuarios en entornos virtuales ahorrando así espacio y costes en este ámbito.

La aplicación está ideada para que existan dos tipos de usuarios que ejecutan la aplicación en terminales distintos, uno de los usuarios será el que controle el panel de control y que utilizará la aplicación de forma tradicional, es decir, como una aplicación de escritorio en el que los controladores son el ratón y el teclado. El otro usuario será el jugador que realiza las actividades dentro del entorno virtual, para ello deberá utilizar un casco de realidad virtual específico y ambos usuarios se conectarán a través de la red desde sus respectivos terminales.

Además del desarrollo de este panel de control, también se crea una base de datos donde se almacenan y se recupera toda la información relevante de la aplicación, como elementos referentes al panel de control, a los distintos tipos de usuarios y a lo ocurrido durante las partidas de juego, dotando así a la aplicación de una base para realizar futuros estudios y analizar el paso de múltiples usuarios por un entorno que simula una situación real.

La aplicación está basada en el desarrollo de una aplicación comercializada para la marca IVECO y por ello se han restringido muchas de las funcionalidades del software previo, como el número de escenarios de juego y muchas de las funcionalidades y el flujo general de la aplicación entre otras cosas, por ello se ha decidido que el usuario que utiliza la aplicación con las gafas de realidad virtual no utilice los controladores de las gafas para realizar las acciones, sino que utilizará un controlador tradicional como un mando o un teclado, estos detalles se explican en el manual de usuario de este trabajo.

Durante el desarrollo de este proyecto se han llevado a cabo tareas de investigación sobre las distintas posibilidades y métodos que existen para construir una aplicación que funcione tanto en dispositivos de realidad virtual como en dispositivos tradicionales como un PC de sobremesa.

Tabla de Contenidos

Contenido

Lista de figuras

Lista de tablas

1. INTRODUCCIÓN

1.1 Descripción del proyecto

El auge de las tecnologías de realidad virtual está suponiendo una revolución en muchos ámbitos, ya que permite tele transportar al usuario a un entorno en el que es posible simular todo tipo de contenido, pudiendo utilizar su cuerpo y sus sentidos e interactuar con el sistema de una forma que hasta hace poco se reservaba para las más imaginativas películas de ciencia ficción.

Uno de los campos donde se abren más posibilidades utilizando este tipo de tecnología es en el ámbito de la formación laboral y entrenamiento de todo tipo de actividades, desde aprender a montar un mueble en tu propio salón de casa hasta aprender nuevos oficios de diversa índole y casi desde cualquier parte de un modo totalmente inmersivo.

Este proyecto está enfocado a la creación de un sistema que permita a un usuario manager controlar a un jugador en realidad virtual a través de un panel de control en el que podrá modificar el entorno del jugador a su gusto para ver cómo reacciona este, además el jugador obtendrá una puntuación al final de la partida en función de cómo haya realizado el juego y estos datos se recogerán y enviarán a una base de datos local para un posterior análisis.

1.2 Motivación

El hecho de que una persona pueda trasladarse virtualmente a todo tipo de entornos y que además pueda utilizar su propio cuerpo para navegar a través de este ofrece tantas posibilidades que es difícil no encontrarle un uso práctico en casi cualquier ámbito, si hasta ahora el contenido llegaba a través de un portal o una ventana, con la realidad virtual traspasamos ese portal para meternos dentro de la acción, lo que supone una experiencia totalmente nueva y otra forma de visualizar e interactuar con los sistemas, pudiendo disfrutar de contenido enfocado al entretenimiento como pueden ser los videojuegos, el cine o una mezcla de ambos hasta llegar a utilizarse en aplicaciones de salud orientados a curar enfermedades mentales o realizar ejercicios de rehabilitación.

Las posibilidades están únicamente limitadas por la imaginación. Además, el mercado está apostando por este tipo de tecnologías y experiencias de realidad virtual, por lo que incentiva aún más el interés que despierta el investigar y desarrollar aplicaciones y software relacionado con esta nueva forma de interactuar con el contenido digital.

1.3 Objetivos

El objetivo principal de este proyecto será abordar la construcción de un sistema destinado a la formación y entrenamiento laboral para dos puestos de trabajo específicos que existen actualmente en la fábrica de IVECO Valladolid. Para cada uno de estos trabajos el jugador deberá superar una serie de actividades para completar el juego y obtener una puntuación en función del tiempo en un escenario concreto asociado al tipo de trabajo a desempeñar.

Para llevar esto a cabo el jugador deberá ponerse unas gafas de realidad virtual junto con unos controladores que harán la función de manos con los que podrá interactuar con el entorno para superar estas actividades, teniendo en cuenta el tiempo empleado, el orden correcto de la realización de estas y otros parámetros a tener en cuenta. Esta será la aplicación del cliente que deberá ejecutarse junto con un dispositivo adecuado para realidad virtual (Un equipo PC con una configuración recomendada)

Por otro lado, se construye una aplicación para que un usuario manager pueda controlar en todo momento la jugabilidad e interacción del usuario desde el mismo u otro dispositivo. A través de esta aplicación el usuario manager podrá cambiar parámetros del entorno del jugador para ver cómo

reacciona este, visualizar el entorno y al jugador desde diferentes ángulos y obtener toda la información relevante de la partida y que estos datos sean alojados en una base de datos local para su posterior estudio.

1.4 Software previo

Para este proyecto se cuenta con un desarrollo previo de una aplicación que ya contiene gran parte del escenario y funcionalidad básica a la que se debe enfrentar el jugador que debe superar las actividades de la partida. Este software incluye la realización de los modelos 3D, texturizado y animación de todos los objetos que verá el jugador en la partida, así como la mayor parte de la funcionalidad del juego.

Este software previo cuenta con dos escenarios, uno de fabricación de piezas de carrocería y otro de montaje de piezas de carrocería. A continuación, veremos que funcionalidades contienen cada uno de estos mapas previamente desarrollados en los que se basa el desarrollo de software de este proyecto:

Escenario de Fabricación. En este escenario el jugador debe colocarse una serie de protecciones para evitar cualquier posible daño que pueda sufrir en el desarrollo de su actividad laboral, estas protecciones o *EPIS* constan de guantes de protección, manguitos industriales anti-cortes, gafas de protección y cascos anti-ruido. Además, el jugador debe llevar una serie de piezas concretas y en un orden determinado hacia una máquina que se encargará, a través de unos brazos robóticos, de soldar las piezas colocadas. Este ejercicio de colocación de piezas metálicas consta de dos fases, el jugador tiene que colocar 3 en la primera fase y otras 3 en la segunda fase, teniendo que pulsar el botón correcto de un panel de control para pasar de la fase 1 a la fase 2.

Todos estos pasos vienen señalizados y guiados a través de indicadores dentro del propio juego y además a gracias a una narración que puede escucharse a través de los auriculares, ya que el software inicial está orientado a la concienciación sobre prevención laboral y riesgos laborales y cómo poder evitarlos.

En este escenario se modificará este software haciendo que la experiencia ya no sea guiada y no se le indique al jugador en cada momento lo que debe hacer, de esta forma dejamos al jugador a su libre albedrío dentro del juego, pudiendo optar por ponerse las protecciones o no, coger una pieza metálica en un orden incorrecto o pulsar el botón equivocado en el panel de mandos, de esta forma desviamos el objetivo de la aplicación desde la prevención de riesgos laborales hacia la formación laboral y entrenamiento.

Escenario de Montaje. El jugador realiza la actividad de montaje de una de las piezas de una furgoneta, para llevar a cabo esta actividad el jugador debe colocarse correctamente los EPIS, que en este caso son el casco de protección y las botas con puntera de hierro. Su función principal será la de colocar correctamente unas piezas a lo largo de la base de la carrocería del vehículo y, una vez colocadas, el jugador deberá coger un destornillador industrial para apretar cada una de esas piezas y dar por concluida la actividad. Al igual que en el escenario de fabricación este proceso es guiado a través de indicadores y de narración auditiva.

2. REALIDAD VIRTUAL

2.1 Introducción y breve historia

En este capítulo se explican algunos de los fundamentos de funcionamiento y conceptos relevantes de las tecnologías de realidad virtual más destacadas, así como una breve descripción y clasificación de los dispositivos y tecnologías existentes para poder entender con mayor claridad algunos de los aspectos clave en los que se basa este trabajo.

La idea de realidad virtual aparece de la mano del ingeniero Ivan Sutherland en 1965 cuando publicó un artículo con el título '*The Ultimate Display*' donde describiría los conceptos básicos de esta tecnología al fabricar un dispositivo de Realidad Virtual muy primitivo basado en un visor construido con tubos de rayos catódicos para cada ojo y un sistema mecánico de seguimiento.

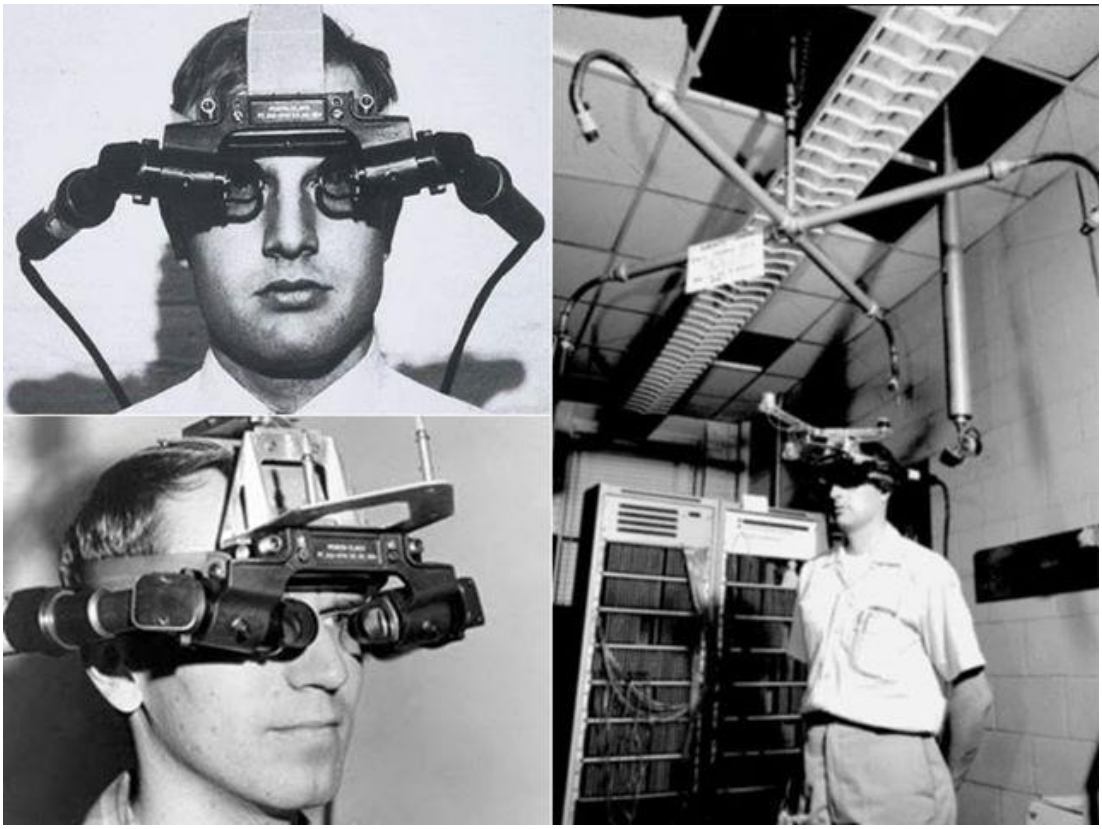


Figura 1: Tubo de rayos catódicos ideado por Ivan Shutherland

El siguiente gran intento de construir visores de realidad virtual aparece a finales de los años 80 y principios de los 90 de la mano de grandes marcas de videojuegos como Sega y Nintendo que llegaron a construir prototipos funcionales pero que nunca llegaron a ser comercializados. El dispositivo de Sega se conoce como '*Sega VR*' y sus pantallas tenían una resolución de 320x224 píxeles y mostraban hasta 64 colores, funcionaba con unos sensores que irían incorporados y que actualizaban la posición del dispositivo en el espacio. El dispositivo que lanzó Nintendo unos años se conoce como '*Virtual Boy*' y a diferencia de la consola de Sega, esta sí que llegó a comercializarse y contaba con una resolución de pantalla de 384 x 224 píxeles y contaba también con sensores de posicionamiento del casco, pero debido a sus bajas ganancias se descatalogó después de un año en el mercado.

Figura 2: Dispositivo Virtual Boy desarrollado por Nintendo

Se dice que fue Google en el año 2000 quien volvió a revivir el fantasma de la realidad virtual sin pretenderlo gracias al desarrollo de su '*Street View'* y también gracias a la evolución de las tecnologías móviles que contribuyeron en gran medida al desarrollo de las tecnologías que usan hoy en día los dispositivos de realidad virtual como el giroscopio o el acelerómetro entre otras.

En 2010 apareció finalmente el primer dispositivo que puede considerarse puramente de realidad virtual gracias a la creación de las Oculus Rift gracias a una campaña en '*Kickstarter'*. Estas gafas tenían un ángulo de 90 grados de visión, una resolución y capacidad de procesamiento y refresco de imagen nunca vistas hasta la fecha, llegando a pasar por varios prototipos y siendo comprado finalmente por el gigante Facebook.

Figura 3: Primer dispositivo desarrollado por Oculus Rift conocido como DK1

Actualmente se conoce como realidad virtual al entorno construido mediante software que, a través de un dispositivo preparado para soportar tecnologías de realidad virtual de distinta índole, le proporciona al usuario una sensación de inmersión en dicho entorno. Esta sensación de inmersión se reconoce por presentar características como la visión en 360º, en la que el usuario puede visualizar el entorno virtual a través de su mirada y movimientos de cabeza y ojos [6], muy diferente al modo tradicional de interacción con el software, que suele ser a través de una pantalla (bien en un PC o en un widget como puede ser un reloj inteligente).

Esta sensación de inmersión se hace todavía más intensa gracias a algunos periféricos con los que ya contamos en el mercado actual, como la inclusión de dispositivos para poder hacer uso de tus propias manos en dicho entorno virtual gracias a dispositivos como *leap motion*, o los controladores de HTC VIVE y Oculus Rift. Además, en estos entornos virtuales podemos movernos como en el mundo real o mediante la interfaz de la aplicación si el dispositivo y la aplicación en concreta lo permiten, haciendo que la sensación sea mucho más real.

2.1.1 El entorno virtual

Se pueden diferenciar a grandes rasgos dos tipos de entornos de realidad virtual, teniendo aquellos que han sido creados a través de herramientas de modelado, texturizado y animación 3D en el cual nos introducimos gracias a los distintos tipos de dispositivos de realidad virtual y que necesitan de un desarrollo a través de diversas herramientas de software como por ejemplo una aplicación de pintar en 3 dimensiones con unas gafas de realidad virtual, por otro lado, aquellos que han sido grabados o fotografiados en entornos reales y por tanto lo que vemos es un video o una imagen en 360 grados o parte de ella, lo que conocemos como video/fotografía real 360.

La característica principal que define un entorno virtual es que el usuario puede visualizar en 360º de una forma similar a cuando lo hace en la realidad, es decir, mediante giros de cabeza y ojos. Además, dependiendo de si el software permite o no interacciones del usuario y a qué nivel, también puede interactuar con otras partes de su cuerpo como las manos gracias a distintos tipos de controladores que realizan un seguimiento del dispositivo [2].

2.1.2 Definiciones

En este apartado se describen algunas de las definiciones y conceptos básicos utilizados con frecuencia en el ámbito de la realidad virtual y que serán recurrentes en esta memoria.

VR – Siglas de Virtual Reality, también en español RV (realidad virtual).

UE4 – Siglas de Unreal Engine 4, motor gráfico creado por la compañía Epic Games.

Estereoscópico – Técnica en la que puede visualizarse un contenido en 3 dimensiones o que crea la ilusión de ello. Concepto normalmente utilizado para referirse a información visual proyectada a los dos ojos por separado, de forma que en conjunto crea la ilusión de contenido tridimensional.

Tracking – Conjunto de técnicas que sirven para determinar la posición y orientación de un usuario dentro de un espacio determinado. En realidad virtual, este seguimiento o posicionamiento se realiza a través de sensores que se comunican con el casco de realidad virtual.

Controllers – Se hace referencia a los dispositivos que se utilizan junto a cascos de realidad virtual y que permiten la interacción con los elementos del entorno virtual. Son una especie de mandos que también poseen tracking con respecto al casco de realidad virtual.

Video e imagen 360 – Contenido multimedia creado con el fin de ser visualizado en 360º grados en cualquier dirección por el usuario.

Antialiasing – En el ámbito de la visualización gráfica de objetos, nos referimos a las técnicas de eliminación de ruido o atenuación de *aliasing* de formas representadas en un monitor, bien sea a través de monitores convencionales o de dispositivos de realidad virtual.

Blueprint – En el motor Unreal Engine 4, nos referimos a fragmentos de código encapsulados que se conectan entre sí mediante nodos.

Constellation – Es el dispositivo encargado de realizar el seguimiento/posicionamiento de las gafas y los controladores de las gafas de Oculus Rift.

HMD – Del inglés, Head Mounted Display, es un dispositivo de visualización de contenido de realidad virtual con las características propias de esta.

Asset – En el mundo del diseño, se dice de todo aquel recurso que sirve para crear contenido, por ejemplo, un vehículo en un videojuego.

Preset – En el editor de proyectos del lanzador de Epic Games [9], hace referencia a ajustes preestablecidos para un determinado tipo de juego.

Wdiget – En Unreal Engine, nos referimos a todos los elementos gráficos que forman parte de la interfaz del usuario, por ejemplo, todo tipo de menús de juego.

Mesh – En el ámbito del diseño 3D, también conocido como malla en castellano, es un conjunto de polígonos formados por caras, vértices y puntos.

Escenario – En Unreal Engine 4, también conocido como mapa o nivel, se refiere al conjunto de objetos y componentes con los que se puede interactuar dentro de la aplicación.

2.1.3 Dispositivos existentes y funcionamiento

En este apartado se describen brevemente algunos de los dispositivos de realidad virtual más utilizados actualmente y qué tipo de contenido se puede reproducir en cada uno de ellos con el fin de clarificar el ámbito objetivo del software desarrollado en este trabajo.

2.1.3.1 Cardboard

Se trata de un dispositivo normalmente construido con materiales de baja calidad como plástico o cartón (de ahí el nombre de *cardboard* o cartulina en castellano) que posee dos *lentes biconvexas* separadas entre sí y con una ranura frente a las lentes para introducir un dispositivo móvil de un tamaño y tecnología compatible. El usuario se coloca las gafas de forma que las lentes coincidan con la posición de los ojos a forma de gafas.

Figura 4: Gafas de cartón o Cardboard

Con este tipo de gafas podemos reproducir contenido multimedia (video, imágenes o aplicaciones basadas en estas dos últimas) gracias a la pantalla estereoscópica dividida [7] que se proyecta en el dispositivo móvil. Este tipo de aplicaciones deben detectar el movimiento del usuario haciéndose uso del giroscopio del dispositivo haciendo que la aplicación o contenido reaccione a las acciones del usuario cuando se aplica movimiento al dispositivo.

Figura 5: Dispositivo móvil ejecutando una aplicación en vista estereoscópica

Gracias a la acción de las dos *lentes biconvexas* se crea la ilusión de inmersión, haciendo que el usuario sienta la sensación de que se encuentra en el centro de la acción de lo que se está reproduciendo en el dispositivo móvil.

2.1.3.2 Samsung Gear VR

Las gafas de Samsung conocidas como *Samsung Gear VR*, son dispositivos que funcionan únicamente con smartphones inteligentes compatibles fabricados también por *Samsung* y que funcionan de una forma similar a las *Cardboard*, es decir, ambos utilizan un dispositivo móvil para su funcionamiento, poseen lentes binoculares y la percepción del contenido es en 360º, sin embargo, estas gafas tienen un añadido y es que el teléfono inteligente se conecta mediante un *puerto dock* a las gafas de *Samsung*, permitiendo que a través del hardware de las gafas puedan controlarse algunos aspectos de las aplicaciones del dispositivo mediante un pequeño *touchpad* desde el propio hardware [8].

Estas gafas también hacen uso del acelerómetro del Smartphone para visualizar su contenido en 360º, existe una mayor variedad de aplicaciones y la resolución del contenido suele ser de mejor calidad.

Como en las gafas cardboard no contamos con posicionamiento o tracking, de modo que, aunque nos desplacemos hacia alguna dirección siempre se visualizará el contenido en el centro de la acción, es decir, únicamente podemos ver el contenido en 360º pero desde una posición estática.

Figura 6: Gafas Samsung Gear VR y las diferentes partes de las gafas

2.1.3.3 Oculus Rift

Estas gafas están desarrolladas por la empresa *Oculus Rift* y están destinadas a funcionar en equipos informáticos de altas prestaciones. A diferencia de las gafas *cardboard* o de las *Samsung gear VR*, estas gafas permiten un posicionamiento o tracking en 3 dimensiones, haciendo que la sensación de inmersión sea mucho mayor, ya que el usuario puede acercarse al contenido virtual generado.

Estas gafas cuentan con dos paneles *OLED* (uno para cada ojo) con una resolución de 1080 x 1200 píxeles y con una ratio de refresco de 90 Hz. Van conectadas al equipo a través de un cable HDMI y un cable USB y requieren de un software específico para su correcto funcionamiento [9]. Por otra parte, tenemos el dispositivo de tracking o '*constellation'* [10] que es el encargado de realizar el seguimiento de las gafas a través del espacio 3D de las aplicaciones gracias a una serie de sensores que junto con unos pequeños LEDs de infrarrojos que van en las gafas determina de forma precisa la posición del dispositivo sin prácticamente latencia.

Los controladores '*Oculus Touch'* permiten además utilizar tus propias manos en el entorno virtual, que también cuentan con posicionamiento, pero requieren de otro dispositivo '*Constellation'* adicional. Estos cuentan con una gran variedad de botones y funcionalidades haciendo que la experiencia de realidad virtual sea mucho más enriquecedora y abriendo así la puerta a un montón de nuevas posibilidades de desarrollo de software en VR.

Figura 7: Gafas Oculus Rift y las distintas partes de las gafas

2.1.3.4 HTC VIVE

Las gafas *HTC VIVE* están desarrolladas por la empresa *HTC* en conjunto con la empresa *Valve* mundialmente conocida por el desarrollo de su plataforma de juegos y software para PC *Steam*, estas gafas de realidad virtual comparten muchas de las características que poseen las gafas de *Oculus Rift* mencionadas en el apartado anterior, ya que también requieren de un equipo informático de alto rendimiento para su funcionamiento.

Estas gafas cuentan también con una tasa de refresco de 90Hz y una pantalla *OLED* para cada ojo con una resolución de 1080x1200 píxeles cada una, contando al igual que las Oculus Rift, con una serie de sensores que permiten calcular la velocidad, aceleración y posición del usuario gracias también a sus propios dispositivos de tracking o posicionamiento, llamados 'lighthouse' [11].

Los 2 *lighthouse* que vienen junto con las gafas *HTC VIVE* sirven para localizar tanto el casco *HMD* como los controladores del mismo que también vienen incluidos junto con las gafas y que se utilizan normalmente como mandos para las manos al igual que ocurría con los *Oculus Touch* de *Oculus*, para ello utilizan una tecnología de láseres de luz estructurada [11] de modo que al deformar esa malla generada entre los dos *lighthouse* permite calcular la posición exacta del usuario.

Estos dispositivos de tracking pueden deben configurarse o bien en parejas, teniendo uno de ellos emitiendo en el canal B y otro en el canal C y permitiendo así que podamos desplazarnos a través del espacio configurado a través de la plataforma de steam, o bien puede utilizarse un único *lighthouse* emitiendo en el canal A, pero esto reduce nuestra libertad de movimiento por el espacio virtual.

Figura 8: Gafas de realidad virtual HTC VIVE con los dos lighthouse y controladores

2.1.3.5 Steam VR

Las gafas de Oculus cuentan con su propia plataforma de contenidos para PC desde la que podemos acceder a múltiples experiencias desarrolladas por la propia compañía o por terceros, además de configurar algunos parámetros de las gafas y conectar con nuestra cuenta online de Oculus, pero además, la plataforma Steam ya integra este tipo de hardware como una categoría más en sus listas de software y juegos, pudiendo ejecutar programas y aplicaciones de Oculus Rift o compatibles con este directamente a través de la plataforma de steam gracias al desarrollo de la aplicación 'Steam VR' [12] que monitoriza las gafas y nos permite acceder al contenido de esta plataforma utilizando estas gafas además de indicarnos información relevante acerca de la conexión de las gafas con el PC o si se encuentran problemas con el tracking entre otras cosas.

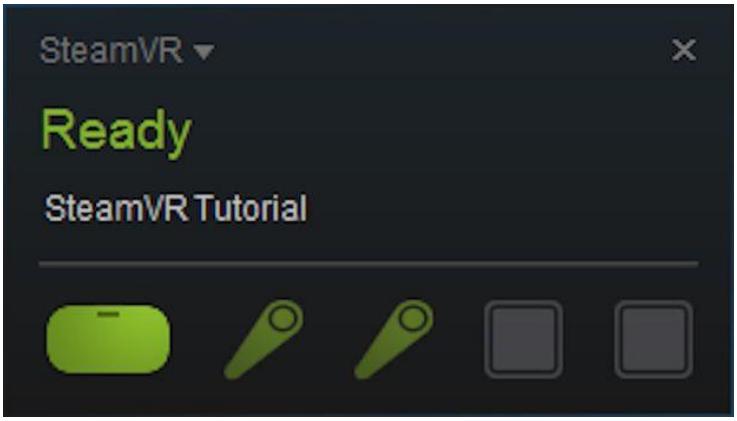


Figura 9: Interfaz de la aplicación de escritorio Steam VR

2.2 Procesos y herramientas de desarrollo de un entorno virtual

La creación de un entorno virtual con distintas funcionalidades puede requerir de conocimientos en múltiples disciplinas dependiendo del tipo de contenido que se quiera desarrollar, teniendo desde aplicaciones sencillas para dispositivos móviles basadas en video 360 destinadas a funcionar con las gafas cardboard o las de Samsung Gear VR, o también puede ser el desarrollo de un videojuego completo que tiene como objetivo funcionar en las gafas de HTC VIVE y Oculus Rift [2].

Dependiendo del tipo de contenido a desarrollar deberemos utilizar unas herramientas y recursos u otros. En el caso de este trabajo se desarrolla un panel de control con conexión a una base de datos que monitoriza las partidas de otros jugadores para así guardar información relevante sobre estas y ser analizada a posteriori, y se desarrolla sobre una base de aplicación que cuenta con gran cantidad de modelos 3D con alta calidad de texturas y optimizados para que la aplicación pueda funcionar con fluidez en la mayor parte de dispositivos que sean compatibles con estas tecnologías, y que además cuenta con una importante implementación de funcionalidad previa, y aunque son imprescindibles conocimientos en programación, desarrollo de software y planificación de proyectos para el desarrollo de la funcionalidad, en muchas ocasiones es necesario contar con conocimientos sobre creación de objetos 3D, texturizado, animación y optimización de modelos para videojuegos, diseño gráfico de interfaces de usuario y otros elementos gráficos si va a desarrollarse una aplicación como sobre la que se explica en esta memoria.

Como en este trabajo ya partíamos sobre una base previa de modelos y animaciones que pueden englobarse en el ámbito del diseño, y, además esta base estaba realizada con el motor gráfico Unreal Engine 4 de Epic Games y por lo tanto el desarrollo de las nuevas funcionalidades sobre las que trata este trabajo se han desarrollado con la misma herramienta, sin embargo, vamos a explicar brevemente las opciones más actuales y que más posibilidades nos ofrecen a la hora de elegir un software con el que trabajar para desarrollar este tipo de aplicaciones de realidad virtual.

2.2.1 Unity

Se trata de uno de los motores gráficos más utilizados y conocidos en todo el mundo, gracias a que cuenta con una comunidad enorme de desarrolladores y artistas que lo utilizan para multitud de propósitos distintos dado que es una herramienta multiplataforma y que se ha hecho famosa por adaptarse fácilmente a casi cualquier tipo de dispositivo destino, desde videojuegos de complejos de realidad virtual hasta simple contenido de navegador o software destinado a dispositivos móviles.

Está desarrollado por la empresa Unity Technologies [2] y actualmente se encuentra en la versión 5.6.1, contando hasta la fecha con un sinfín de tutoriales y documentación que han ido creciendo poco a poco desde su lanzamiento en 2004. Existen varios tipos de suscripción que ofrecen más servicios a mayores en función del tipo de suscripción que se realice, teniendo el motor con algunos servicios básicos para su uso gratuito siempre y cuando no se obtengan beneficios por encima de los 100.000 $ anuales, en cuyo caso la suscripción debe actualizarse a una de mayor categoría.

Unity se caracteriza por ser un software robusto y capaz de adaptarse a las nuevas tecnologías como en el caso de la realidad virtual, en parte porque existe una comunidad muy grande de personas que utilizan el motor gráfico para multitud de tareas y muchas de estas personas se dedican a crear contenido y plugins para compartirlo a través de la propia tienda de Unity, la ‘Asset Store’, esto hace que rápidamente haya una gran cantidad de gente trabajando e integrando en el motor lo último en tecnología y desarrollando a su vez nuevo contenido. Desde su página web nos exponen algunos de los trabajos más relevantes que han sido desarrollados con esta herramienta [14].

Esta herramienta utiliza lo que se denominan como *Scripts*, que son fragmentos de código que permiten modificar y añadir funcionalidades personalizadas a distintos tipos de elementos dentro del software que se está desarrollando, como por ejemplo hacer que una esfera haga un recorrido determinado a través un terreno. Para programar estos scripts son necesarios conocimientos de programación en los dos tipos de lenguajes nativos soportados por el motor gráfico, que son [13]:

- Lenguaje C#
- UnityScript

Además, Unity cuenta con una enorme tienda o market online que dispone de gran cantidad de assets y otras herramientas de software que gracias a una activa comunidad contribuye a que sea uno de los baluartes de este potente motor gráfico.

2.2.2 Unreal Engine 4

Al igual que Unity se trata de un motor gráfico creada por la compañía Epic Games que aparece por primera vez en 1998 y que a partir de entonces irá evolucionando e incorporando nuevas herramientas y tecnologías y estando disponible en su versión 3 a finales del 2009 hasta llegar a la última versión del motor hasta la fecha, el Unreal Engine 4 disponible desde el 2 de marzo de 2015.

Este motor gráfico se crea con el objetivo de facilitar la tarea de creación de contenido digital destinado a múltiples plataformas de una manera simple e intuitiva y que no requiera de amplios conocimientos de programación si el software no requiere una heurística o una complejidad muy elevada, de esta forma muchos profesionales de distintos ámbitos prefieren utilizar esta herramienta para su trabajo, de esta forma surgen los blueprints o fragmentos de código encapsulados en cajitas visuales que se interconexionan entre sí dando lugar a lo que vendría a ser el equivalente al código tradicional desarrollado en otros lenguajes de programación [15].

Unreal Engine permite crear un proyecto basándonos en código C++ (opción que sería muy similar al modo de trabajo que se utiliza en Unity) o bien crear un proyecto con blueprints, además nos ofrece una amplia gama de presets y contenido predefinido que podremos utilizar tanto en proyectos basados en C++ como en proyectos basados en blueprints, incluso recientemente cuenta con ajustes preestablecidos para aplicaciones destinadas a realidad virtual en distintos dispositivos [16].

Las características que hacen que este motor sea una muy buena opción en contraposición con otros motores, es que pone especial atención en la calidad de los aspectos gráficos y del entorno general de las escenas, con esto nos referimos al tratamiento que hace por defecto de las sombras en objetos, la iluminación de la escena y el motor de renderizado entre otras muchas cosas, pudiendo de esta forma conseguir resultados profesionales de una forma más sencilla.

3. UNREAL ENGINE 4

3.1 Requisitos de uso

Dado que el software previo sobre el que se basa esta aplicación ha sido desarrollado mediante esta herramienta, y por tanto se ha considerado como la opción más inteligente a la hora de desarrollar las nuevas funcionalidades que se incorporarán al proyecto. Además, Unreal permite manejar widgets y otros elementos gráficos que funcionan muy bien con los objetivos de este proyecto.

Para empezar a utilizar Unreal Engine es necesario registrarse en la página de la desarrolladora [17] para más tarde comenzar con la descarga del lanzador de Epic Games para escritorio, se necesita un equipo que cumpla las siguientes características mínimas para su correcto funcionamiento [18]:

- OS: Windows 7 en adelante
- CPU: Intel Core 2 Duo / AMD Athlon 2x2
- RAM 2GB
- GPU: NVIDIA Geforce 7800 / AMD Radeon HD 4600 / Intel HD 4000
- DirectX 9.0c

Aunque esto son solo los requisitos mínimos para un funcionamiento correcto de la herramienta, más adelante veremos algunos de los requisitos necesarios para funcionar con las últimas funcionalidades añadidas al motor y para aplicaciones de realidad virtual.

Además del propio lanzador de Epic Games, necesitaremos instalar Visual Studio 2015 de Microsoft [19] y en el proceso de instalación seleccionar algunas herramientas adicionales de desarrollo como el SDK de Windows.

3.2 La interfaz de desarrollo y el lanzador de Epic Games

El lanzador o launcher de Epic nos ofrece una interfaz desde la que podremos instalar las diferentes versiones del motor disponibles hasta la fecha, contando con los apartados de ‘Comunidad’ donde se publican algunos de los últimos y más destacados trabajos de los usuarios de esta herramienta, ‘Aprender’ donde los desarrolladores de la compañía publican tutoriales y contenido que sirven de ayuda en el aprendizaje para otros usuarios, dispone también de una tienda o ‘Bazar’ donde se puede adquirir distinto tipo de contenido y por último cuenta con la zona de ‘Bliblioteca’ desde donde podremos ver todos nuestros proyectos y acceder a ellos para comenzar a desarrollarlos en el editor, pero también nos permite crear nuevos proyectos basándonos en distintos tipos de presets de contenido ya dados por el motor.

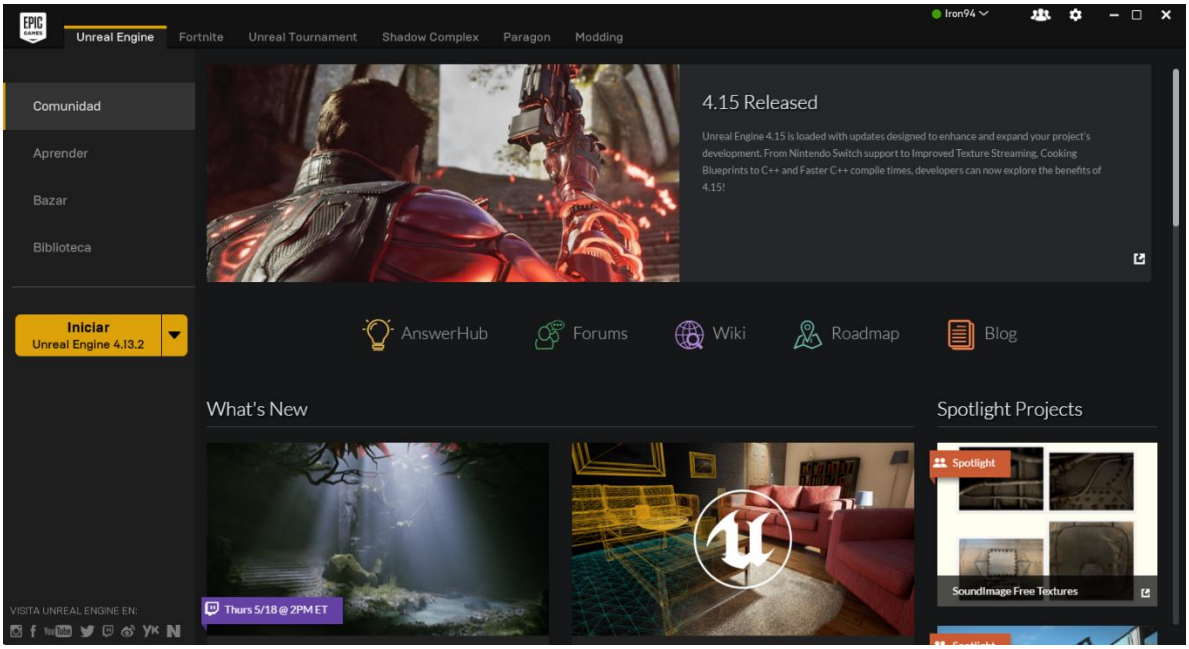


Figura 10: Interfaz del lanzador de Epic Games

Para crear un proyecto deberemos instalar la versión del motor que mejor se adapte a las necesidades de nuestro proyecto, podemos leer las notas y herramientas incorporadas a cada versión desde aquí [21]. Una vez hayamos decidido con qué versión del motor vamos a trabajar y hayamos instalado las herramientas pertinentes de las que se hablaba en el apartado 3.1 podremos crear un nuevo proyecto pudiendo seleccionar si nuestro proyecto va a basarse en blueprints o por el contrario va a desarrollarse mediante programación en lenguaje C++.

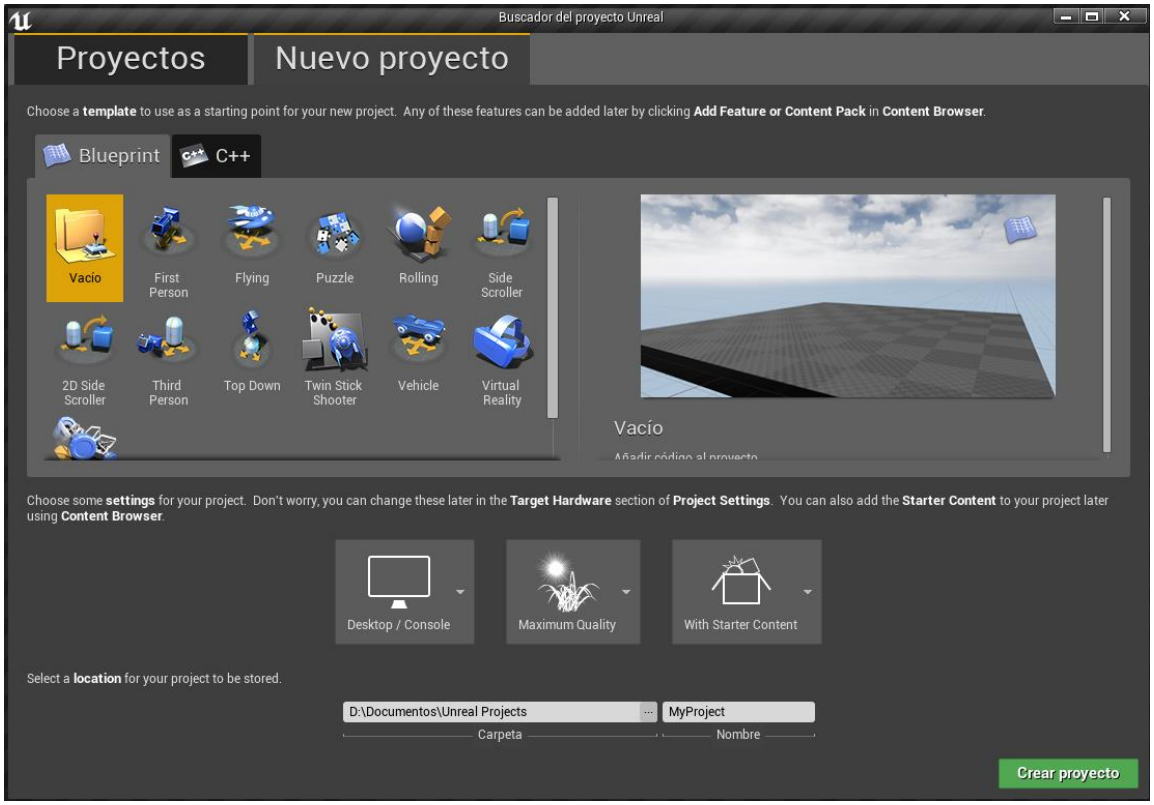


Figura 11: Interfaz de creación de proyectos de Unreal Engine 4

Desde esta interfaz podremos elegir alguno de los presets existentes que mejor se ajusten al trabajo que vamos a desarrollar en esta plataforma, además de otras opciones como la selección de la plataforma objetivo (móvil o PC), cómo queremos que sea la calidad gráfica del proyecto por defecto (por ejemplo, para móviles deberemos reducir esto para evitar problemas de funcionamiento y optimizar nuestro contenido) y si deseamos contar con algunos materiales y objetos predefinidos por el motor (‘Starter content’), también deberemos indicar una ruta y un nombre de archivo donde se alojará nuestro trabajo. Existe de manera estable desde la versión 4.12 de Unreal Engine el preset de realidad virtual que incorpora una configuración básica para contenido destinado a funcionar en dispositivos de realidad virtual, pudiendo hacer uso de algunas funciones ya desarrolladas como teleportarse a través del entorno virtual gracias a los controller de las gafas de realidad virtual, agarrar y lanzar objetos y un pequeño editor de objetos en realidad virtual. Esta será la base sobre la que se asienta gran parte del software previo desarrollado en este proyecto [20].

Una vez hayamos seleccionado todas las opciones y se haya creado nuestro proyecto, pasaremos a visualizar la interfaz del ‘editor del nivel’ de la versión del motor seleccionada de Unreal Engine. A través de esta interfaz principal controlaremos la mayor parte del contenido y funcionalidad desarrolladas en nuestro trabajo, teniendo, por defecto, en la parte derecha del editor aquella información relacionada con nuestro escenario actual, como son objetos, elementos y assets y sus características, pudiendo seleccionar uno de estos y ver así algunos de los ajustes básicos que poseen, como la posición en la que se encuentran dentro del escenario, la rotación y escala y otros ajustes relacionados con dicho elemento.

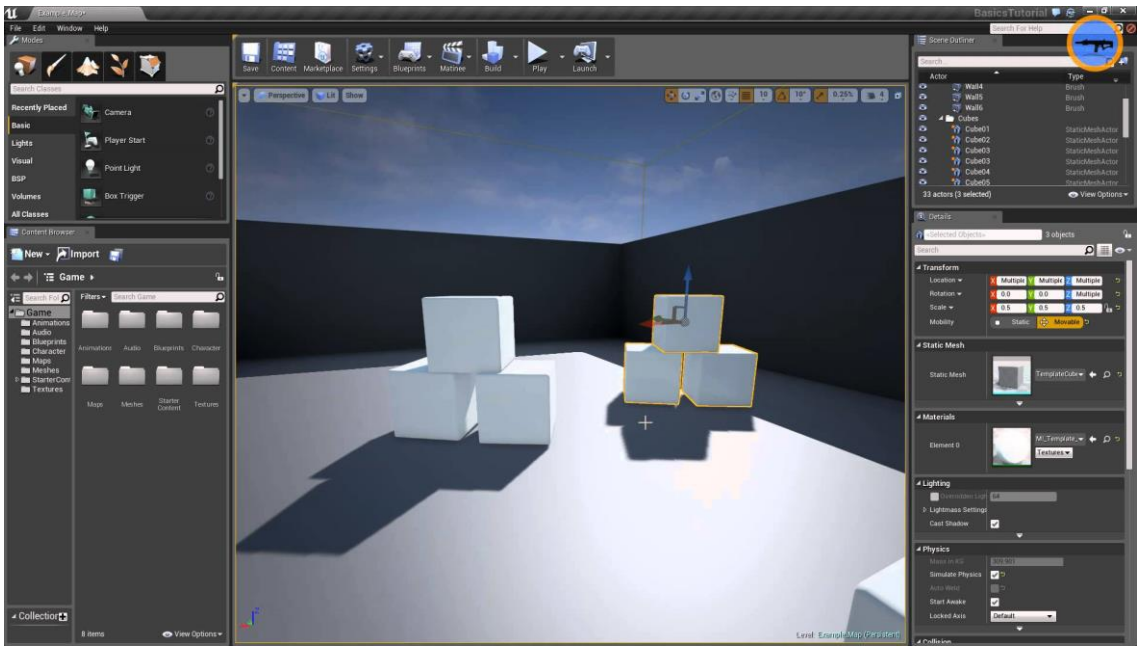


Figura 12: Interfaz principal de desarrollo en el motor Unreal Engine 4

En la parte central podemos **visualizar el contenido de forma gráfica** y desplazarnos a través de este, pudiendo colocar y mover objetos, colocando luces, cámaras…etc. y en general poder ver y controlar todos los aspectos gráficos que va a contener nuestro escenario y nuestro proyecto, esta zona se conoce como *viewport*, además en la parte superior de esta ventana disponemos también de los accesos a los *blueprints* del nivel o escenario actual, que vendría a ser el equivalente a una clase Java que contendría el nivel y todos los componentes visuales y no visuales que actuarán en el juego,

también tenemos acceso a algunos ajustes del editor, a poder **simular la ejecución en tiempo real de nuestro contenido** y al compilado y diseño de la iluminación de la que se habla en el apartado siguiente. Podemos consultar estas herramientas en la documentación oficial de Epic Games [22].

3.3 Conceptos y nociones básicas

En este apartado se detallan algunos de los aspectos más esenciales del funcionamiento del motor para poder comprender mejor aquellas tareas llevadas a cabo durante el desarrollo de este proyecto.

En primer lugar, se hablará de algunos conceptos básicos y fundamentales de un proyecto de Unreal Engine 4. Después, se comentarán algunos elementos básicos del framework de Unreal y su flujo de trabajo y se pondrán algunos ejemplos. Más adelante hablaremos de cómo estas estructuras se comunican entre sí y de qué forma influyen en nuestro contenido y finalmente se hablará de lagunas de las restricciones y problemas que existen actualmente con algunas de estas estructuras y componentes.

3.3.1 Nociones básicas

A continuación, se habla de algunos de los conceptos más relevantes a la hora de comprender el funcionamiento de una aplicación desarrollada a través de Unreal Engine y sus símiles en lenguajes de programación convencionales.

3.3.1.1 El concepto de nivel

Cada objeto o elemento con el que interactuamos dentro del juego o aplicación reside en un espacio, y este espacio es lo que se conoce como mapa o nivel (o a veces también llamado escenario), y que viene a ser el conjunto de objetos estáticos o mesh, luces, volúmenes, blueprints y otros componentes funcionales y de visualización y composición de escena. Podría definirse como el entorno 3D en el cual el desarrollador coloca objetos y crea funcionalidades dando lugar así al mundo virtual en el que se desarrollará la acción de nuestra aplicación [23].

Todo proyecto cuenta con al menos un nivel o mapa inicial, pudiendo tener una gran cantidad de ellos, de esta forma podemos diseñar distintos niveles e ir pasando de uno a otro según se desarrolle la acción de nuestro contenido. De esta forma conseguimos modularizar las distintas partes que contendrá nuestro proyecto. El nivel, por tanto, actuaría como una especie de clase principal o main en lenguajes como java o C.

3.3.1.2 El concepto de Actor

Un actor [24] es cualquier objeto que ha sido colocado en el nivel y que permite ser escalado, rotado y cambiado de posición dentro del mismo, por tanto, la clase ‘Actor’ es una clase genérica de objetos que actúa de forma parecida a la clase ‘Object’ en java, teniendo que en Unreal Engine 4 la clase ancestro sería AActor en C++.

Al iniciar el nivel se crean o destruyen instancias de estos y se representan de forma visual dando lugar a la creación del entorno virtual, estos actores, que pueden ser de muchos tipos como por ejemplo un cubo o una luz direccional, tienen a su vez múltiples propiedades dependiendo del tipo de actor que sean y actuando de forma diferente en el entorno según su especialización, una luz proyectará

sombras y generará una iluminación determinada por unas propiedades modificables en otros actores circundantes también situados en el nivel, mientras que un cubo podrá actuar como elemento de colisión con el jugador y para definir estructuras visuales.

Dentro de Unreal Engine podemos destacar los siguientes actores básicos:

- **Player Start**
  - Define la posición de salida inicial de un jugador en el nivel.
- **Componente de audio**
  - Permite ejecutar un archivo de audio en un lugar del nivel determinado.
- **Luces** (varios tipos)
  - Generan iluminación en los distintos actores del nivel, poseen parámetros regulables, existen distintos tipos de luces y parámetros de iluminación.
- **Partículas**
  - Actores formados generalmente por pequeños objetos u otros elementos gráficos en 2 o 3 dimensiones que se rigen por unas físicas determinadas definidas por el usuario.
- **Actor Blueprint**
  - Consiste en un actor que posee funcionalidades propias y personalizadas creadas por el desarrollador, se tratará este tipo de actor más adelante.

3.3.1.3 Blueprints y Actor Blueprints

Unreal Engine utiliza un sistema de scripting orientado a objetos al igual que otros motores gráficos como Unity (que utiliza C# para la generación de scripts), pero con la diferencia de que se trata de un **sistema visual basado en nodos** [16] que pueden conectarse y comunicarse entre sí para formar bloques de código más grandes, la gran ventaja de este sistema es que podemos hacer todo esto a través de una interfaz gráfica sencilla.

Existen muchos tipos de *blueprints* que vendrían a ser el equivalente, en Java y otros lenguajes orientados a objetos, a una clase, es decir, pueden contener variables de muchos tipos (primitivos y complejos definidos por el usuario), se pueden definir funciones y eventos propios y pueden comunicarse con otros *blueprints* y componentes.

Los *Actor Blueprint* son *blueprints* especializados que actúan como actores, es decir, pueden colocarse sobre el nivel, permiten rotación, escalado... Pueden llevar asociados varios componentes visuales que a su vez son actores en el juego, por ejemplo, podemos tener un *blueprint* que contenga una malla/mesh de un semáforo con 3 actores de iluminación para cada una de las luces roja, ámbar y verde que además vaya activando y desactivando cada una de las 3 luces según pasa el tiempo, es decir, posee una función que hace que por sí solo vaya mostrando una de las 3 luces.

3.3.1.4 Blueprint del nivel

El blueprint del nivel [25] es un tipo especial de blueprint que se comporta a modo de clase global y que actúa sobre un nivel específico, es decir, **cada nivel posee su propio blueprint de nivel**. A través de este tipo de blueprint podemos conocer y acceder a todos aquellos actores que se encuentran sobre el nivel, ya que estos se tratan de forma similar a instancias de objetos en una clase Java.

3.3.1.5 Nodos

Nos referimos con nodos a aquellos fragmentos de código encapsulados en pequeñas cajitas gráficas que únicamente tienen efecto cuando nuestro proyecto de Unreal Engine 4 ha sido creado para trabajar con *blueprints*.

3.3.1.6 Eventos

Los eventos [26] son nodos especiales y una de las funcionalidades más utilizadas, que se utilizan para construir la funcionalidad de nuestro proyecto y **modularizar el sistema** y se utilizan para permitir a los *blueprints* responder a distintos eventos que ocurran durante el proceso de juego de nuestra aplicación, como por ejemplo saber cuándo empieza este a ejecutarse, cuando se ha cambiado de nivel, si un actor está siendo tocado por otro… etc.

3.3.1.7 Entradas

Las entradas [27] son unas funciones especiales del motor encargadas de transformar los datos de entrada de los distintos tipos de controladores, como el ratón y el teclado, en información que otros actores del juego pueden utilizar. Estas entradas pueden configurarse desde los ajustes del proyecto, pudiendo añadir o quitar entradas y pudiendo 'mapearlas', es decir, configurar un nombre para ese tipo de entrada y unos rangos de valores a los que afecta, por ejemplo, si queremos que la entrada del ratón genere datos de entrada en los ejes X e Y podemos definir esto mismo y en qué medida influye (más rápido, hacia adelante o hacia atrás…).

3.3.1.8 Compilación

Existen dos tipos de compilación, una es la compilación del código subyacente en el editor, bien estemos trabajando con blueprints o con código C++, y que actúa de forma similar a otros compiladores de código, teniendo además la opción de **compilar dentro de cada blueprint**, informándonos el motor de fallos o advertencias si las hubiere.

Por otro lado, tenemos el compilador gráfico o compilador de luz y entorno; cuando el desarrollador crea el entorno gráfico colocando actores dentro del nivel lo que ve a través del 'viewport' principal es una **pre visualización** del entorno virtual sin todos los detalles propios de un 'render' [28] como son la oclusión ambiental generada por los objetos, el cálculo de las sombras con respecto al terreno y otros actores…etc.

3.3.1.9 Plugins

Como ocurre con otros programas y herramientas, Unreal Engine 4 permite la posibilidad de añadir y crear plugins [29] concretos para determinados tipos de tareas que añaden funcionalidades extra, además la comunidad de Epic Games cuenta con una gran cantidad de ellos que, o bien vienen integrados en el motor, o que provienen de desarrolladores conocidos o pueden adquirirse en el Marketplace de Unreal, teniendo desde plugins para aplicaciones de integración de texturas como es el de '**Substance Painter'** como plugins para el manejo de dispositivos de realidad virtual.

3.3.2 Flujo de ejecución del motor

En este apartado se explica brevemente cuál es el flujo de juego que sigue el motor [30] a la hora de iniciar las herramientas del editor y cuando ejecutamos nuestra aplicación.

Para empezar, debemos distinguir entre los dos tipos de ejecución a grandes rasgos de los que dispone el motor. El primero se conoce como '**Editor mode**' y es el modo de ejecución por defecto, donde se crean instancias de los actores y de las clases principales necesarias después de la inicialización del motor, por tanto, el motor de inicia y empieza a funcionar de forma inmediata, sin esperar a la creación de otros actores y clases.

El segundo modo de ejecución se conoce como '**Standalone'** y es una forma de separar el contenido del editor, es decir, se crean los objetos necesarios en el nivel antes de la inicialización del motor, iniciándose la aplicación cuando se han creado los objetos *Game Mode* y *Game State* de los que hablamos en el apartado siguiente.

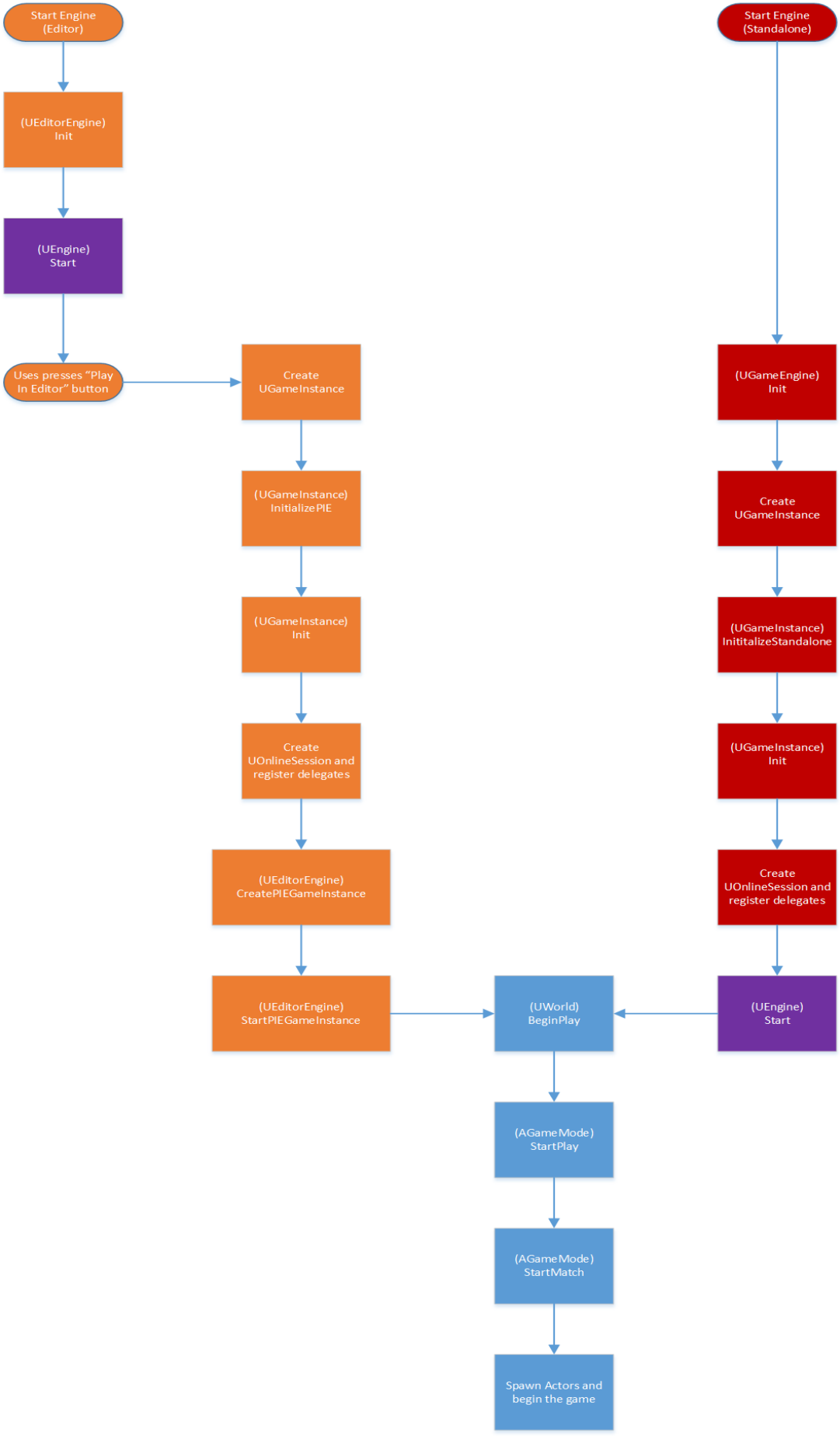


Figura 13: Flujo de juego desde que se inicia el motor de juego en Unreal Engine 4

3.3.3 Elementos básicos del framework

En esta sección se explican algunos elementos importantes del framework que se han considerado de gran interés a la hora de comprender el trabajo realizado y documentado en esta memoria.

3.3.3.1 Game Mode

El *Game Mode* [31] es una de las clases principales en el manejo de información de nuestra aplicación, es el encargado de manejar las 'reglas del juego', es decir, cuáles van a ser las leyes o reglas que definan nuestra aplicación. Esta clase deriva de la clase *AGameModeBase* desde la versión 4.14 y de la clase *6* en versiones anteriores, y en ella se encuentran algunas variables y funciones importantes:

- El número de jugadores y el máximo número de jugadores permitidos.
- Cómo entran los jugadores al juego (dónde aparecen dentro del entorno… etc.)
- Qué ocurre cuando el juego es pausado
- Si ha comenzado o no la aplicación

El ***Game Mode*** no es único en la aplicación, podemos tener, al igual que pasaba con los mapas o niveles, varios tipos de modos de juego (que sería su traducción al castellano) dependiendo de en qué nivel o zona dentro de un nivel nos encontremos. Esta clase **es única en juegos y aplicaciones multijugador y es definida por el servidor** (El servidor es el que controla el modo de juego) no siendo 'replicada' al resto de clientes del juego. Al iniciar la ejecución de una aplicación se crea una instancia de tipo *Game Mode Actor* que puede implementarse y extender sus funcionalidades.

De esta clase derivan algunos de los *blueprints* más importantes dentro del motor gráfico y que todo proyecto contiene:

- **Default Pawn Class**
  - Un 'pawn' [32] es la representación en 3D de un jugador o entidad que posee una IA en la aplicación, en un proyecto debemos crear o personalizar nuestro propio *pawn* redefiniendo algunas funciones, como por ejemplo todo el tratamiento de entradas. Esta clase no persiste entre niveles o mapas [2].

- **HUD Class**
  - Clase encargada de mostrar por defecto algunos elementos gráficos en pantalla relacionados directamente con el modo de juego [32], como por ejemplo la barra de salud de un jugador encima de su avatar de juego (que sería el *pawn*).

- **PlayerController Class**
  - El 'Player Controller' es la interfaz entre el avatar o *Pawn* y el usuario real que está utilizándolo [32]. Este tipo de clase debe especificar una 'cámara' desde la que el jugador verá el escenario cuando la aplicación esté en ejecución. Se tratará el tema de las cámaras en el apartado 3.3.3.4.

- **Game State Class**
  - Es la clase responsable de manejar toda aquella información que necesitan conocer todos los clientes conectados a una aplicación [31], como por ejemplo la lista de todos los jugadores conectados, las puntuaciones actualizadas de equipo…etc. Define el estado en que se encuentra la aplicación de forma global, no es relativa a un solo cliente o usuario.

Esta clase al contrario que pasa con la clase *Game Mode* existe en el servidor y es replicada a todos los clientes.

- **Player State Class**
  - Se crea una instancia de esta clase únicamente en el servidor (o en la aplicación del usuario si es una aplicación arcade) para cada jugador [2]. Estos estados se replican a cada uno de los demás clientes y contienen información relevante acerca de la conexión de cada jugador, como el nombre del jugador, la puntuación, la latencia…

3.3.3.3 Widgets

En Unreal Engine los widgets [33] son un **tipo especial de blueprints** que permiten ajustarse a la pantalla del usuario en las medidas que definamos y que sirven para representar normalmente elementos gráficos planos como puede ser un **menú o una pantalla de carga**. Al crear un widget se nos abrirá una ventana en la que podremos diseñar la apariencia del mismo, para ello contamos con varias áreas de trabajo; En la parte central tenemos el apartado visual que nos mostrará de forma gráfica lo que estamos diseñando.

En la parte izquierda del editor se encuentra la paleta de componentes que podemos añadir a nuestro widget, bastando con arrastrarlos hacia el panel central, además en esta parte también se nos muestra la jerarquía de elementos al igual que ocurre con los layout xml en Android studio y otras herramientas similares que trabajan con elementos jerárquicos.

En la parte baja se encuentra la barra de animación con la que podremos animar algunos componentes de este, en la parte derecha se encuentran las propiedades de cada uno de los componentes que forman parte de nuestro diseño y que han sido añadidos al nuestro widget desde la paleta de selección de componentes, desde esta pestaña podremos cambiar propiedades como el nombre, si debe ser tratado como variable o el tamaño que ocupa dentro del widget entre otras cosas.

Por último, arriba a la derecha se encuentran dos modos, el ‘modo diseño’ que es del que hemos estado hablando y luego está el modo de grafos o de funcionalidad, al seleccionar este modo pasamos a una nueva interfaz donde podremos decidir cómo actúan los componentes por ejemplo al recibir un evento de click sobre ellos. Una vez definidos hayamos definido nuestro widget podremos adherirlo a la pantalla de un jugador e interactuar con este cuando la aplicación lo requiera.

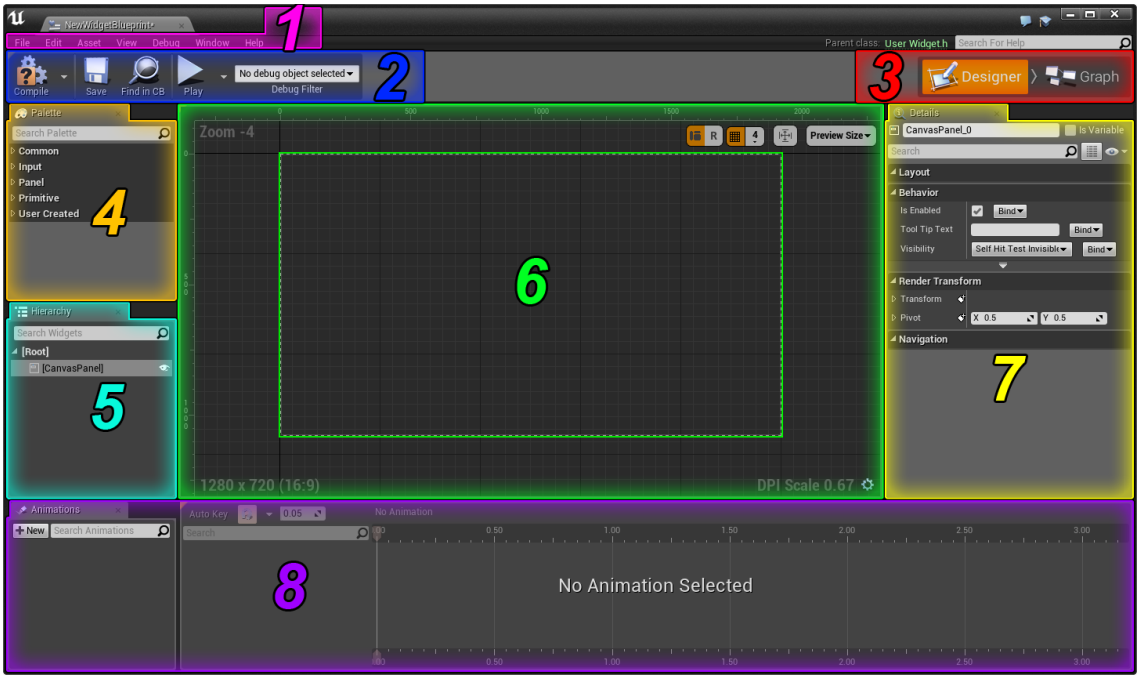


Figura 14: Interfaz de creación de widgets

3.3.3.4 Cámaras

Las cámaras representan el punto de vista desde la que el usuario ve parte del contenido de la aplicación. Existen múltiples tipos de cámaras según el objetivo de esta, pero en todos estos tipos su funcionamiento se basa en la captura de un fragmento del entorno basándose en unas propiedades definidas por el desarrollador, tales como el campo de visión o la posición que ocupa dentro del escenario.

Además, las cámaras son capaces de recoger información relativa a otros componentes de los que ya se ha hablado a lo largo de este capítulo, como son los efectos de post procesado y render que sirven para la configuración visual del entorno mediante parámetros editables por el desarrollador.

3.3.3.5 Event Dispatchers

Son componentes que cumplen las mismas funciones que un 'Listener' o manejador de eventos, es decir, que esperan a que un evento se produzca para reaccionar ante este y realizar una serie de acciones.

Los *Event Dispatcher* [2] tienen dos formas de actuación, de modo que en el componente en que se declaran actuarán como '**listeners'**, es decir, que manejarán el evento ocurrido en la clase en que se crea. Por otro lado, otro componente del motor puede ligar un evento propio a uno de estos manejadores de eventos, actuando como 'Open-mouth' o '**talkers'** en este otro componente, de modo que cuando se llame a un *event dispatcher* de un componente del juego desde otra clase, se ejecutarán la serie de acciones que hayamos decidido. En la figura 15 un componente define un *event dispatcher* y lo llama cuando se inicia (Evento BeginPlay).

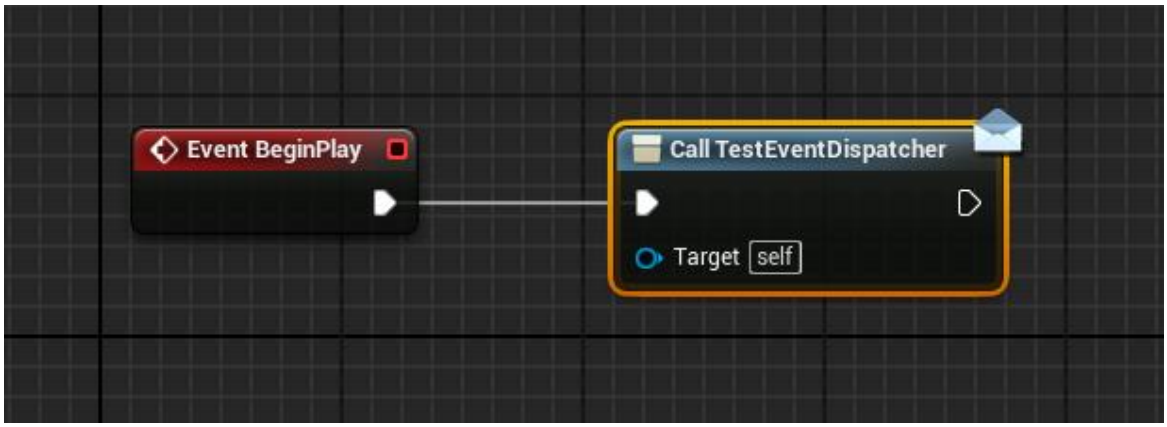


Figura 15: Se llama a un event dispatcher definido en una clase blueprint

En la Figura 16 un componente *liga* el event dispatcher del componente que lo declara a un evento local de nombre 'Do something' y que ejecutará otro evento local del mismo componente 'Some Action' cuando el componente que declara al event dispatcher le llame.

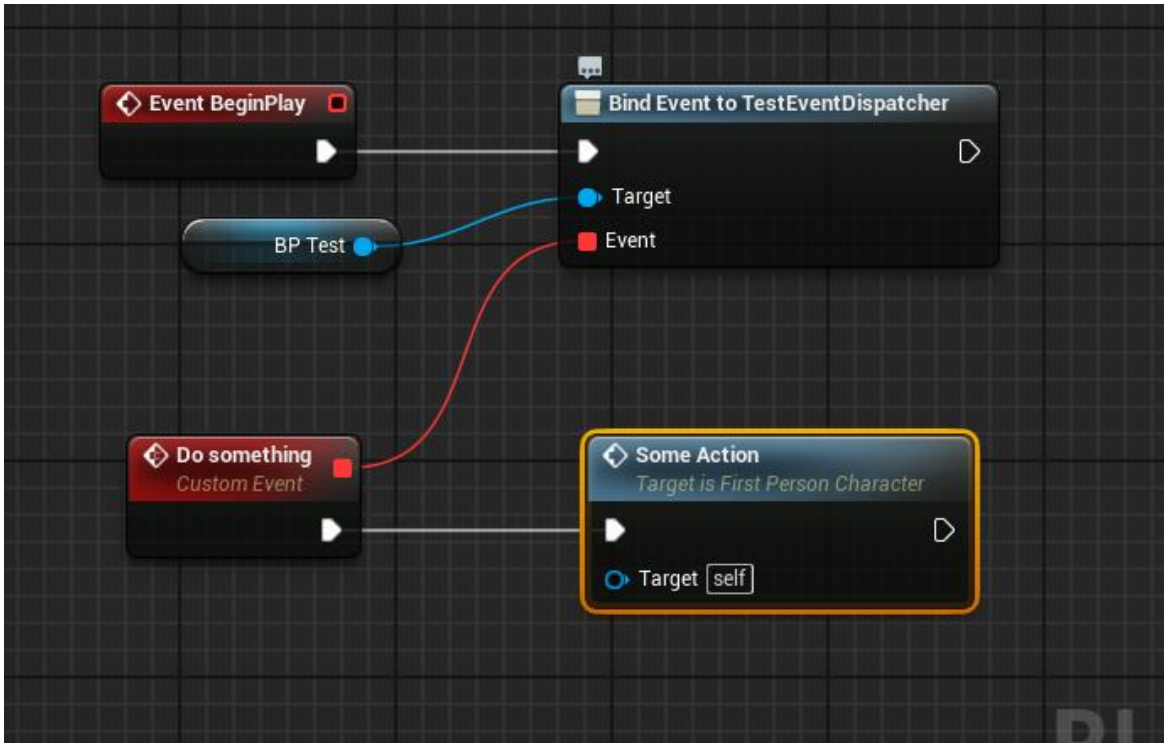


Figura 16: El componente First Person Character liga un event dispatcher de la clase BP Test

3.3.3.6 Funciones y Eventos

Como en la mayoría de lenguajes de **programación orientados a objetos**, Unreal Engine 4 que está basado en C++, también provee de esta característica a sus componentes cuando trabajamos con blueprints, de modo que podemos declarar funciones que toman parámetros como entradas y devuelven una salida mediante el uso de nodos de 'return'.

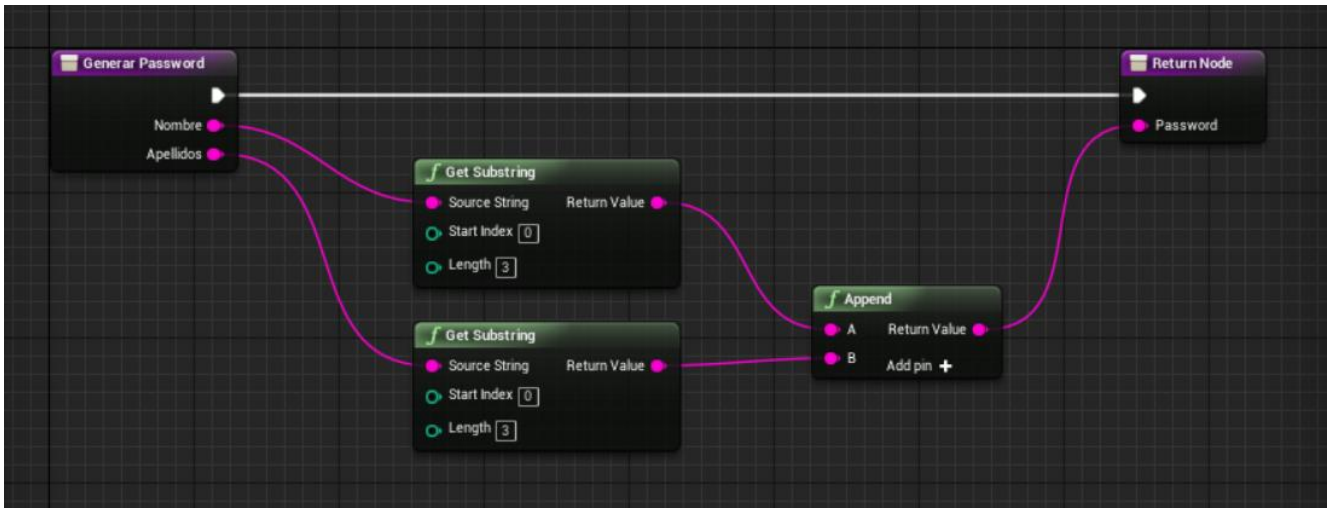


Figura 17: Función de generación de una contraseña que recibe dos parámetros de tipo String y devuelve otro de tipo String

También existen otro tipo de nodos llamados '**Eventos'**. Los eventos cumplen la misma tarea que una función corriente, puede recibir parámetros de diferentes tipos y son accesibles desde la propia clase que los define desde cualquier otro objeto que se comunique con esta, pero, al contrario que en las funciones, **no devuelven datos**.

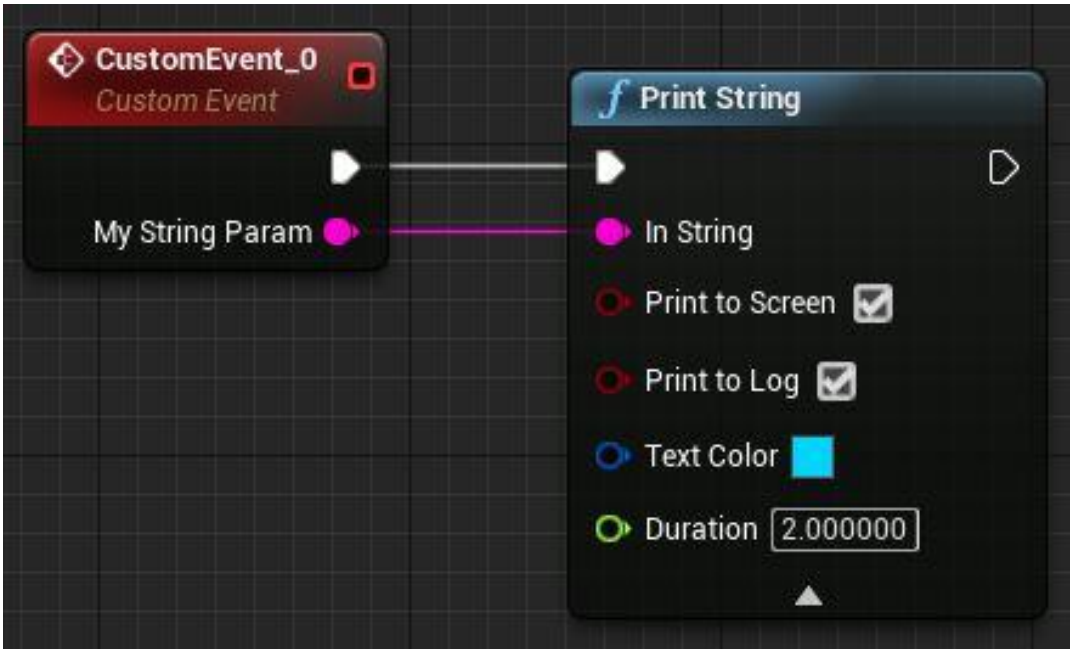


Figura 18: Evento que recibe una variable de tipo String como parámetro i la imprime por pantalla cuando este se lanza

7.1.2.3 Gestión de hilos de ejecución

Unreal Engine 4 gestiona la creación y ejecución de hilos de forma automática, por ejemplo, cuando se utiliza un nodo de *Secuencia* [35] lo que está haciendo el motor es crear hilos de ejecución paralelos con un **orden de prioridad** específico. Por ejemplo, en la figura 19 podemos ver un nodo de ejecución paralela teniendo prioridad descendiente, es decir, la primera salida del nodo es la que tiene mayor preferencia y la segunda sería la siguiente de forma consecutiva.

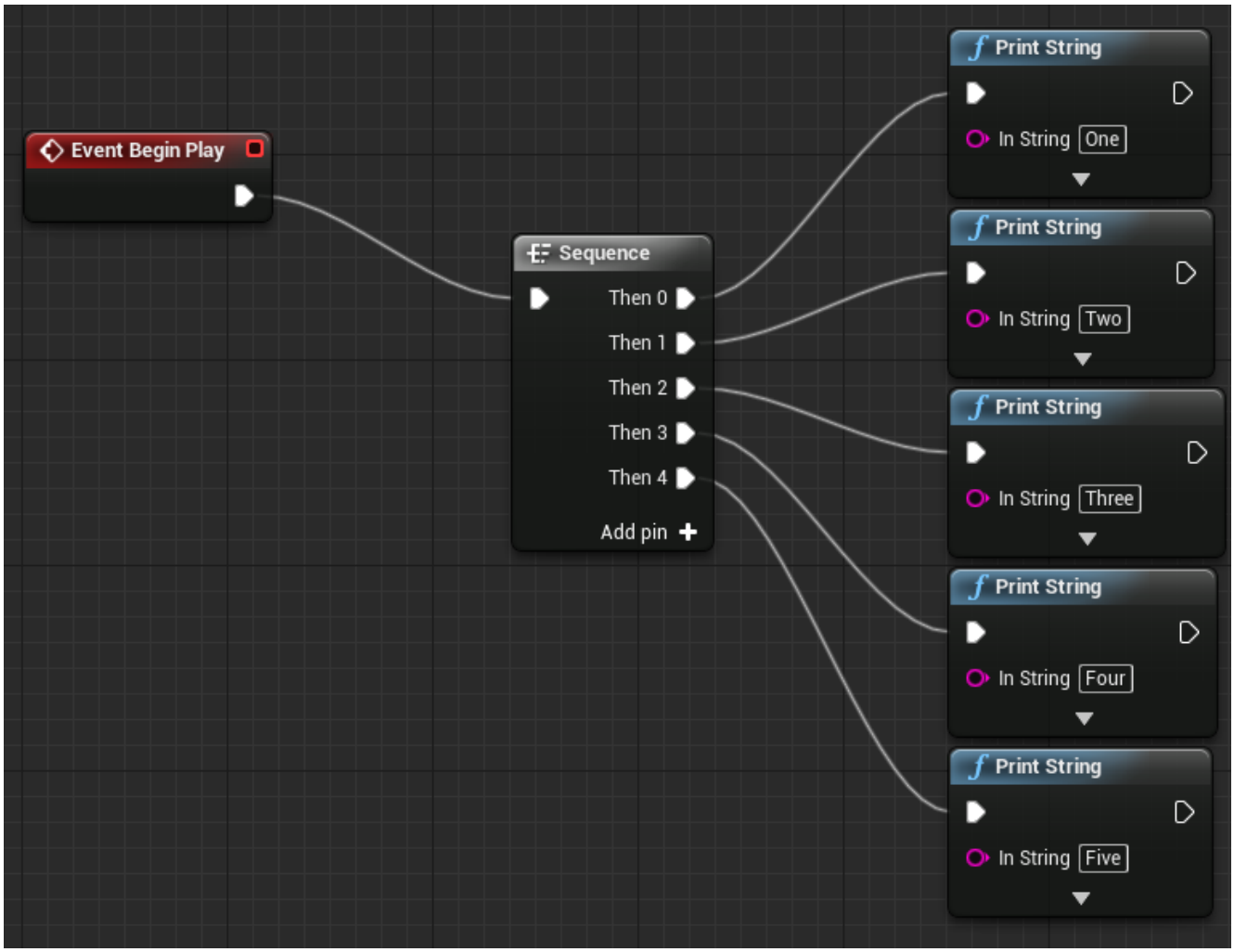


Figura 19: Ejemplo de nodo de secuencia en Unreal Engine 4

3.4 Estructura cliente-servidor

En este apartado se detalla el funcionamiento básico del modelo cliente-servidor que incorpora la herramienta Unreal Engine 4, así como conceptos y descripciones que pueden resultar de utilidad para la compresión de este trabajo.

3.4.1 Modelo

El motor Unreal Engine está construido pensando en el desarrollo de aplicaciones y juegos multijugador y por ello incorpora una base amplia e infraestructura que sigue un modelo cliente-servidor basado en la replicación de actores y datos que explicaremos en detalle a continuación, además lo que convierte a Unreal en una excelente herramienta para el desarrollo de este tipo de contenido es que puede configurarse fácilmente para poder hacer pruebas de forma local simulando que estamos conectados a un servidor remoto sin la necesidad de tenerlo físicamente, lo cual ayuda en gran medida al desarrollo de contenido multijugador.

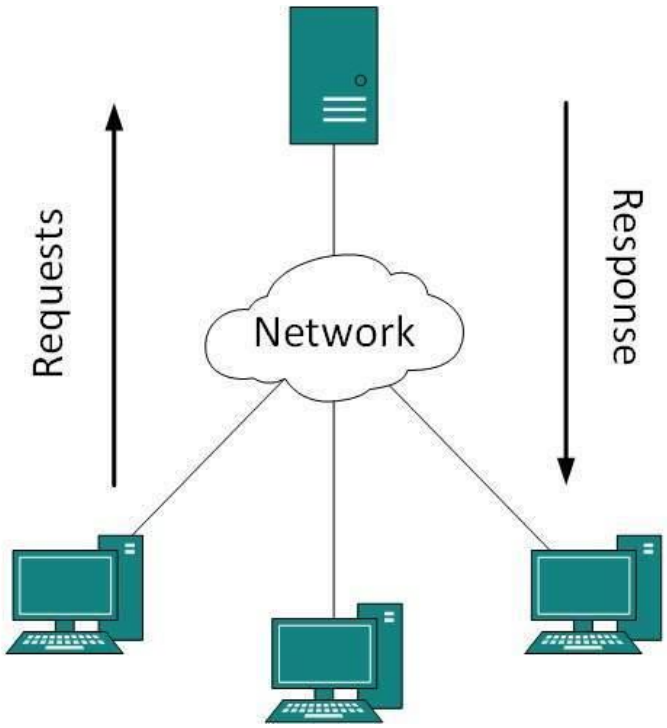


Figura 20: Modelo típico de la estructura cliente-servidor en red

Como en cualquier modelo cliente-servidor, este se basa en un dispositivo que actúa como servidor [36] y varios que actúan como clientes pudiéndose comunicar entre ellos a través de diferentes redes y protocolos. En Unreal Engine la lógica del modo de juego (‘*GameMode*’) es definido única y exclusivamente por el servidor actualizando a los demás clientes según se avanza en la aplicación y se guarda la información relativa en el ‘GameState’ y en otras clases y componentes importantes para el funcionamiento de esta comunicación.

Por tanto, el desarrollador siempre debe definir la lógica del contenido en el servidor y únicamente actualizar a los clientes con aquellos datos relevantes para estos, pudiendo el servidor actuar como un jugador más (Listen Server) o como un servidor que únicamente se encarga de mantener la actualización del estado en los clientes y procesar la lógica de la aplicación (Servidor dedicado). Como se comentó en el apartado 3.3.3.1, el servidor es el único que dispone una instancia de la clase

‘*GameMode’* y para comunicarse con los clientes acerca de cómo se está desarrollando la aplicación o en qué estado se encuentra se utiliza el ‘*GameState’* que sí se replica a los demás clientes.

3.4.2 Replicación

Para que el modelo cliente-servidor tenga sentido, necesitamos que algunos elementos y propiedades sean compartidos y actualizados al mismo tiempo en distintos dispositivos interconectados, estos elementos serán en su mayor parte actores de la aplicación. Un actor puede ser actualizado mediante llamadas de procedimiento remotas o RPC [37] o bien pueden actualizarse algunas de sus propiedades, la diferencia entre ambas es que las propiedades son actualizadas de forma inmediata y las llamadas remotas son replicadas solamente en la ejecución de las mismas.

Mediante esta técnica [38] podemos hacer que distintos elementos del juego sean visibles y compartidos por todos los usuarios que se encuentran en la sesión, pero siempre con el objetivo de que sea el servidor quien se encargue de dichas actualizaciones, además, Unreal Engine limita la capacidad que tienen los clientes de la aplicación de enviar datos hacia el servidor que es sabido que esto debe evitarse siempre que se pueda debido a que puede provocar agujeros de seguridad y a que rompe en general la estructura cliente-servidor.

De modo que en una aplicación podremos elegir el modo de actuación de nuestros eventos, teniendo 3 tipos [2]:

- **Eventos Multicast**
  - Estos eventos se caracterizan porque se replican desde la aplicación servidor hacia todos los demás clientes de la misma que se encuentren en la misma sesión (en el siguiente capítulo ahondaremos en el tema de las sesiones). Es el tipo de evento más utilizado en aplicaciones multicliente.

- **Eventos ‘Run on Server’**
  - Estos eventos son afectados por la restricción de transferencia de datos desde clientes a servidores, lo que hace este evento es llamar a un evento que únicamente se ejecutará en el servidor, como, por ejemplo, en un shooter un cliente informa al servidor de que su barra de salud se ha modificado al ser disparado por otro jugador. Estos eventos únicamente tienen efecto cuando se llaman desde el propio pawn del jugador.

- **Eventos ‘Run on Client’**
  - Se ejecutan en el propio cliente que llama al evento, es decir, de forma individual para cada cliente que llame a dicho evento. Se utilizan para funcionalidades que no afectan directamente al estado del juego/sesión compartida por todos los usuarios.

3.4.3 Sesiones

En este apartado se habla de algunas bases de uso y funcionamiento de las sesiones en el motor Unreal Engine, centrando las explicaciones en la creación de las mismas mediante blueprints y no a través de código C++.

Para que dos usuarios puedan compartir datos a través de la red ambos deben estar en una misma **sesión** [36]. Para ello al menos uno de los usuarios deberá actuar como servidor y crear una sesión y el resto de usuarios deberá unirse a dicha sesión. Como se comentaba al final del apartado 3.4.1, nuestro servidor puede ser un servidor dedicado y no permitir la jugabilidad como tal, sino que sólo se encarga de procesar la lógica de la aplicación y de controlar el flujo de juego para los demás clientes, o puede tratarse de un servidor de escucha o 'listen server', es decir, puede actuar como un cliente más de la aplicación y además procesa la lógica de juego y los datos de la red.

Al lado servidor se le conocerá como '*authority*' (autoridad en castellano) en la aplicación, de modo que podremos diferenciar dentro del propio código de la aplicación (en un blueprint de cualquier tipo, bien en el del nivel o en cualquier otro actor blueprint, en widgets o en código C++ si nuestro proyecto está construido con este…) qué parte del código va a ejecutar el servidor y qué parte los clientes, conocidos como lado 'Remote' (remoto en castellano), esto podemos hacerlo, en blueprints, con el nodo 'Switch has authority', de esta forma podemos separar las funcionalidades que contendrán nuestros usuarios cliente y por otro lado la lógica del servidor.

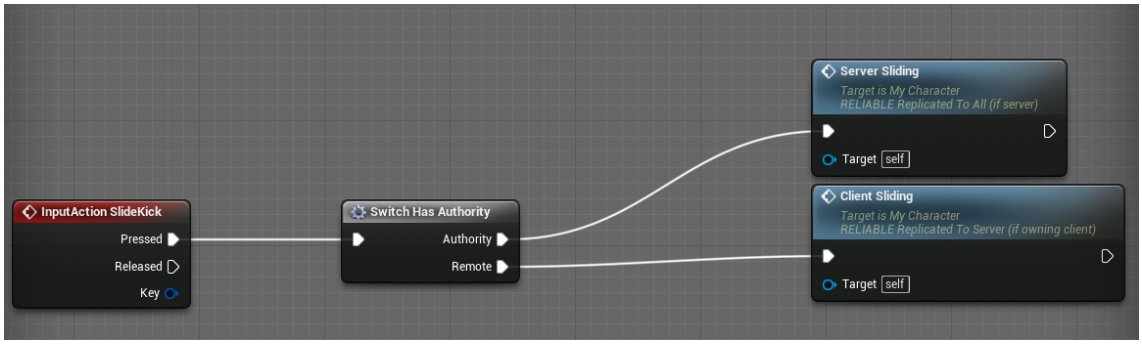


Figura 21: Utilización del nodo 'Switch has authority' para determinar qué evento debe ejecutar cada cliente de la aplicación

Para crear una sesión, en el lado del servidor deberá existir un nodo de creación de sesión, y para ello debemos pasarle como argumentos un objeto de tipo 'Player controller', podemos especificarle el número máximo de conexiones que soportará la sesión y si debe ser una sesión configurada mediante LAN. Si la sesión se crea con éxito el flujo pasará a través de la salida del conector 'On Success', sin embargo, si la creación de la sesión falla, el flujo pasará por el conector 'On Failure'.

Figura 22: Nodo de creación de sesión

Para unirse a una sesión creada es necesario utilizar el nodo 'Join Session', que recibe como argumentos un objeto de tipo *'Player Controller'* y una variable de tipo 'Array' que es un conjunto de Estructuras de '*Blueprint Session Result'*.

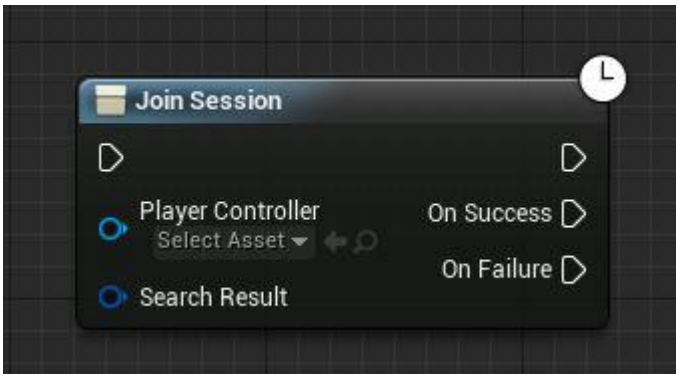


Figura 23: Nodo para unirse a una sesión

Para poder unirse a una sesión activa, antes es necesario buscarlas mediante el nodo '*Find Sessions*', que también recibe como argumentos un objeto de tipo '*Player Controller'*, el número máximo de sesiones a buscar y si debe realizar la búsqueda en conexiones LAN, además, da como resultado un array de estructuras '*Blueprint Session Result'* que podremos utilizar como parámetro de entrada a la hora de unirnos a una sesión activa. Si la búsqueda de sesiones tiene éxito, el flujo pasa a través del conector de salida 'On Success', en cambio sí se encuentra un fallo pasará a través del conector de salida 'On Failure'.

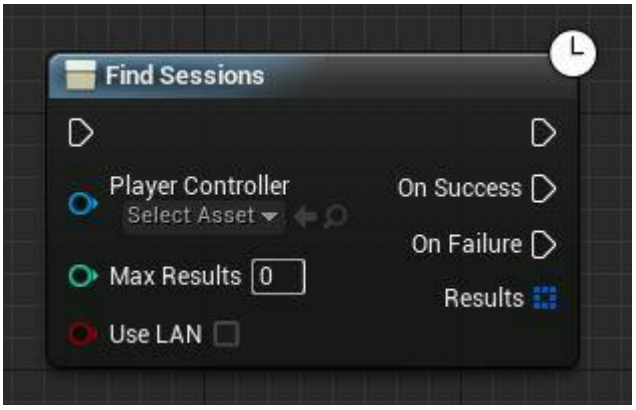


Figura 24: Nodo de búsqueda de sesiones en red

En este ejemplo puede verse una secuencia completa de cómo el lado servidor o host crea la sesión pasándole como argumento el objeto 'Player Controller' predeterminado en el proyecto y abriendo el nivel '*escenarioFabricacion'* con el parámetro '?listen' que hace que sea un servidor no dedicado y pueda actuar como un cliente más, mientras que en el lado Cliente (la parte de abajo) se une a una sesión previamente almacenada en un array de estructuras de tipo 'Blueprint Session Result'.

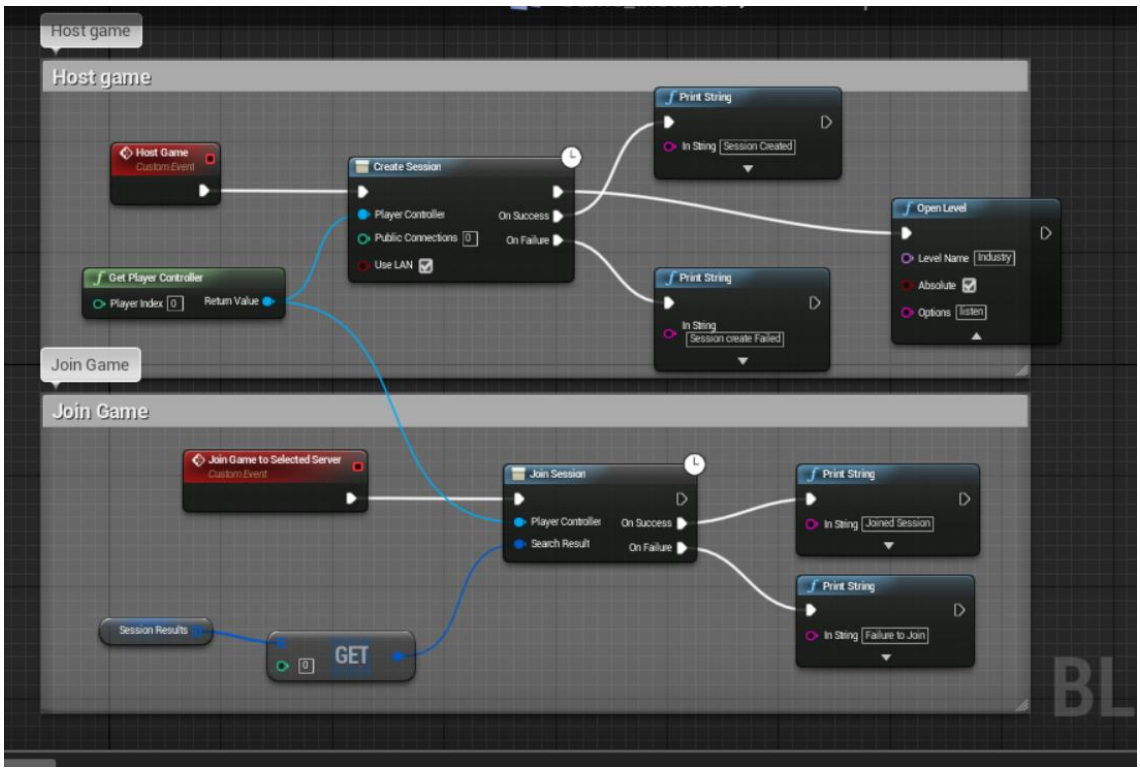


Figura 25: Ejemplo de creación de sesión y búsqueda y unión a la sesión

3.4.4 Configuración en Unreal Engine

Para hacer pruebas en local es necesario modificar una serie de parámetros y tener en cuenta algunas recomendaciones para poder hacer pruebas en local y verificar si estamos consiguiendo la aplicación multijugador objetivo sin tener que desplegar la aplicación cada vez que necesitemos probar que nuestra aplicación funciona correctamente.

En primer lugar, debemos buscar la ruta donde se aloja nuestro proyecto de Unreal Engine, una vez localizado debemos ir al fichero de configuración '…/MyProject/Config/DefaultEngine.ini' y colocar las siguientes líneas de texto al principio del documento:

[OnlineSubsystem]

DefaultPlatformService=Null

Una vez hecho esto evitaremos problemas de ejecución de funcionalidad que siga la estructura modelo cliente – servidor y podremos probar nuestro software de forma local sin ser necesario el despliegue de la aplicación.

Una vez hemos realizado este paso, el siguiente será configurar algunos parámetros del proyecto dentro del propio editor de Unreal, para ello lo abrimos en el editor y nos vamos a la sección de plugins. Tenemos que verificar que disponemos de los plugins de realidad virtual activados según la plataforma destino que tenga como objetivo nuestro proyecto. Los detalles de cada plugin se especifican en la sección '3.6.3.1 – Plugins necesarios'.

Una vez activamos los plugins debemos ir a la pestaña de 'Ejecutar' en el panel superior y cambiar el número de jugadores a 2 y ejecutar la aplicación mediante la forma 'New Editor Window (PIE).

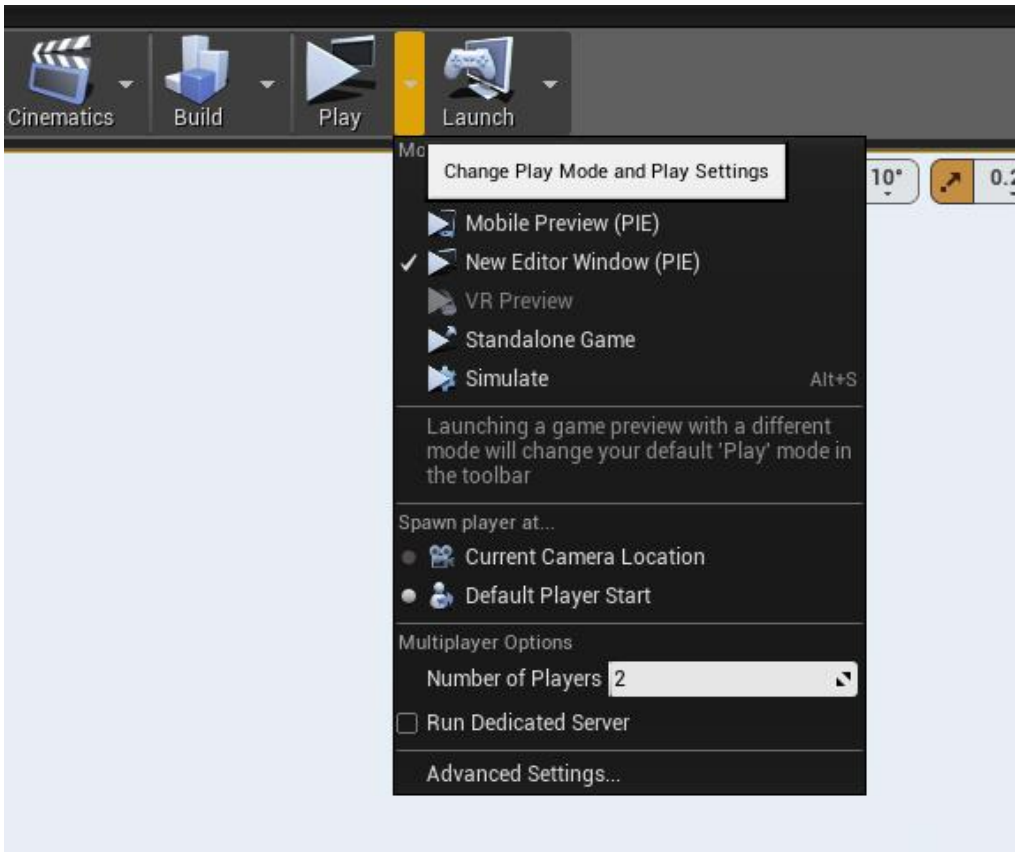


Figura 26: Menú de selección del modo de previsualización y opciones de multijugador

De esta forma el sistema simula una partida multijugador creando una sesión de forma interna y haciendo que exista un número de jugadores ficticio igual al indicado en las opciones de multijugador, siendo uno de ellos el servidor que actuará como jugador si no está marcada la opción de 'Run dedicated server'.

3.4.5 Problemas y restricciones del modelo

Dado que la tecnología y formas de programación de realidad virtual son relativamente nuevas y es recientemente soportada por Unreal Engine de forma oficial, habiendo hecho que este motor esté creciendo exponencialmente gracias a que muchos desarrolladores han descubierto las grandes ventajas que ofrece en este ámbito tan nuevo, todavía existen muchas restricciones y problemas asociados al desarrollo de este tipo de aplicaciones, una de ellas y que más relevancia ha tenido durante el desarrollo del trabajo que se describe en esta memoria, es el de no poder testear de forma fiable una aplicación multijugador de realidad virtual con tan sólo unas gafas de realidad virtual (en este caso unas gafas HTC VIVE), ya que ocurre que se superponen las pantallas de todos los jugadores activos, dando lugar a una imagen distorsionada en el dispositivo y haciendo que sea muy difícil de utilizar [39].

Para poder probar una aplicación que requiere movimiento, como es la desarrollada durante este trabajo, es necesario contar con un espacio adecuado para poder moverse con libertad y poder testear todos los aspectos de la aplicación, además los métodos tradicionales de debug como imprimir algunos parámetros por pantalla se hacen mucho más difíciles de visualizar cuando se llevan unas gafas de realidad virtual. En la aplicación desarrollada en este TFG, contamos además con dos tipos de usuario, el usuario servidor y el usuario cliente, el servidor o manager utilizará la aplicación como una aplicación de escritorio tradicional, es decir, podrá utilizarla mediante un equipo informático, ratón, teclado y una pantalla de dimensiones adecuadas, mientras que el usuario cliente o jugador deberá ponerse las gafas y utilizar la aplicación; dado que se trata de un panel de control en tiempo real que requiere dos usuarios y uno de ellos debe ponerse las gafas, lo que le aísla de poder ver lo que ocurre en otra pantalla u otro equipo, se hace complicado el poder depurar en tiempo real las funcionalidades de la aplicación.

3.5 Características de una aplicación de RV

En esta sección se describen algunas de las características más importantes que deben tener la mayoría de aplicaciones de realidad virtual y en concreto veremos algunos de los ajustes y recomendaciones para el desarrollo de las mismas dentro del motor Unreal Engine 4 y que se han seguido en la creación de este proyecto.

3.5.1 Escala del entorno y distancia de los objetos

Una de las primeras cosas que deben tenerse en cuenta es la escala del entorno y de los elementos del mismo en la aplicación, ya que en realidad virtual el usuario tiene la noción de altura, posición y profundidad dentro de este, por ello los objetos deben ser de un tamaño que vaya en relación a la altura y escala que vaya a tener el jugador dentro de la aplicación, por ejemplo, si queremos mostrar una oficina mediante técnicas de realidad virtual la mesa de esta debe percibirse como si fuera una mesa de tamaño real y no que parezca pequeña o muy grande, ya que el usuario es capaz de percibir estas anomalías con las gafas. Esto mismo pasa con la altura, si nuestro escenario tiene un suelo y hemos situado la cámara del jugador a una altura demasiado baja, cuando se ponga las gafas y 'entre'

en el entorno virtual, este va a percibir una sensación extraña ya que va a sentirse más bajito de lo que realmente es y por lo general no suele dar buenas sensaciones, afectando a la usabilidad y comodidad de uso de nuestra aplicación [2].

Por tanto, para conseguir un buen resultado en cuanto a escala, deberemos prestar especial cuidado a las medidas de los distintos actores y a la situación de la cámara dentro del nivel. En Unreal Engine además de poder modificar las propiedades de cada actor para ajustar nuestro contenido al formato de realidad virtual, disponemos de otras herramientas editables desde los 'Ajustes de mundo', desde los que podremos ajustar la escala global del entorno entre otras cosas.

3.5.2 Técnicas de movimiento

En este apartado comentaremos algunas de las formas más habituales [41] y que menos problemas causan a los usuarios a la hora de desplazarse en el entorno virtual de una aplicación de realidad virtual.

3.5.2.1 Tele portación

La tele portación es un método que hace que el usuario pueda recorrer grandes distancias de 'terreno virtual' transportándose mediante la técnica de tele portación, es decir, el usuario se desplaza de un punto a otro de forma casi instantánea sin pasar por puntos intermedios (dependiendo del estilo visual que queramos darle a nuestra aplicación). Este es uno de los métodos más comunes de movimiento en aplicaciones de realidad virtual con interacción ya que no resulta molesto al usuario y nos permite desplazarnos de una manera rápida y sencilla hacia cualquier parte del escenario a la que permita desplazarse la aplicación.

3.5.2.2 Movimiento natural

Mediante la utilización de sensores de captación de movimiento o 'tracking' un usuario puede moverse dentro del espacio virtual dentro de unos límites prefijados por el software o la aplicación concreta. Este modo es el que resulta más natural al usuario ya que la sensación de inmersión en ocasiones engaña al cerebro y hace que el usuario crea que está físicamente dentro de la aplicación, pudiendo moverse en el entorno virtual como lo hace en el mundo real.

3.5.2.3 Avance simulado

Este avance se consigue a través de diversas técnicas y consiste en implementar el movimiento del usuario a través de la pulsación de un joystick o de cualquier patrón de movimiento o pulsación de los controladores del jugador que interprete la aplicación como movimiento, por ejemplo, la forma de pulsación del botón de un mando o *controller* en un software pensado para trabajar con unas gafas HTC VIVE hace que el usuario se desplace por el escenario virtual de una determinada forma.

3.6 Requisitos para el desarrollo de aplicaciones de RV

En esta sección se explican algunas recomendaciones de hardware y software y también otros elementos necesarios para desplegar aplicaciones de realidad virtual a través de Unreal Engine 4.

3.6.1 Requisitos de software y hardware

Los requisitos necesarios para hacer funcionar de forma correcta una aplicación de realidad virtual no suelen ser los convencionales, en este apartado veremos cuáles son los requisitos hardware y software mínimos y necesarios tanto para el desarrollo de aplicaciones de realidad virtual como para la utilización de las mismas [2].

3.6.1.1 Hardware

Según la página oficial de *HTC VIVE* que son las gafas que se han utilizado en el desarrollo de esta aplicación, los requisitos de hardware mínimos son los siguientes según su página oficial [40]:

Procesador. Intel™ Core™ i5-4590 o AMD FX™ 8350 o superior

Memoria. Según la página son 4GB mínimos, pero para el desarrollo de aplicaciones de este tipo es recomendable disponer de al menos 8GB de RAM.

Gráfica. NVIDIA GTX 1060 o superior según la página oficial, para desarrollo es mejor disponer de al menos una NVIDIA GTX 1070, su equivalente de AMD o superior, siempre dependiendo del tipo de aplicación en concreto que vayamos a desarrollar.

Gafas. Unas gafas HTC VIVE o unas Oculus Rift + Touch controllers si vamos a desarrollar aplicaciones de escritorio que permitan funcionalidades avanzadas (uso de *controllers* y posicionamiento) o un dispositivo móvil con Android o unas gafas Samsung Gear VR dependiendo de la plataforma objetivo de nuestra aplicación.

3.6.1.2 Software

Los requisitos de software implican desde el sistema operativo hasta el software propio de los desarrolladores de las gafas entre otros. En esta sección no nos centraremos en el software necesario para hacer funcionar cada uno de los distintos tipos de gafas de realidad virtual, ya que esto se explicaba al principio del capítulo 3.

Una recomendación muy a tener en cuenta cuando estamos desarrollando aplicaciones de escritorio destinadas a funcionar en dispositivos de realidad virtual, es que estas funcionan mucho mejor con la tecnología **DirectX 12** y la diferencia con las versiones anteriores de DirectX se hace mucho más evidente en aplicaciones de realidad virtual.

En la figura 26 podemos observar una gráfica realizada por *Anandtech* que mide el tiempo que tarda la GPU, en este caso una NVIDIA 1080, en dibujar un fotograma y se puede ver como a medida que se estabiliza el test el tiempo de dibujado con DirectX12 es mucho menor que con la versión anterior de esta tecnología, DirectX11.

DirectX12 únicamente está disponible con el sistema operativo de Windows 10, por lo que es recomendable tanto desarrollar sobre este sistema como ejecutar las aplicaciones si lo que queremos conseguir es el mñaximo rendimiento.

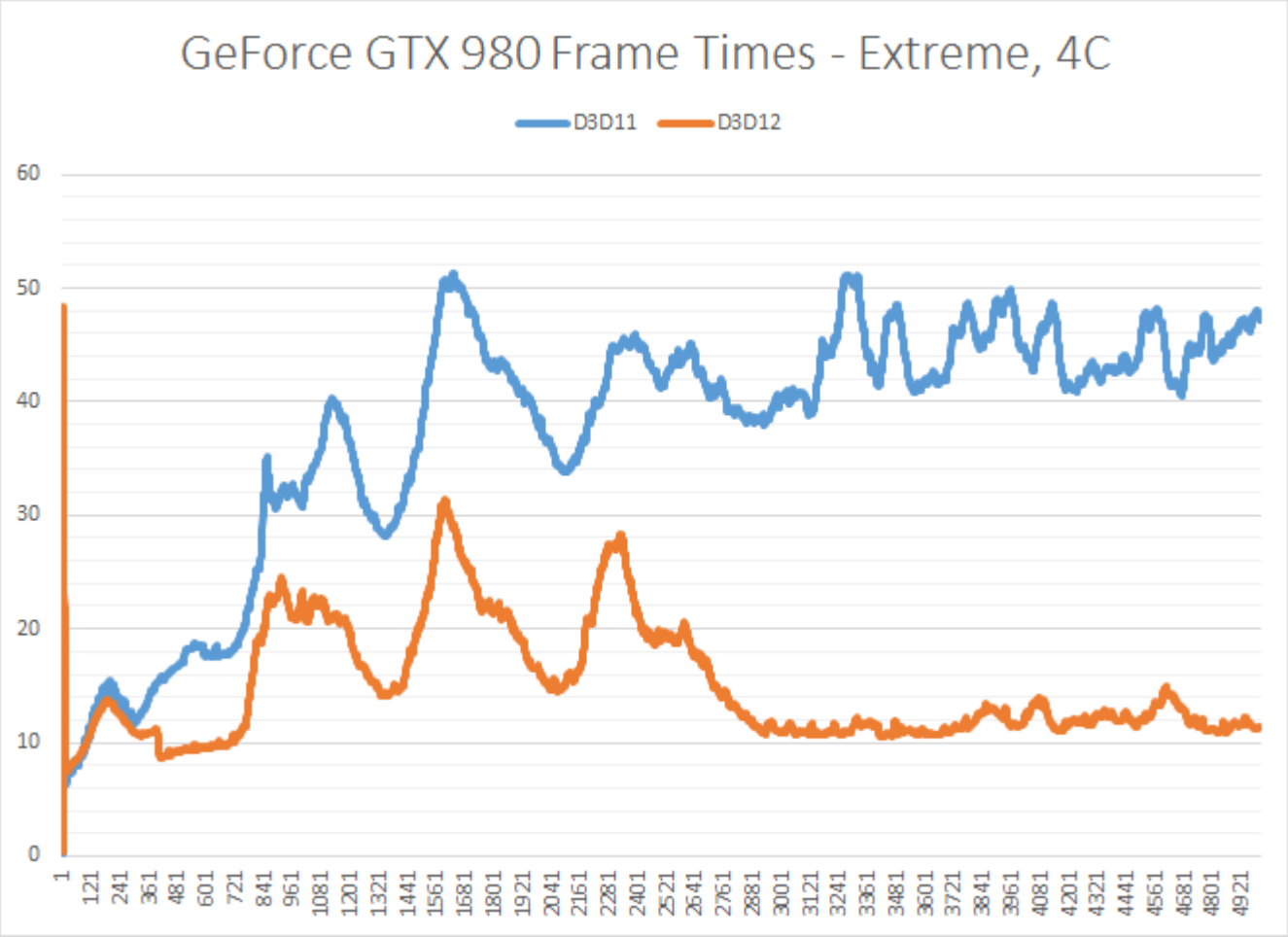


Figura 27: Gráfica con los tiempos de dibujado de frames en utilizando las tecnologías DirectX11 y DirectX12

3.6.2 Despliegue de aplicaciones de realidad virtual

En este apartado se comentan algunas de las buenas prácticas y otros elementos necesarios y de utilidad que sirven a la hora de desplegar correctamente una aplicación realizada en el motor Unreal Engine 4 y destinada a funcionar en un dispositivo de realidad virtual.

3.6.2.1 Plugins necesarios

Una de las grandes ventajas del motor Unreal Engine 4 es que dispone de varias librerías y utilidades que permiten saltarse la barrera de tener que programar a mano el empaquetado y configuración de aplicaciones de realidad virtual, la mayoría de estos plugins provienen de los propios editores y creadores del hardware que permite visualizar contenido en realidad virtual, dependiendo de si nuestra aplicación va a funcionar en la plataforma de móviles Samsung Gear VR, si va a utilizar Steam VR en el caso de que el hardware destino sea HTC VIVE, o incluso destinadas a la propia plataforma de Oculus Rift para escritorio.

Estas utilidades no sólo nos permiten desplegar nuestra aplicación configurada para la plataforma de realidad virtual que nos interese, sino que además nos permite probar las funcionalidades dentro del propio editor del motor gráfico y sin tener que preocuparnos por muchos detalles que de otra forma deberíamos programas a través de las APIs de los editores correspondientes. Las bibliotecas de esta índole se encuentran en la categoría de 'Virtual Reality' en la sección de plugins del proyecto dentro del editor de Unreal. A continuación, se muestran los plugins que existen actualmente en los motores de las versiones más recientes de Unreal Engine 4 y que son:

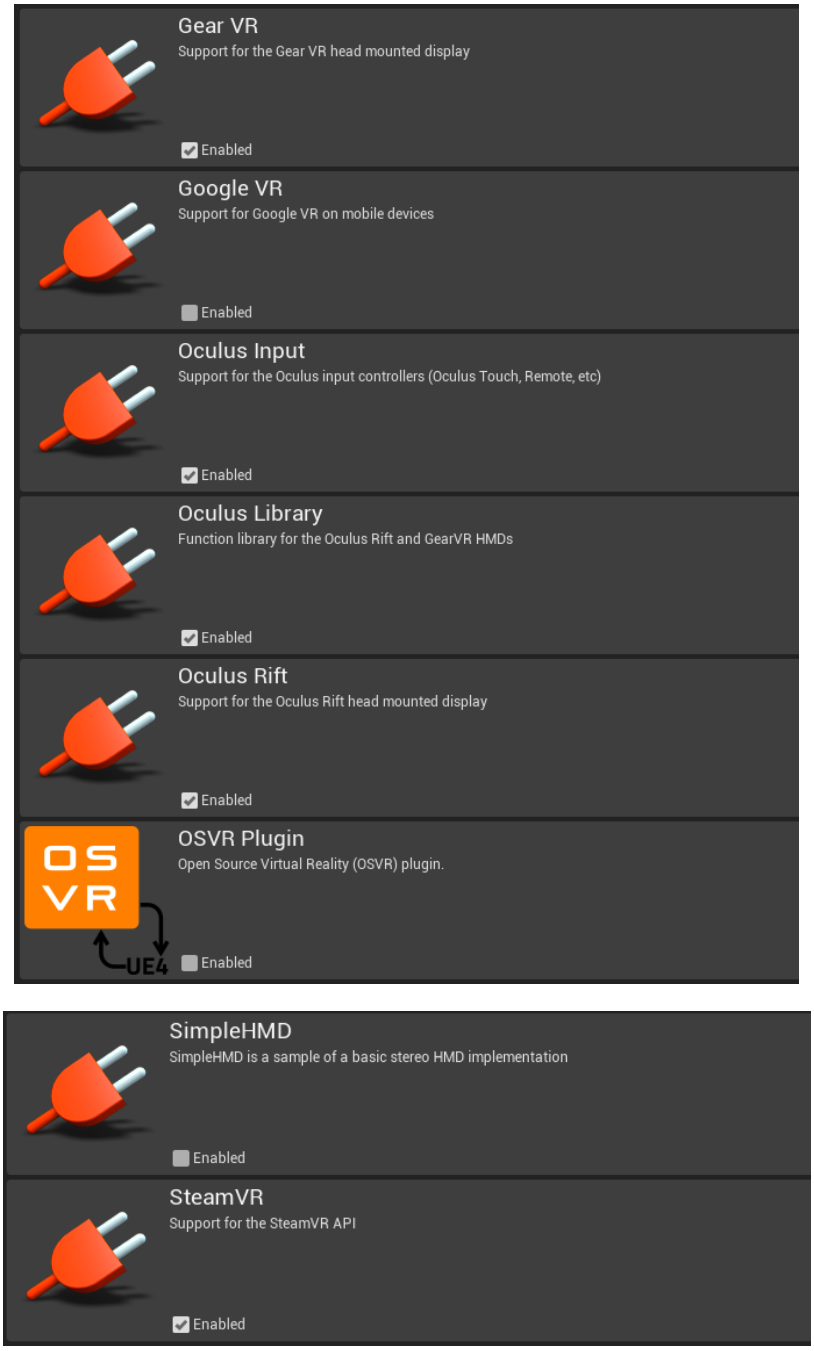


Figura 28: Menú de plugins de realidad virtual de Unreal Engine 4

Por ejemplo, para una aplicación destinada a la plataforma Samsung Gear VR es necesario activar el plugin 'Gear VR', ya que esto hará que al desplegar nuestra aplicación el proyecto se configure automáticamente para que se integre con el software y el hardware de Samsung, e igual pasa con el resto de plugins y plataformas.

3.7 Problemas asociados a la Realidad Virtual

En este apartado veremos cuáles son los problemas de usabilidad y de uso más frecuentes con los que se encuentran los usuarios a la hora de utilizar los distintos tipos de aplicaciones realidad virtual que existen actualmente en el mercado, explicando cómo podemos evitarlos en la medida de lo posible y qué debemos tener en cuenta de cara a la experiencia de usuario cuando desarrollamos una aplicación de realidad virtual.

3.7.1 Problemas físicos

Un usuario que pruebe una experiencia de realidad virtual puede sentir una sensación agradable y sentirse cómodo dentro del entorno virtual, sin embargo, dependiendo del tipo de experiencia, el usuario puede encontrarse con diversos problemas físicos. Estos problemas suelen ser provocados porque el cerebro humano no está acostumbrado a manejarse en un entorno que parece real, pero en el que no se maneja como en el mundo real, experimentando una sensación que puede ocasionar desde ligeros síntomas como mareos, fatiga visual, desorientación o pérdida del equilibrio hasta casos extremos como sufrir convulsiones o ataques al corazón que normalmente se derivan del uso de algunas experiencias de terror verdaderamente inmersivas [41].

Dado que se trata de una tecnología muy reciente se desconocen con certeza todos los síntomas y posibles problemas que puedan ocasionar las gafas de realidad virtual, por ello los fabricantes de gafas de realidad virtual como son Oculus Rift y HTC VIVE entre otros, dan algunos consejos para utilizar correctamente este tipo de dispositivos, las más destacadas son:

- Tomar un descanso cada 10 o 15 minutos por cada 30 minutos de uso.
- Si un usuario no se encuentra cómodo probando una experiencia virtual es muy recomendable que no continúe probándola.
- La edad mínima recomendada es de 13 años.

3.7.2 Problemas económicos

La realidad virtual es una tecnología que en la actualidad se encuentra al alcance de muy pocos debido a las exigencias de hardware y software que requieren ya necesitan un ordenador potente para funcionar correctamente y un dispositivo que cumple dichas características **suele rondar los 1.000 €** en la actualidad, lo que no está al alcance de todo el mundo. Al precio del hardware mínimo necesario hay que sumarle el propio precio de las gafas de realidad virtual que en el caso de las gafas **HTC VIVE son unos 899 €** y las **Oculus Rift + Touch controllers ronda los 708 €** sin envío incluido según sus páginas oficiales (precios consultados a día 26/06/2017).

Por si el precio de la tecnología no fuera suficiente, además es necesario contar con un espacio de juego adecuado para muchas de las experiencias disponibles para estos dispositivos, siendo recomendado para el caso de las HTC VIVE un espacio mínimo de 1,5 x 2 metros.

4. PLANIFICACIÓN DEL PROYECTO

4.1 Resumen del proyecto

A continuación, se detalla el plan de proyecto seguido durante la implementación del software sobre el que se basa este trabajo, explicando en qué consiste el proyecto, cuál es el plan de proceso y cómo se ha gestionado.

4.1.1 Propósito, alcance y objetivos

Este proyecto consiste en desarrollar un panel de control que permita el control y la monitorización de otro usuario en un entorno de realidad virtual simulando un escenario de fabricación de piezas de carrocería, además la aplicación guardará los datos relevantes de las acciones que hayan tenido lugar durante la ejecución de la aplicación.

La herramienta surge para dar respuesta a la necesidad de añadir funcionalidad para monitorizar a un jugador en un entorno de realidad virtual de prevención de riesgos laborales y que la aplicación pueda utilizarse también como una herramienta de formación laboral que registre datos relevantes del paso del jugador por el escenario virtual.

4.1.2 Suposiciones y restricciones

La aplicación se desarrolla a partir de un software previamente construido del que se omitirá gran parte de su funcionamiento, explicando en detalle sólo aquellas partes más directas y relevantes para la comprensión de la herramienta desarrollada durante este TFG.

4.1.3 Costes del proyecto

En este apartado se especifican los costes estimados del proyecto en dos categorías: costes de hardware y costes de software.

4.1.3.1 Costes Hardware

Los costes hardware para un funcionamiento correcto de la aplicación se basan en unas recomendaciones mínimas que deben tener los dispositivos en los que vaya a ejecutarse la aplicación, teniendo dos PCs uno de ellos ejecutará la aplicación en realidad virtual y por lo tanto deberá tener mayores capacidades de procesado y renderizado gráfico, y el otro visualizará únicamente el panel de control, estos requisitos mínimos son:

Dispositivo 1 – manager

El dispositivo del manager ejecutará el panel de control desde el que se controlará al otro jugador en el entorno de realidad virtual.

Requisitos mínimos:

- Procesador Intel i5-4590 o superior
- Tarjeta gráfica NVIDIA GTX 1060 con 2GB de dedicados o superior
- 4GB de memoria RAM
- Puertos ethernet o tarjeta de red con conexión wifi
- Al menos 64 GB de espacio libre
- Windows 10 64 bits

Costes del equipo adquirido en el proyecto:

MSI GT62VR 7RE-246XES Intel Core i7-7700HQ 16GB RAM 1TB+256SSD Gráfica GTX1070 15.6 pulgadas	1655€

Dispositivo 2 – jugador

El jugador dos, además de disponer de un PC deberá también contar con un casco de realidad virtual HTC VIVE con dos controladores y disponer, además, de un espacio mínimo 2 x 1.5 metros.

Requisitos mínimos:

- Procesador Intel i7-4770 o superior
- Tarjeta gráfica NVIDIA GTX 970 con al menos 4GB de memoria
- 8GB de memoria RAM
- Al menos 1 conector HDMI a mayores de la pantalla principal y dos puertos USB 3.0
- Puertos ethernet o tarjeta de red con conexión wifi
- Al menos 64 GB de espacio libre
- Windows 10 64 bits

Costes del equipo adquirido en el proyecto:

MSI GT62VR 7RE-246XES Intel Core i7-7700HQ 16GB RAM 1TB+256SSD Gráfica GTX1070 15.6 pulgadas	1655€
HTC Vive Gafas de Realidad Virtual HMD/Visor x1 Controladores x2 Base Station x2	919€
TOTAL COSTES:	**2574€**

4.1.3.2 Costes de software

Unreal Engine que ha sido la principal herramienta de software con la que se ha trabajado, junto con Visual Studio 2015, es gratuita siempre y cuando no se obtenga un beneficio trimestral por proyectos desarrollados superior a los 3.000€, en cuyo caso se debe abonar un 5% del total del beneficio a la compañía Epic Games. Aparte de esta herramienta también se han utilizado otros programas de diseño como Adobe Photoshop y herramientas de modelado 3D como Cinema 4D.

Los costes del software empleado en el proyecto son:

Adobe Photoshop CC 2017	12,90€/mes x 4 meses = 51,6 €
Cinema 4D R18	Three month license: 600€
TOTAL COSTES	**651,6€**

4.1.3.3 Costes de personal

El desarrollo de este proyecto requiere de uno o varios profesionales que posean conocimientos en diversas áreas para cumplir con los objetivos del proyecto, son necesarios conocimientos en:

- Programación.
- Manejo avanzado del motor Unreal Engine 4 en la versión 4.13.2 o superior.
- Modelado de objetos 3D preparados para videojuegos e integración dentro del motor Unreal Engine 4.
- Animación y texturizado de modelos 3D e integración dentro del motor Unreal Engine 4.
- Diseño gráfico de elementos del juego.
- Experiencia de usuario en aplicaciones de realidad virtual.

Por tanto, los costes totales teniendo en cuenta el hardware, el software y el salario medio del trabajador que reúne los conocimientos requeridos para el desarrollo del proyecto son de 20.000€ anuales basado en el salario medio anual de un responsable de proyecto titulado en Ingeniería Informática, pero dado que el proyecto requiere conocimientos en otras disciplinas como el modelado, texturizado y animación de objetos 3D y el diseño gráfico de interfaces y otros elementos gráficos además del manejo y programación en Unreal Engine 4, además de ser necesarios conocimientos sobre tecnologías de realidad virtual y su desarrollo, el precio asciende a **17€ la hora** atendiendo a la demanda del mercado.

Sumando las horas que se invertirán en el proyecto, el coste de personal sería de:

100 días de duración del proyecto * 3 horas diarias * 17/hora = 5.100€

Costes de hardware	**4.229€**
Costes de software	**651,6€**
Costes de personal	**5.100€**

TOTAL COSTES DEL PROYECTO	**9.980,6€**

4.2 Metodología del proyecto

En esta sección se detalla tanto la metodología y etapas básicas que compondrán el proyecto como también de los distintos planes y actividades que compondrán la planificación y desarrollo de la aplicación.

4.2.1 Metodología del proyecto

Se ha escogido una metodología RUP (Proceso Unificado) para el desarrollo de este trabajo incorporándole algunas etapas de investigación previas de cara al posterior desarrollo de algunas partes del software.

El motivo por el cual se ha escogido esta metodología a la hora de realizar el trabajo descrito en esta memoria es que, dado que el proyecto se basa en gran medida en investigación de técnicas y herramientas dentro un motor gráfico y que además se trabaja sobre un software previo del que se desconocen partes del código y del funcionamiento de los componentes, los requisitos del sistema y el diseño general de la aplicación puede ir variando según avanza el desarrollo del proyecto, y por ello se ha considerado que una metodología iterativa dirigida por los casos de uso y centrada en la arquitectura como RUP constituye una opción más que adecuada.

RUP es una metodología que reúne algunas de las siguientes características:

- **UML**: Se utilizará UML como lenguaje de modelado.
- **Dirigido por los casos de uso**: Estará dirigido por los casos de uso, es decir, cobra especial importancia qué es lo que demanda el usuario o los usuarios finales del sistema. Por este motivo se revisan los modelos y artefactos del proyecto en cada iteración haciendo que el diseño del sistema sea acorde a los objetivos finales del usuario.
- **Centrado en la arquitectura**: El proceso estará centrado en la arquitectura.
- **Iterativo e incremental**: Será iterativo e incremental.
- **Enfocado en los riesgos del sistema**: Controlando y anticipándose a los posibles riesgos que puedan surgir durante el desarrollo del proyecto, identificándolos y llevando un seguimiento de alta prioridad en cada una de las iteraciones del ciclo de vida del proyecto.

Al tratarse de un desarrollo individual, será el propio alumno el que asuma los roles de analista, desarrollador, tester y manager, de modo que todos los artefactos del proyecto serán elaborados por la misma persona y verificados en la medida de los posible por los tutores del proyecto cuando corresponda.

La metodología RUP sigue un ciclo de vida basado en el desarrollo en espiral [43] haciendo que cada etapa del proyecto cuente con una o varias iteraciones, teniendo además 4 fases que son las fases de **Inicio**, **Elaboración**, **Construcción** y **Transición** que se explican en el siguiente apartado. Además, la etapa de inicio ha sido modificada para que incluya varias etapas de investigación que serán necesarias a la hora de implementar el software y se ha intentado agilizar lo máximo posible esta etapa de acuerdo con los tiempos relativos estimados de las fases que marca RUP.

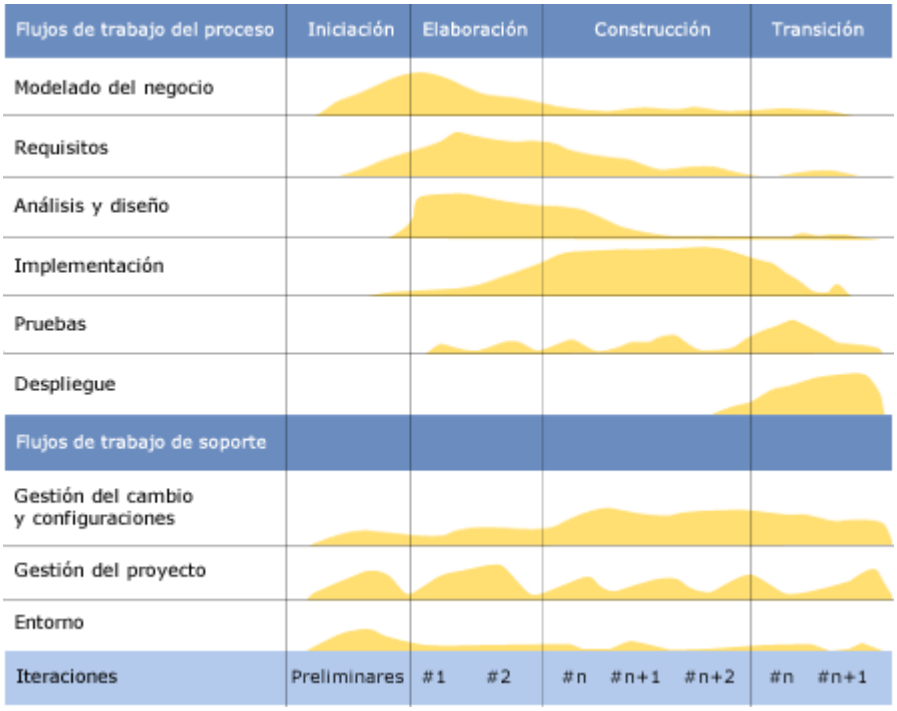


Tabla 1: Fases del proceso RUP

4.3 Gestión del proyecto

A continuación, se explican las etapas y actividades para la puesta en marcha del proyecto, así como las actividades del proyecto y su posterior secuenciación y calendarización.

4.3.1 Plan inicial de desarrollo

Se estimará la duración y coste en recursos de todas las actividades del proyecto, poniendo especial atención en aquellas que se basan en el aprendizaje de nuevos métodos y herramientas, estas se darán sobre todo al principio del comienzo del proyecto y debido a su naturaleza solo puede estimarse su tiempo en base a suposiciones y experiencias previas en ese campo.

El desarrollo de este proyecto es individual y, por tanto, se comunicará a los tutores del proyecto la situación del trabajo realizado y las dudas que puedan surgir durante el desarrollo de este cada cierto tiempo. Por tanto, en un primer momento se hará un pequeño análisis del proyecto y se recopilará la información necesaria acerca de los conocimientos que serán necesarios adquirir para terminar el proyecto de forma correcta.

4.3.2 Gestión de riesgos

En este apartado se presentan todos aquellos riesgos que puede presentar la planificación del proyecto y sus actividades, detallando cada tipo de riesgo y su probabilidad de ocurrencia, además del impacto que supondría en la planificación. Cada riesgo va acompañado de su propio plan de contingencia y de reducción explicados.

En cada una de las fases e iteraciones del ciclo de vida del proyecto se revisarán los posibles riesgos y si se han dado o no y cómo mitigar o reducir cada uno de estos.

Impacto\ Probabilidad	Muy Alto	Alto	Medio	Bajo	Muy Bajo
Catastrófico	Alto	Alto	Moderado	Moderado	Bajo
Crítico	Alto	Alto	Moderado	Bajo	Ninguno
Marginal	Moderado	Moderado	Bajo	Ninguno	Ninguno
Despreciable	Moderado	Bajo	Bajo	Ninguno	Ninguno

Tabla 2: Relación impacto/probabilidad de los riesgos

4.3.2.1 Lista de riesgos detectados

RG-01 Re-configuración de los equipos informáticos de trabajo
Descripción
Los equipos informáticos prestados tienen una mala configuración hecha o no hay suficiente espacio en disco para instalar toda la suite de programas necesarios para desarrollar la aplicación.
Probabilidad: 5%
Impacto: Bajo
Plan de reducción
Ninguno
Plan de contingencia
Re configurar los equipos si es necesario tratando de perder el menor tiempo posible para ajustarse a la planificación del proyecto.

Tabla 3: Riesgo 01 - "Re-configuración de los equipos informáticos de trabajo"

RG-02 Préstamo de las gafas de realidad virtual
Descripción
Las gafas de realidad virtual HTC VIVE han sido prestadas por una empresa dedicada, entre otras cosas, a la organización de eventos y alquiler de equipos y gafas de realidad virtual y es muy probable que puedan necesitarlas de nuevo durante algún tiempo y se paralice temporalmente el desarrollo del proyecto.
Probabilidad: 20%
Impacto: Alto
Plan de reducción
Tratar de planificar las tareas que requieren pruebas y desarrollo haciendo uso de las gafas de realidad virtual en fechas en que no haya eventos de la empresa programados.
Plan de contingencia
Re planificar las tareas y revisar la secuenciación y fechas para las actividades sustituyendo las fases de implementación por las de diseño y análisis.

Tabla 4: Riesgo 02 - "Préstamo de las gafas de realidad virtual"

RG-03 Plugins y ajustes predefinidos de realidad virtual
Descripción
El tiempo de aprendizaje del funcionamiento de los plugins y el software previo es mayor del estimado al elaborar la planificación del proyecto.
Probabilidad: 25%
Impacto: Alto
Plan de reducción

Tratar de poner especial atención en la documentación de los mismos
Plan de contingencia
Planificar las tareas iniciales de aprendizaje de nuevo focalizando en aquellas tareas más importantes para cumplir con los objetivos del proyecto.

Tabla 5: Riesgo 03 - "Plugins y ajustes predefinidos de realidad virtual"

RG-04 Estructura del software previo
Descripción
El software previo sobre el que se basa el proyecto es demasiado complejo o no permite el uso de sesiones y multijugador de forma convencional y requiere modificaciones grandes en partes específicas del software previo.
Probabilidad: 15%
Impacto: Crítico
Plan de reducción
Ponerse en contacto con los responsables del desarrollo del software previo sobre el que se asienta buena parte de la aplicación.
Plan de contingencia
Ajustarse a la planificación y cumplir los plazos para obtener las funcionalidades mínimas.

Tabla 6: Riesgo 04 - "Estructura del software previo"

RG-05 Fallos en el diseño
Descripción
Los diseños y modelos de análisis elaborados son erróneos o no se corresponden con el sistema final.
Probabilidad: 10%
Impacto: Alto
Plan de reducción
Tener muy claros los requisitos del sistema y las partes sobrantes o innecesarias que sólo entorpecen las fases de diseño.
Plan de contingencia
Re diseñar el modelo y revisando la planificación y los diseños previos relacionados.

Tabla 7: Riesgo 05 - "Fallos en el diseño"

RG-06 Pérdida de los datos
Descripción
Algunas versiones recientes del proyecto se pierden o se corrompen en el motor Unreal Engine 4 dando lugar a una pérdida de tiempo
Probabilidad: 5%
Impacto: Catastrófico
Plan de reducción
Llevar un control de versiones y de cambios para evitar cualquier pérdida de información importante del proyecto.

Plan de contingencia
Se evalúan las pérdidas del proyecto y se re planifican las actividades necesarias para cumplir con los objetivos del proyecto.

Tabla 8: Riesgo 06 - "Pérdidas de los datos"

RG-07 Calendarización errónea
Descripción
La planificación de la estimación temporal de algunas tareas no se corresponde con el tiempo real de desarrollo.
Probabilidad: 15%
Impacto: Alto
Plan de reducción
Intentar estimar de forma correcta el tiempo que lleva la realización de cada una de las actividades del proyecto.
Plan de contingencia
Cumplir con los objetivos del proyecto y lo estipulado en la planificación del proyecto.

Tabla 9: Riesgo 07 - "Calendarización errónea"

4.4 Secuenciación y planificación de actividades

4.4.1 Etapa de Inicio

En esta fase se definen el alcance y objetivos del proyecto, identificando los posibles riesgos del proyecto en una primera instancia y además se añaden algunas actividades de estudio e investigación de herramientas y formas de trabajo con el motor Unreal Engine 4 [43].

***Las planificaciones de las actividades se han realizado considerando una jornada de trabajo de 2 horas.**

La mayor parte de esta etapa ha consistido en la búsqueda y recopilación de información valiosa acerca de algunos elementos importantes del proyecto y que se desconocían antes del comienzo de este trabajo. Consta de una sola iteración.

1. Conocer e investigar sobre el funcionamiento del modelo cliente-servidor dentro de la plataforma Unreal Engine 4 y su conexión con las tecnologías de realidad virtual.
2. Búsqueda y comprensión de información sobre el funcionamiento del *widget blueprint* dentro del motor gráfico y su conexión con otros componentes del sistema.
3. Investigar acerca de la creación de funcionalidad que permita ver partes del mapa del jugador a través de ventana dentro de un *widget blueprint*, similar a un minimapa con conexión multijugador.
4. Comprensión del funcionamiento de las librerías de realidad virtual en Unreal Engine 4.
5. Comprender en profundidad el funcionamiento del software previo desarrollado sobre el que se basa la aplicación, así como la función que desempeñan las librerías y plugins dentro del proyecto.
6. Buscar información sobre la optimización de software de realidad virtual y despliegue de un mismo software para diferentes plataformas que mantengan una conexión cliente-servidor.
7. Pruebas y experimentación en proyectos vacíos que no contienen funcionalidad previa para poder probar las funcionalidades y los componentes de manera aislada.
8. Con la información adquirida en las anteriores actividades se procede al estudio y estimación de los tiempos y actividades del proyecto, la calendarización y análisis de riesgos y se establece la visión general del sistema.

4.4.1.1 Actividades y calendarización de la fase de inicio

	Modo de	Nombre de tarea	Duración	Comienzo	Fin	Predecesoras
1		Comienzo	0 días	vie 10/02/17	vie 10/02/17	
2		**Inicio**	**27 días**	**vie 10/02/17**	**lun 20/03/17**	
3		**Iteración 1**	**27 días**	**vie 10/02/17**	**lun 20/03/17**	
4		**Proceso**	**27 días**	**vie 10/02/17**	**lun 20/03/17**	
5		Aprendizaje estructura cliente - servidor en UE4	7 días	vie 10/02/17	lun 20/02/17	1
6		Funcionamiento de los widget blueprint	4 días	mar 21/02/17	vie 24/02/17	5
7		Render target y minimapas en widgets	3 días	lun 27/02/17	mié 01/03/17	6
8		Librerias de realidad virtual	3 días	vie 17/02/17	mar 21/02/17	
9		Comprensión del software previo	8 días	jue 02/03/17	lun 13/03/17	7
10		Despliegue cross-platform	2 días	mar 14/03/17	mié 15/03/17	9
11		Experimentación	3 días	jue 16/03/17	lun 20/03/17	10
12		**Gestión**	**27 días**	**vie 10/02/17**	**lun 20/03/17**	
13		Análisis inicial de riesgos	3 días	vie 10/02/17	mar 14/02/17	1
14		Plan de fases	1 día	jue 16/02/17	jue 16/02/17	13
15		Plan de iteración inicial	2 días	dom 19/02/17	lun 20/02/17	14
16		Plan de iteración siguiente	5 días	sáb 25/02/17	jue 02/03/17	15
17		Seguimiento de la iteración actual	27 días	vie 10/02/17	lun 20/03/17	1

Figura 29: Actividades de la fase de inicio

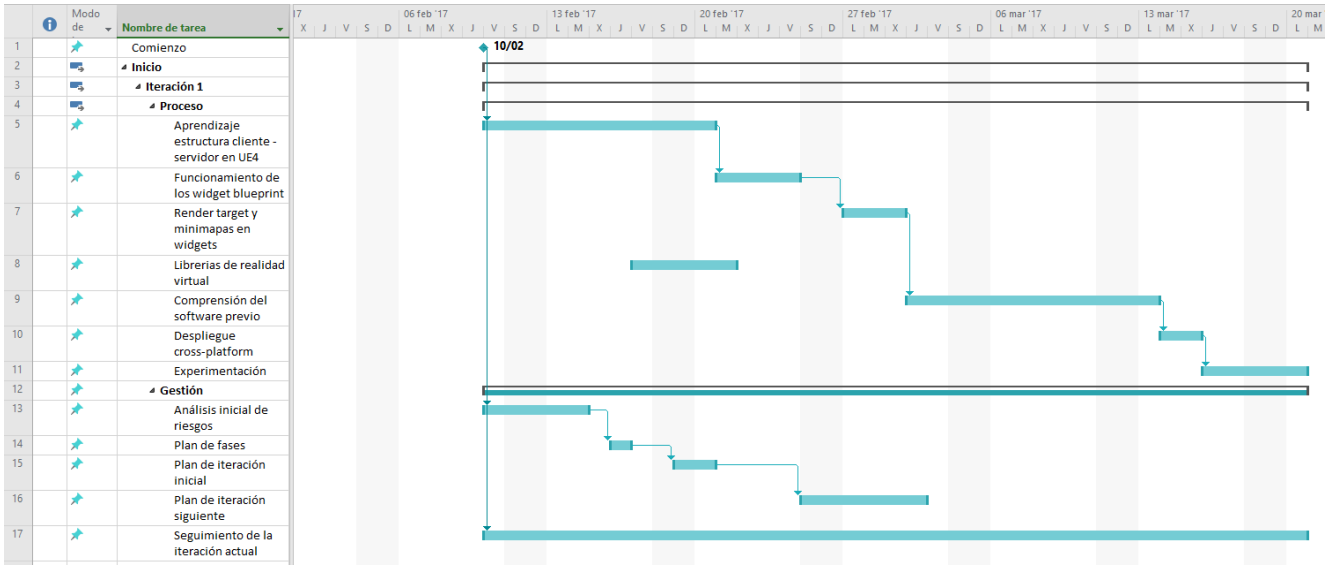


Figura 30: Calendarización gráfica de la fase de inicio

4.4.2 Etapa de Elaboración

Esta fase consiste en analizar y diseñar una primera solución para el sistema, para ello se desarrollan aquellos modelos y diagramas con el objetivo de despejar las incertidumbres sobre el proyecto [43].

Se analizan los requisitos del sistema y se crean todos los modelos de análisis del sistema necesarios como el modelo de dominio, los casos de uso y los diagramas de secuencia y de actividades siendo revisados y rediseñados sucesivamente en cada iteración, además también se crean los diseños de las interfaces del sistema y otros objetos 3D que se utilizarán durante la fase de construcción. Esta fase consta de **3 iteraciones**.

4.4.2.1 Actividades y calendarización de la fase de elaboración

- **Iteración 1**

	Modo de	Nombre de tarea	Duración	Comienzo	Fin	Predecesoras
1		Comienzo	0 días	vie 10/02/17	vie 10/02/17	
2		**Inicio**	**27 días**	**vie 10/02/17**	**lun 20/03/17**	
18		**Elaboración**	**28 días**	**mar 21/03/1**	**jue 27/04/17**	
19		**Iteración 1**	**10 días**	**mar 21/03/1**	**lun 03/04/17**	
20		**Proceso**	**10 días**	**mar 21/03/1**	**lun 03/04/17**	
21		Elicitación de requisitos del	3 días	mar 21/03/17	jue 23/03/17	11
22		Análisis de los casos de uso	2 días	mar 21/03/17	mié 22/03/17	11
23		Detalle de los casos de uso	2 días	jue 23/03/17	vie 24/03/17	22
24		Modelo de dominio del sistema	3 días	lun 27/03/17	mié 29/03/17	21;23
25		Vision general del sistema	1 día	jue 30/03/17	jue 30/03/17	24
26		Prototipo de la aplicación	2 días	vie 31/03/17	lun 03/04/17	25
27		**Gestión**	**10 días**	**mar 21/03/1**	**lun 03/04/17**	
28		Planes de reducción y contingencia para los riesgos	5 días	mar 21/03/17	lun 27/03/17	11
29		Plan de iteración siguiente	5 días	mar 28/03/17	lun 03/04/17	28
30		Seguimiento del plan de iteracción	10 días	mar 21/03/17	lun 03/04/17	11
31		**Iteración 2**	**9 días**	**mar 04/04/1**	**vie 14/04/17**	
44		**Iteración 3**	**9 días**	**lun 17/04/17**	**jue 27/04/17**	

Figura 31: Actividades de la iteración 1 de la fase de Elaboración

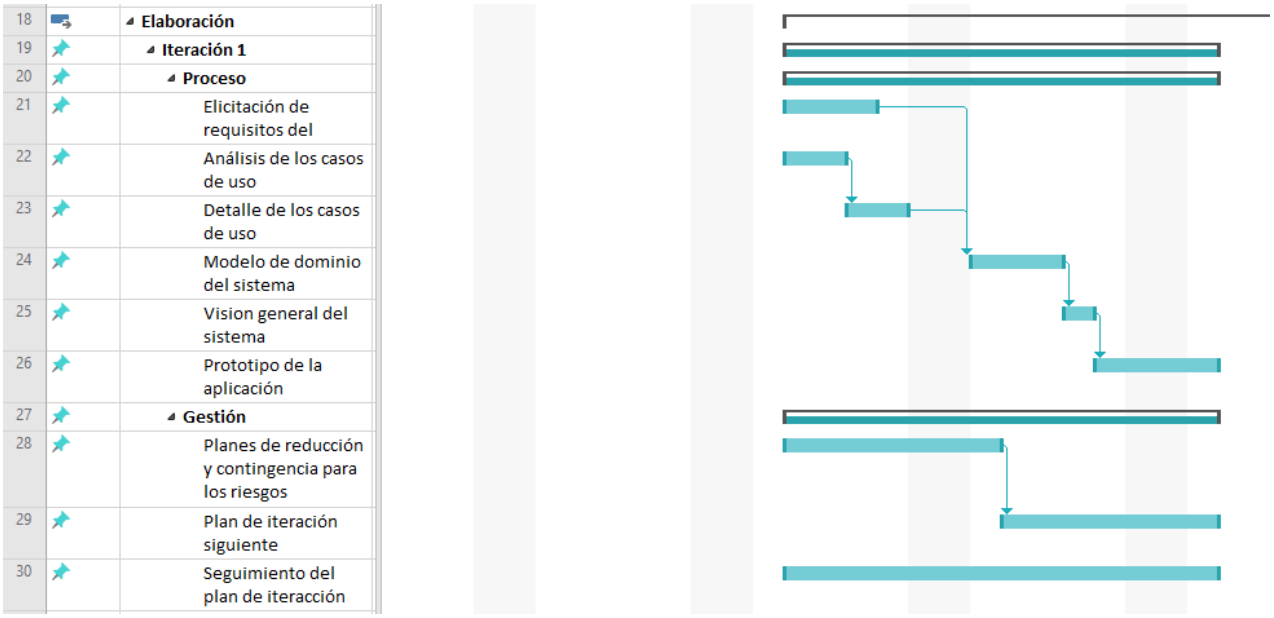


Figura 32: Calendarización gráfica de la iteración 1 de la fase de elaboración

- **Iteración 2**

	Modo de	Nombre de tarea	Duración	Comienzo	Fin	Predecesoras
1		Comienzo	0 días	vie 10/02/17	vie 10/02/17	
2		**Inicio**	**27 días**	**vie 10/02/17**	**lun 20/03/17**	
18		**Elaboración**	**28 días**	**mar 21/03/1**	**jue 27/04/17**	
19		**Iteración 1**	**10 días**	**mar 21/03/1**	**lun 03/04/17**	
31		**Iteración 2**	**9 días**	**mar 04/04/1**	**vie 14/04/17**	
32		**Proceso**	**9 días**	**mar 04/04/1**	**vie 14/04/17**	
33		Actualizar los casos de uso	2 días	mar 04/04/17	mié 05/04/17	26
34		Actualizar los requisitos del	1 día	jue 06/04/17	jue 06/04/17	33
35		Revisar el modelo de dominio	1 día	vie 07/04/17	vie 07/04/17	34
36		Diseño de navegabilidad entre interfaces del	2 días	mar 04/04/17	mié 05/04/17	26
37		Diseño de las interfaces del	4 días	jue 06/04/17	mar 11/04/17	36
38		Prototipo de interfaces de	4 días	jue 06/04/17	mar 11/04/17	36
39		Diseño de la arquitectura del sistema	3 días	mié 12/04/17	vie 14/04/17	38
40		**Gestión**	**9 días**	**mar 04/04/1**	**vie 14/04/17**	
41		Revisión de los riesgos	3 días	mar 04/04/17	jue 06/04/17	26
42		Plan de iteración siguiente	5 días	vie 07/04/17	jue 13/04/17	41
43		Seguimiento de la iteracción actual	9 días	mar 04/04/17	vie 14/04/17	26
44		**Iteración 3**	**9 días**	**lun 17/04/17**	**jue 27/04/17**	

Figura 33: Actividades de la iteración 2 de la fase de elaboración

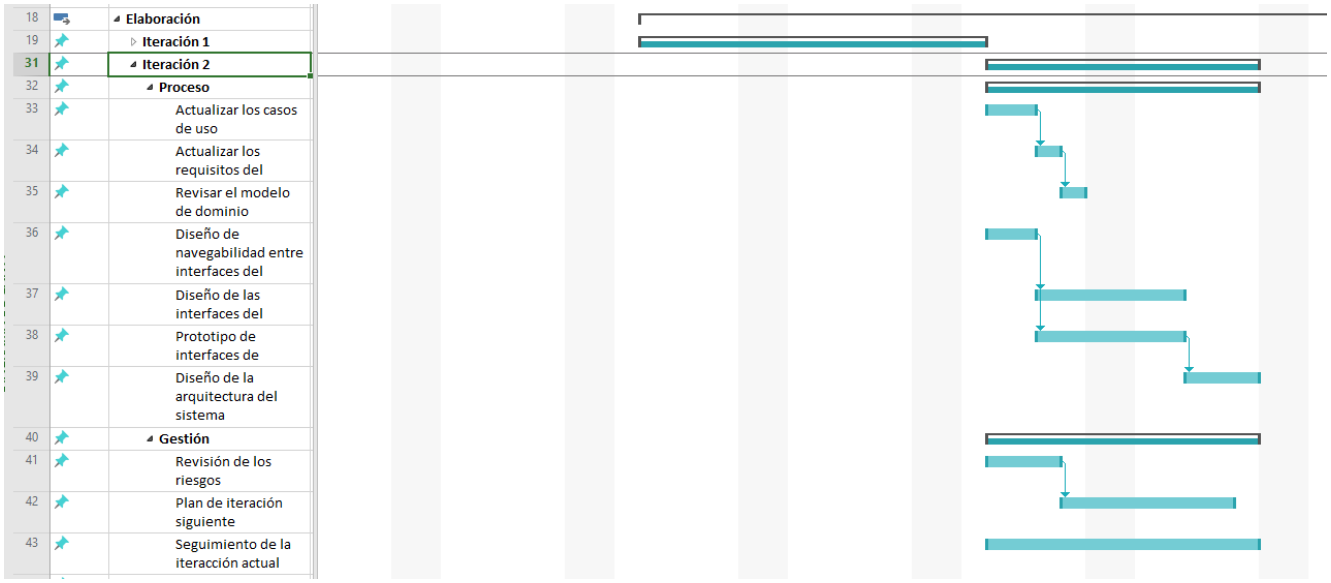


Figura 34: Calendarización gráfica de la iteración 2 de la fase de elaboración

- **Iteración 3**

	Modo de	Nombre de tarea	Duración	Comienzo	Fin	Predecesoras
1		Comienzo	0 días	vie 10/02/17	vie 10/02/17	
2		**Inicio**	**27 días**	**vie 10/02/17**	**lun 20/03/17**	
18		**Elaboración**	**28 días**	**mar 21/03/1**	**jue 27/04/17**	
19		**Iteración 1**	**10 días**	**mar 21/03/1**	**lun 03/04/17**	
31		**Iteración 2**	**9 días**	**mar 04/04/1**	**vie 14/04/17**	
44		**Iteración 3**	**9 días**	**lun 17/04/17**	**jue 27/04/17**	
45		**Proceso**	**9 días**	**lun 17/04/17**	**jue 27/04/17**	
46		Diseño de la arquitectura del	2 días	lun 17/04/17	mar 18/04/17	39
47		Organización por capas del sistema	2 días	mié 19/04/17	jue 20/04/17	46
48		Revisión de los casos de uso	1 día	lun 17/04/17	lun 17/04/17	39
49		Revisión del modelo de dominio	2 días	mar 18/04/17	mié 19/04/17	48
50		Diagramas de secuencia	5 días	vie 21/04/17	jue 27/04/17	47
51		Diseño de diagrama Entidad - Relación	3 días	jue 20/04/17	lun 24/04/17	49
52		**Gestión**	**9 días**	**lun 17/04/17**	**jue 27/04/17**	
53		Revisión de los riesgos	4 días	lun 17/04/17	jue 20/04/17	39
54		Plan de iteración siguiente	5 días	vie 21/04/17	jue 27/04/17	53
55		Seguimiento de la iteracción actual	9 días	lun 17/04/17	jue 27/04/17	39

Figura 35: Actividades de la iteración 3 de la fase de elaboración

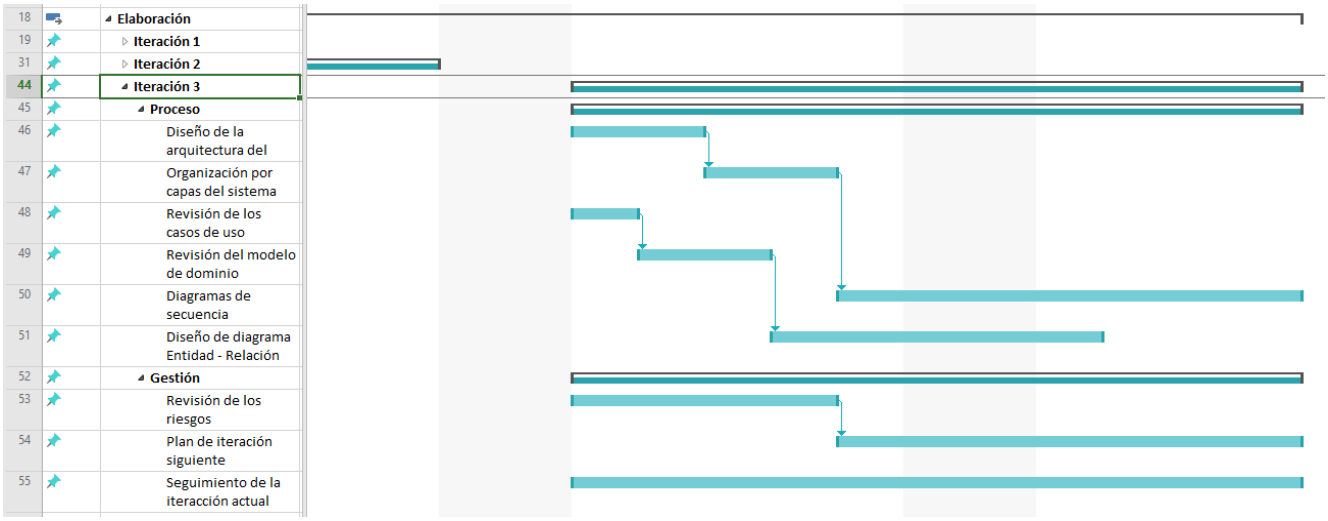


Figura 36: Calendarización gráfica de la iteración 3 de la fase de elaboración

4.4.3 Etapa de Construcción

Constituye la fase más larga del proceso y toma como entradas los artefactos del sistema obtenidos en la etapa anterior y se basa en estos para iterar mejorando las funcionalidades del sistema [43].

En esta fase se obtienen versiones mejoradas de la aplicación de forma incremental, empezando por la configuración inicial del proyecto, desarrollando las funcionalidades tanto de cliente como de servidor y finalmente implementando la base de datos y los casos de uso relacionados con esta. Consta de 4 iteraciones en las que, además, se revisan los modelos y se realiza el seguimiento de la planificación del proyecto.

4.4.3.1 Actividades y calendarización de la fase de construcción

- **Iteración 1**

	Modo de	Nombre de tarea	Duración	Comienzo	Fin	Predecesoras
1		Comienzo	0 días	vie 10/02/17	vie 10/02/17	
2		**Inicio**	**27 días**	**vie 10/02/17**	**lun 20/03/17**	
18		**Elaboración**	**28 días**	**mar 21/03/1**	**jue 27/04/17**	
56		**Construcción**	**38 días**	**vie 28/04/17**	**mar 20/06/1**	
57		**Iteración 1**	**11 días**	**vie 28/04/17**	**vie 12/05/17**	
58		**Proceso**	**11 días**	**vie 28/04/17**	**vie 12/05/17**	
59		Creación y configuración inicial	1 día	vie 28/04/17	vie 28/04/17	50
60		Adaptación del software previo	11 días	vie 28/04/17	vie 12/05/17	50
61		Implementación del modelo cliente -	7 días	lun 01/05/17	mar 09/05/17	59
62		Creación de las interfaces del	5 días	vie 28/04/17	jue 04/05/17	50
63		Diseño gráfico de las interfaces (canvas)	3 días	vie 05/05/17	mar 09/05/17	62
64		Minimapa del escenario en panel	3 días	mié 10/05/17	vie 12/05/17	63;61
65		Modelado 3D de elementos gráficos para las acciones del	6 días	vie 28/04/17	vie 05/05/17	50
66		**Gestión**	**11 días**	**vie 28/04/17**	**vie 12/05/17**	
67		Revisión de los riesg	4 días	vie 28/04/17	mié 03/05/1	50
68		Plan de iteración sigu	5 días	jue 04/05/17	mié 10/05/1	67
69		Seguimiento de la iteración actual	11 días	vie 28/04/17	vie 12/05/17	50
70		**Iteración 2**	**7 días**	**lun 15/05/17**	**mar 23/05/1**	
82		**Iteración 3**	**10 días**	**mié 24/05/1**	**mar 06/06/1**	
94		**Iteración 4**	**10 días**	**mié 07/06/1**	**mar 20/06/1**	

Figura 37: Actividades de la iteración 1 de la fase de construcción

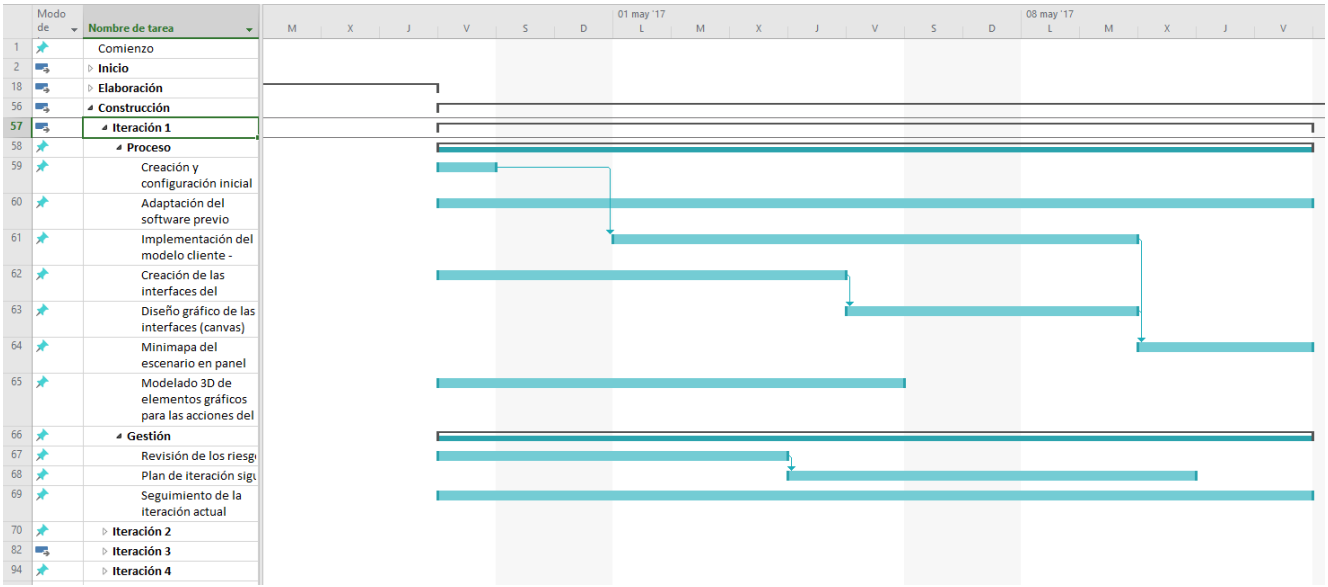


Figura 37: Calendarización gráfica de la iteración 1 de la fase de construcción

- **Iteración 2**

	Modo de	Nombre de tarea	Duración	Comienzo	Fin	Predecesoras
1		Comienzo	0 días	vie 10/02/17	vie 10/02/17	
2		**Inicio**	**27 días**	**vie 10/02/17**	**lun 20/03/17**	
18		**Elaboración**	**28 días**	**mar 21/03/1**	**jue 27/04/17**	
56		**Construcción**	**38 días**	**vie 28/04/17**	**mar 20/06/1**	
57		**Iteración 1**	**11 días**	**vie 28/04/17**	**vie 12/05/17**	
70		**Iteración 2**	**7 días**	**lun 15/05/17**	**mar 23/05/1**	
71		**Proceso**	**7 días**	**lun 15/05/17**	**mar 23/05/1**	
72		Implementación de 'saltar chispas'	5 días	lun 15/05/17	vie 19/05/17	65
73		Implementación de 'Acción fallo cadena'	4 días	lun 15/05/17	jue 18/05/17	65
74		Implementación de 'Acción corte por	5 días	lun 15/05/17	vie 19/05/17	65
75		Implementación de 'Acción evacuación'	4 días	lun 15/05/17	jue 18/05/17	65
76		CU Terminar intento	3 días	lun 15/05/17	mié 17/05/1	65
77		Implementación del contador de juego	2 días	lun 22/05/17	mar 23/05/17	72;73;74;75
78		**Gestión**	**7 días**	**lun 15/05/17**	**mar 23/05/1**	
79		Revisión de los riesg	3 días	lun 15/05/17	mié 17/05/1	65
80		Plan de iteración sigu	4 días	jue 18/05/17	mar 23/05/1	79
81		Seguimiento de la iteración actual	7 días	lun 15/05/17	mar 23/05/17	65
82		**Iteración 3**	**10 días**	**mié 24/05/1**	**mar 06/06/1**	
94		**Iteración 4**	**10 días**	**mié 07/06/1**	**mar 20/06/1**	

Figura 38: Actividades de la iteración 2 de la fase de construcción

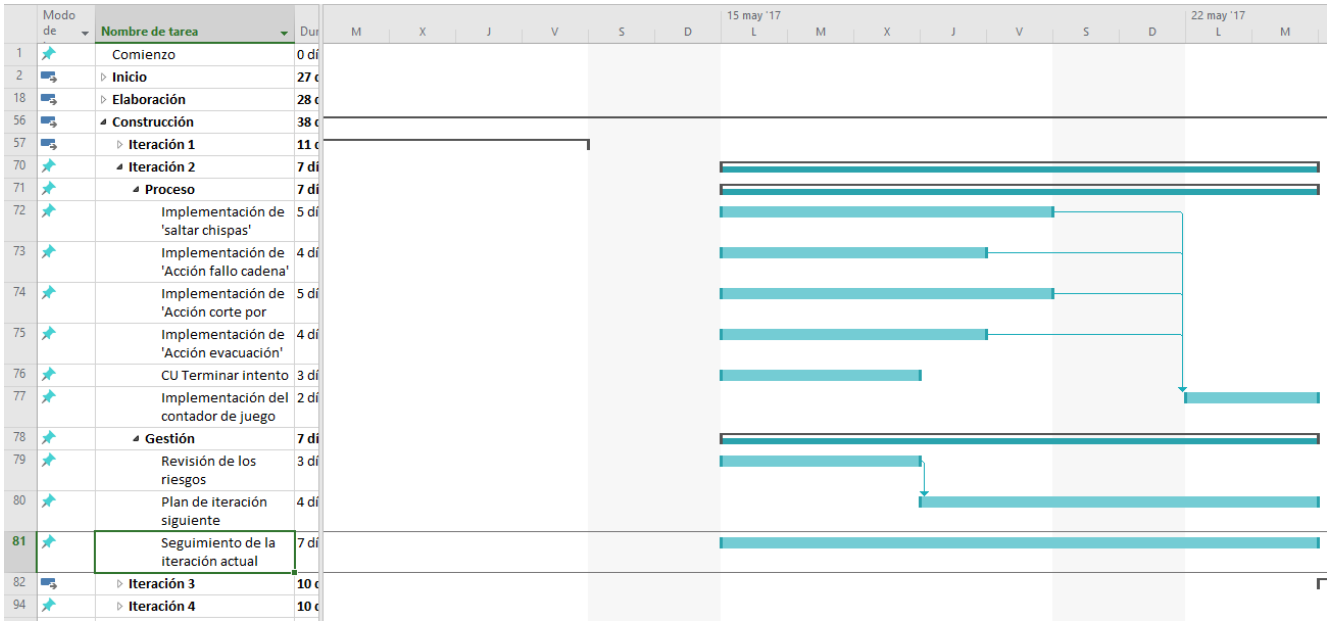


Figura 39: Calendarización gráfica de la iteración 2 de la fase de construcción

- **Iteración 3**

	Modo de	Nombre de tarea	Duración	Comienzo	Fin	Predecesoras
1		Comienzo	0 días	vie 10/02/17	vie 10/02/17	
2		**Inicio**	**27 días**	**vie 10/02/17**	**lun 20/03/17**	
18		**Elaboración**	**28 días**	**mar 21/03/1**	**jue 27/04/17**	
56		**Construcción**	**38 días**	**vie 28/04/17**	**mar 20/06/1**	
57		**Iteración 1**	**11 días**	**vie 28/04/17**	**vie 12/05/17**	
70		**Iteración 2**	**7 días**	**lun 15/05/17**	**mar 23/05/1**	
82		**Iteración 3**	**10 días**	**mié 24/05/1**	**mar 06/06/1**	
83		**Proceso**	**10 días**	**mié 24/05/1**	**mar 06/06/1**	
84		Refinamiento de la implementación de las funcionalidades del manager	4 días	mié 24/05/17	lun 29/05/17	77
85		Implementacion del CU 'Ponerse guantes'	5 días	mar 30/05/17	lun 05/06/17	84
86		Implementación del CU 'Ponerse casco'	5 días	mar 30/05/17	lun 05/06/17	84
87		Implementación del CU 'Ponerse gafas de protección	5 días	mar 30/05/17	lun 05/06/17	84
88		Implementación del CU "Colocar barra"	4 días	mar 30/05/17	vie 02/06/17	84
89		Revisión de las implementaciones de las interfaces	1 día	mar 06/06/17	mar 06/06/17	85;86;87;88
90		**Gestión**	**10 días**	**mié 24/05/1**	**mar 06/06/1**	
91		Revisión de los riesgos	4 días	mié 24/05/17	lun 29/05/17	77
92		Plan de iteración siguiente	6 días	mar 30/05/17	mar 06/06/17	91
93		Seguimiento de la iteración actual	10 días	mié 24/05/17	mar 06/06/17	77
94		**Iteración 4**	**10 días**	**mié 07/06/1**	**mar 20/06/1**	

Figura 40: Actividades de la iteración 3 de la fase de construcción

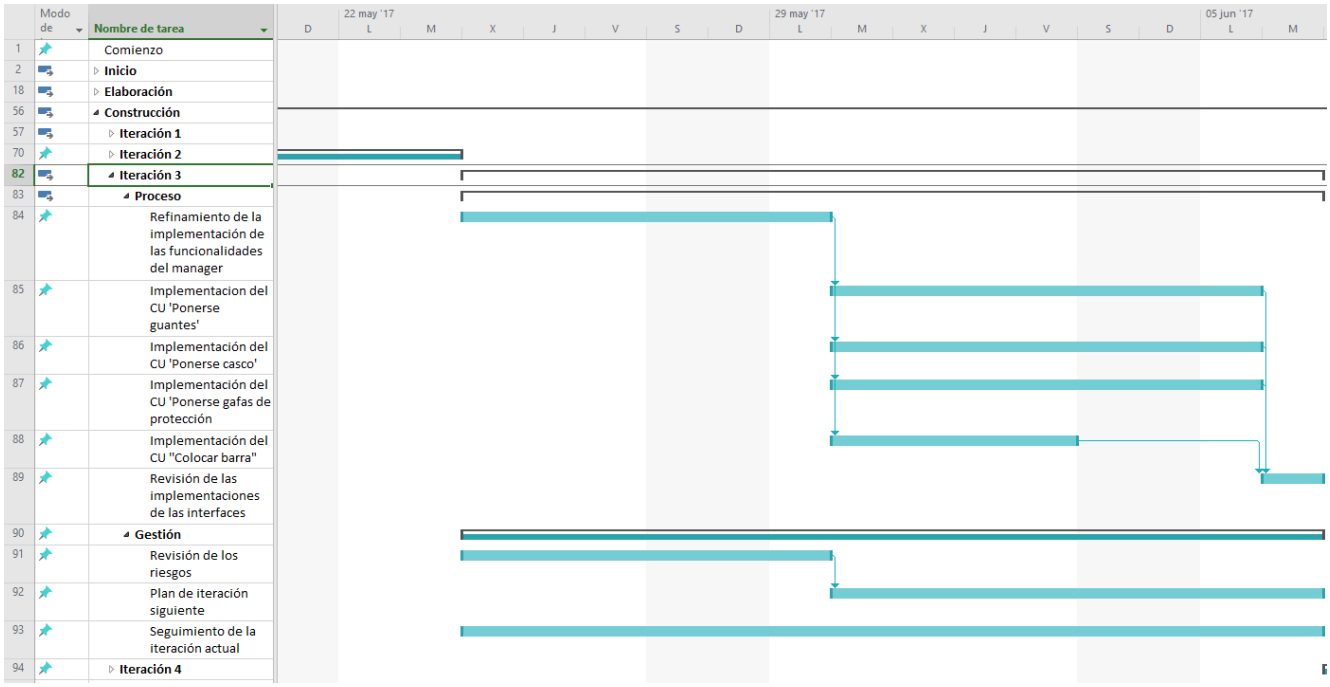


Figura 41: Calendarización gráfica de la iteración 3 de la fase de construcción

- **Iteración 4**

	Modo de	Nombre de tarea	Duración	Comienzo	Fin	Predecesoras
1		Comienzo	0 días	vie 10/02/17	vie 10/02/17	
2		**Inicio**	**27 días**	**vie 10/02/17**	**lun 20/03/17**	
18		**Elaboración**	**28 días**	**mar 21/03/1**	**jue 27/04/17**	
56		**Construcción**	**38 días**	**vie 28/04/17**	**mar 20/06/1**	
57		**Iteración 1**	**11 días**	**vie 28/04/17**	**vie 12/05/17**	
70		**Iteración 2**	**7 días**	**lun 15/05/17**	**mar 23/05/1**	
82		**Iteración 3**	**10 días**	**mié 24/05/1**	**mar 06/06/1**	
94		**Iteración 4**	**10 días**	**mié 07/06/1**	**mar 20/06/1**	
95		**Proceso**	**10 días**	**mié 07/06/1**	**mar 20/06/1**	
96		Refinamiento de la implementación del jugador	1 día	mié 07/06/17	mié 07/06/17	89
97		Implementación de la base de datos	3 días	mié 07/06/17	vie 09/06/17	89
98		Implementación del CU 'Registro de manager'	4 días	lun 12/06/17	jue 15/06/17	97
99		Implementación del CU 'Login del manager'	4 días	lun 12/06/17	jue 15/06/17	97
100		Implementación del CU 'Registro de jugador'	4 días	lun 12/06/17	jue 15/06/17	97
101		Implementación del CU 'Selección de jugador'	4 días	lun 12/06/17	jue 15/06/17	97
102		Cálculo de las puntuaciones	3 días	vie 16/06/17	mar 20/06/17	98;99;100;101
103		**Gestión**	**10 días**	**mié 07/06/1**	**mar 20/06/1**	
104		Revisión de los riesg	3 días	mié 07/06/1	vie 09/06/17	89
105		Plan de iteración siguiente	5 días	lun 12/06/17	vie 16/06/17	104
106		Seguimiento de la iteración actual	10 días	mié 07/06/17	mar 20/06/17	89

Figura 42: Actividades de la iteración 4 de la fase de construcción

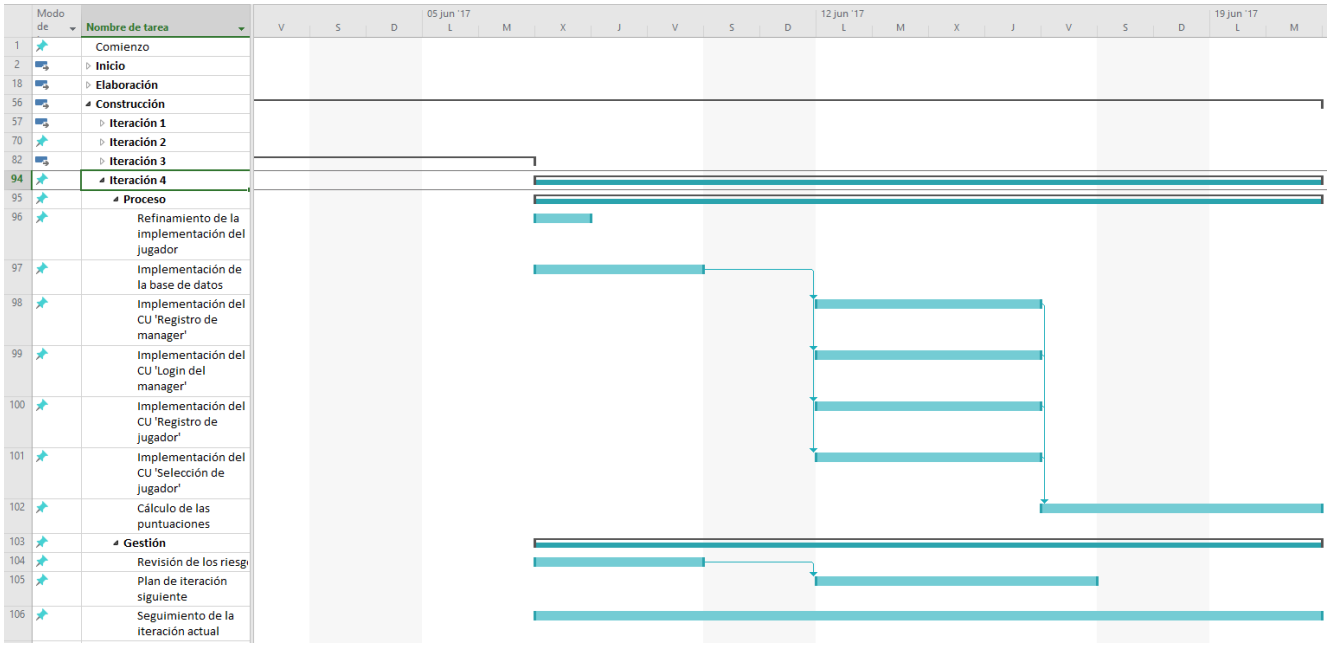


Figura 43: Calendarización gráfica de la iteración 4 de la fase de construcción

4.4.4 Etapa de Transición

Se trata de la última etapa del ciclo de vida del proceso, en esta se valida el sistema y se corrige aplicando los cambios y modificaciones necesarios [43].

Por tanto, en esta fase se despliega la aplicación y realiza una batería de pruebas y errores, se corrige el sistema modificando aquellos defectos identificados para posteriormente proceder a la elaboración del manual de usuario y la documentación necesaria para un correcto uso y comprensión de la aplicación. Consta de una iteración.

4.4.4.1 Actividades y calendarización de la fase de transición

	Modo de	Nombre de tarea	Duración	Comienzo	Fin	Predecesoras
1		Comienzo	0 días	vie 10/02/17	vie 10/02/17	
2		**Inicio**	**27 días**	**vie 10/02/17**	**lun 20/03/17**	
18		**Elaboración**	**28 días**	**mar 21/03/1**	**jue 27/04/17**	
56		**Construcción**	**38 días**	**vie 28/04/17**	**mar 20/06/1**	
107		**Transición**	**7 días**	**mié 21/06/1**	**jue 29/06/17**	
108		**Iteración 1**	**7 días**	**mié 21/06/1**	**jue 29/06/17**	
109		**Proceso**	**7 días**	**mié 21/06/1**	**jue 29/06/17**	
110		Optimización del código y revisión de	3 días	mié 21/06/17	vie 23/06/17	102
111		Despliegue de la aplicación	2 días	lun 26/06/17	mar 27/06/17	110
112		Pruebas finales				
113		Manual de usuario y de instalación	4 días	mié 21/06/17	lun 26/06/17	102
114		Redacción de la memoria	7 días	mié 21/06/17	jue 29/06/17	102
115		**Gestión**	**7 días**	**mié 21/06/1**	**jue 29/06/17**	
116		Revisión de riesgos	4 días	mié 21/06/1	lun 26/06/17	102
117		Seguimiento del plan de iteración	7 días	mié 21/06/17	jue 29/06/17	102

Figura 44: Actividades de la fase de transición

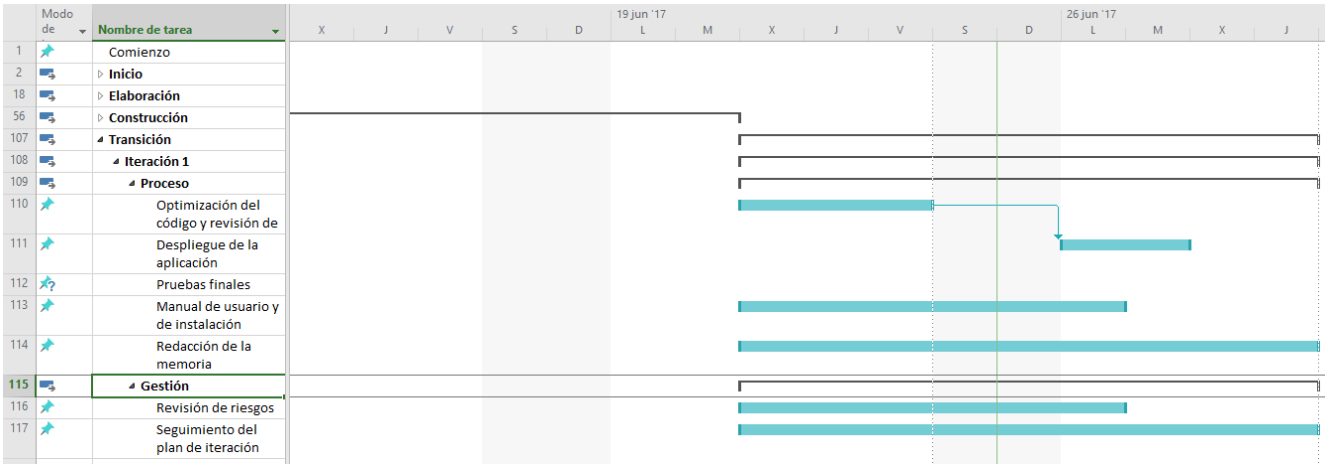


Figura 45: Calendarización gráfica de la fase de transición

4.5 Seguimiento de la planificación

En este apartado se comentan algunos de los riesgos y fallos de planificación que han ocurrido durante el proyecto:

4.5.1 Seguimiento de los riesgos

El riesgo '**RG-04 Estructura del software previo**' se dio durante la fase de inicio haciendo que se retrasara bastante la actividad posterior de la fase de construcción '**ID:60 – Adaptación del software previo**' durante la iteración 1:

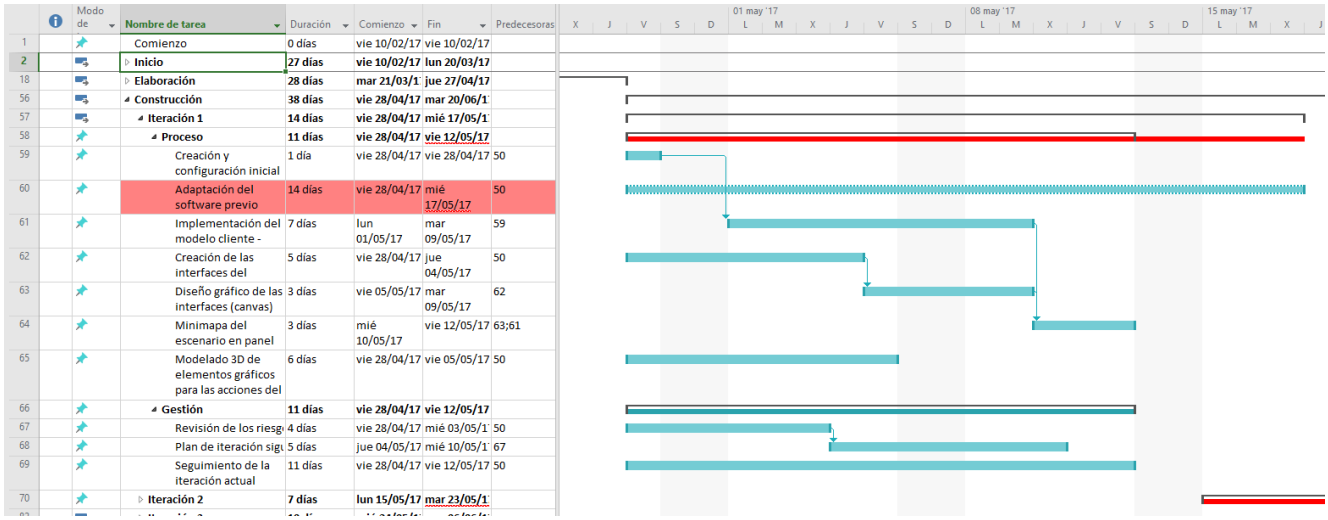
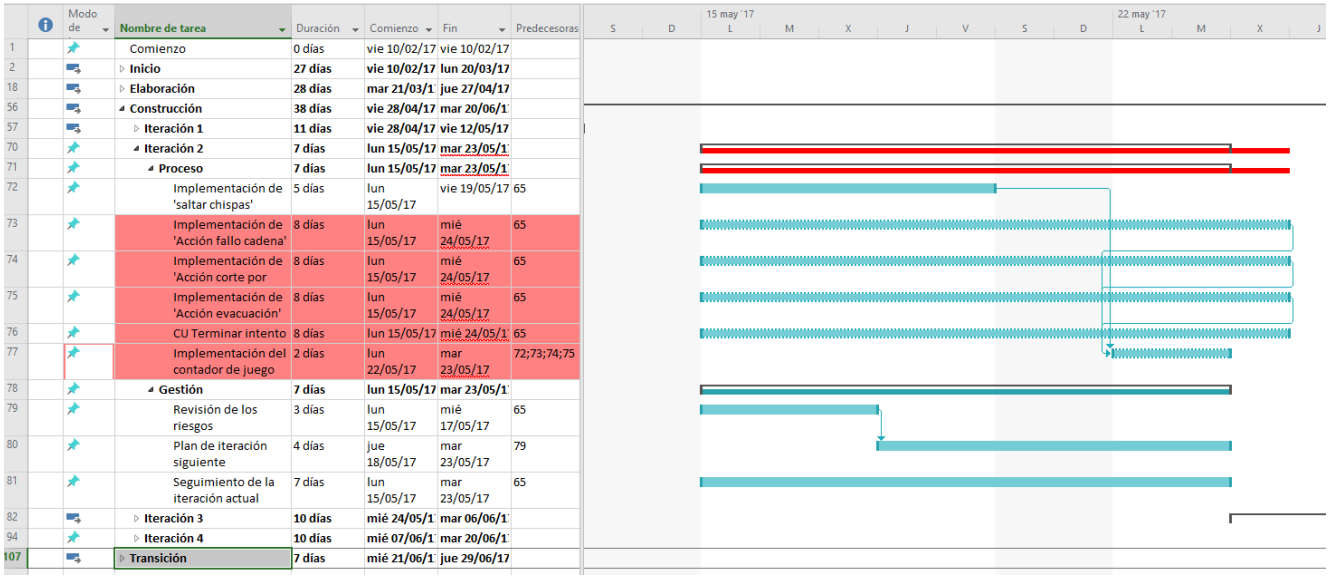


Figura 46: Retraso de la planificación durante la actividad 'Adaptación del software previo' de la fase de construcción

El riesgo '**RG-02 Préstamo de las gafas de realidad virtual**' se ha dado hasta en tres ocasiones durante la fase de construcción, haciendo que algunas actividades de esta fase queden retrasadas en el tiempo. En estas ocasiones se ha optado por trabajar sobre el papel y desarrollar partes de la funcionalidad que no requieren necesariamente de las gafas de realidad virtual.

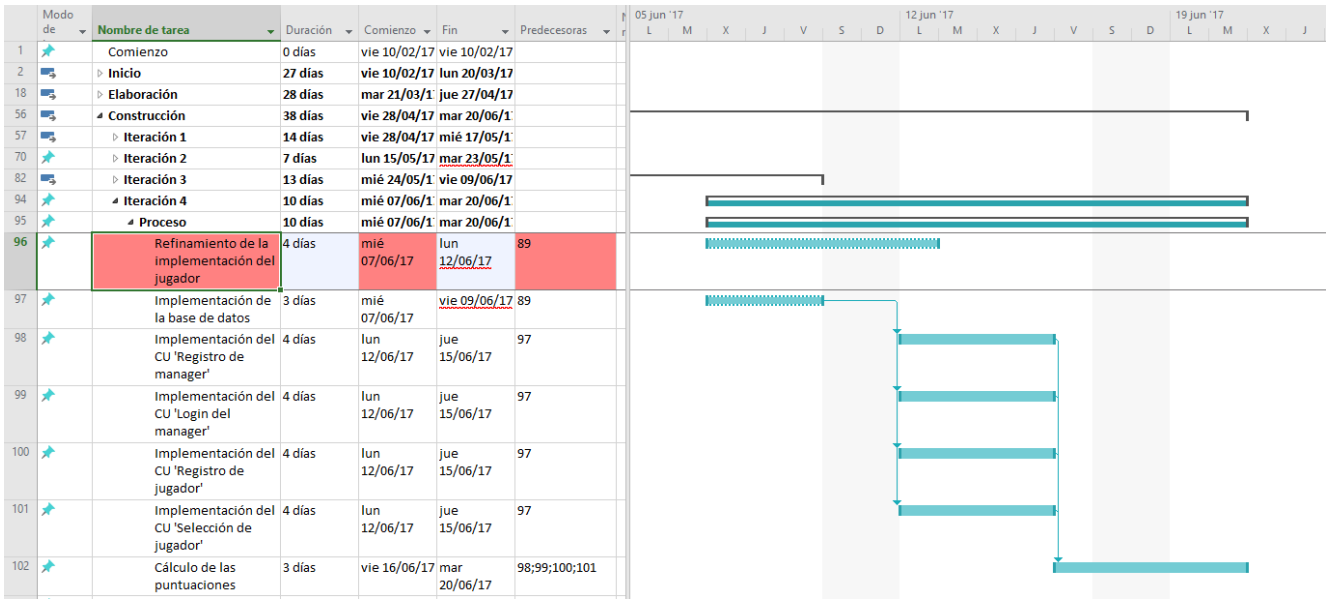
Durante la **iteración 2 de la fase de construcción**, las gafas de realidad virtual tuvieron que ser devueltas durante 4 días haciendo que la planificación de las tareas programadas para esas fechas tuviera que retrasarse.



	Modo de	Nombre de tarea	Duración	Comienzo	Fin	Predecesoras
1		Comienzo	0 días	vie 10/02/17	vie 10/02/17	
2		**Inicio**	**27 días**	**vie 10/02/17**	**lun 20/03/17**	
18		**Elaboración**	**28 días**	**mar 21/03/17**	**jue 27/04/17**	
56		**Construcción**	**38 días**	**vie 28/04/17**	**mar 20/06/17**	
57		**Iteración 1**	**11 días**	**vie 28/04/17**	**vie 12/05/17**	
70		**Iteración 2**	**7 días**	**lun 15/05/17**	**mar 23/05/17**	
71		**Proceso**	**7 días**	**lun 15/05/17**	**mar 23/05/17**	
72		Implementación de 'saltar chispas'	5 días	lun 15/05/17	vie 19/05/17	65
73		Implementación de 'Acción fallo cadena'	8 días	lun 15/05/17	mié 24/05/17	65
74		Implementación de 'Acción corte por	8 días	lun 15/05/17	mié 24/05/17	65
75		Implementación de 'Acción evacuación'	8 días	lun 15/05/17	mié 24/05/17	65
76		CU Terminar intento	8 días	lun 15/05/17	mié 24/05/17	65
77		Implementación del contador de juego	2 días	lun 22/05/17	mar 23/05/17	72;73;74;75
78		**Gestión**	**7 días**	**lun 15/05/17**	**mar 23/05/17**	
79		Revisión de los riesgos	3 días	lun 15/05/17	mié 17/05/17	65
80		Plan de iteración siguiente	4 días	jue 18/05/17	mar 23/05/17	79
81		Seguimiento de la iteración actual	7 días	lun 15/05/17	mar 23/05/17	65
82		**Iteración 3**	**10 días**	**mié 24/05/17**	**mar 06/06/17**	
94		**Iteración 4**	**10 días**	**mié 07/06/17**	**mar 20/06/17**	
107		**Transición**	**7 días**	**mié 21/06/17**	**jue 29/06/17**	

Figura 47: Retraso en las actividades de implementación de las acciones del manager durante la etapa de construcción

También durante las **iteraciones 3 y 4**:



	Modo de	Nombre de tarea	Duración	Comienzo	Fin	Predecesoras
1		Comienzo	0 días	vie 10/02/17	vie 10/02/17	
2		**Inicio**	**27 días**	**vie 10/02/17**	**lun 20/03/17**	
18		**Elaboración**	**28 días**	**mar 21/03/17**	**jue 27/04/17**	
56		**Construcción**	**38 días**	**vie 28/04/17**	**mar 20/06/17**	
57		**Iteración 1**	**14 días**	**vie 28/04/17**	**mié 17/05/17**	
70		**Iteración 2**	**7 días**	**lun 15/05/17**	**mar 23/05/17**	
82		**Iteración 3**	**13 días**	**mié 24/05/17**	**vie 09/06/17**	
94		**Iteración 4**	**10 días**	**mié 07/06/17**	**mar 20/06/17**	
95		**Proceso**	**10 días**	**mié 07/06/17**	**mar 20/06/17**	
96		Refinamiento de la implementación del jugador	4 días	mié 07/06/17	lun 12/06/17	89
97		Implementación de la base de datos	3 días	mié 07/06/17	vie 09/06/17	89
98		Implementación del CU 'Registro de manager'	4 días	lun 12/06/17	jue 15/06/17	97
99		Implementación del CU 'Login del manager'	4 días	lun 12/06/17	jue 15/06/17	97
100		Implementación del CU 'Registro de jugador'	4 días	lun 12/06/17	jue 15/06/17	97
101		Implementación del CU 'Selección de jugador'	4 días	lun 12/06/17	jue 15/06/17	97
102		Cálculo de las puntuaciones	3 días	vie 16/06/17	mar 20/06/17	98;99;100;101

Figura 48: Retraso en la actividad de refinamiento de la implementación del jugador durante la fase de construcción

5. ANÁLISIS DE REQUISITOS

5.1 Objetivos del sistema e introducción

En este capítulo se analizan los requisitos de la aplicación desarrollada durante este trabajo, primero veremos los objetivos del sistema, después los requisitos funcionales de la aplicación, más tarde se hace un estudio de los requisitos no funcionales, después de los requisitos de información del sistema y, por último, se analizan algunas de las reglas de negocio que tendrá la aplicación.

Los objetivos del sistema son requisitos funcionales del usuario final de la aplicación, cada uno de los objetivos del sistema se especifica a continuación [42]:

5.1.1 OBJ-01: Control y seguimiento de un usuario a través de un panel de control en una red local en tiempo real

La aplicación deberá permitir gestionar y controlar la partida de un jugador a través de un panel de control manejado por un usuario previamente registrado como manager. Existirán dos aplicaciones, una preparada para el cliente y otra para el servidor, pudiéndose conectar entre ambas a través de red local (LAN).

5.1.2 OBJ-02: Obtención de resultados y guardado de datos

La aplicación deberá guardar información de las partidas que se lleven a cabo a través de esta en una base de datos. Además, deberá permitir el registro de jugadores y de usuarios manager que controlarán el panel de control y tendrán acceso a estas funciones.

5.2 Actores del sistema

En esta sección se especifican cuáles son los actores que existen en el sistema, habiendo identificado 3, el usuario administrador, el usuario manager y el usuario jugador.

5.2.1 Administrador

Se trata de un usuario que puede manejar tanto el panel de control como dar de alta a otros managers y tener acceso a la base de datos.

5.2.2 Manager

Representa a un usuario manager que puede manejar el panel de control pero que no tiene permisos de administrador.

5.2.3 Jugador

Representa al jugador de la aplicación del lado cliente y el que utilizará las gafas de realidad virtual junto con la aplicación.

5.3 Requisitos funcionales

A continuación, se exponen los requisitos funcionales, no funcionales y de información extraídos del análisis de la aplicación, así como los actores, objetivos y reglas de negocio del sistema.

5.3.1 RF-01: Registro de usuarios manager

El sistema deberá permitir a un usuario administrador registrar usuarios de tipo manager que se guardarán en una base de datos a través del panel de control que gestiona las partidas de usuario.

5.3.2 RF-02: Login de usuarios manager

El sistema deberá permitir iniciar sesión en la aplicación servidor a un usuario manager para poder gestionar partidas y usuarios a través del panel de control.

5.3.3 RF-03: Registro de jugadores

El sistema deberá permitir a un usuario manager registrar nuevos jugadores proporcionando nombre, apellidos, dni, edad y si tiene o no experiencia.

5.3.4 RF-04: Iniciar partidas

El sistema deberá permitir a un usuario manager iniciar partidas proporcionando un jugador existente que se encuentre registrado en la base de datos.

5.3.5 RF-05: Visualizar escenario

El sistema deberá permitir la visualización en tiempo real del jugador sobre el escenario de juego a través del panel de control del usuario manager.

5.3.6 RF-06: Control de cámaras

El sistema deberá permitir a un usuario manager escoger y cambiar entre diferentes puntos de vista sobre el escenario.

5.3.7 RF-07: Visualización de datos en tiempo real

El sistema deberá mostrar a través del panel de control o dashboard toda la información relevante acerca de la partida que está llevando a cabo el jugador registrado dependiendo del tipo de escenario concreto (tiempo transcurrido, número de objetivos cumplidos…).

5.3.8 RF-08: Control de elementos del entorno del jugador

El sistema deberá permitir controlar a un usuario manager a través del panel de control algunos elementos del entorno que afectarán en tiempo real a la jugabilidad del jugador (Control de brazos robot, control de elementos externos como evacuación…).

5.3.9 RF-9: Conexión con base de datos

El sistema deberá conectarse a una base de datos local para guardar y extraer información relevante sobre las partidas que se lleven a cabo y deberá mostrarse a través de la interfaz del manager.

5.3.10 RF-10: Guardado de información de partidas

El sistema deberá guardar en la base de datos todos aquellos datos de importancia sobre las partidas llevadas a cabo por jugadores.

5.3.11 RF-11: Conexión vía red

El sistema deberá permitir la gestión y control de partidas de los jugadores desde otro dispositivo distinto al del jugador, bien por red local o a través de red remota.

5.3.12 RF-12: Consultar información del jugador

El sistema deberá permitir a un usuario manager la visualización de los datos relevantes del jugador que se encuentra en una partida multijugador a través del panel de control principal de la aplicación.

5.3.13 RF-13: Accionar proyección de material

El sistema deberá permitir a un usuario manager proyectar material desde el brazo soldador de la máquina del escenario de fabricación cuando se encuentre en una partida activa

5.3.14 RF-14: Provocar atasque en el robot soldador

El sistema deberá permitir atascar el robot soldador de la máquina del escenario de fabricación cuando se encuentra en una partida activa.

5.3.15 RF-15: Provocar fallo en la producción en cadena

El sistema deberá permitir a un usuario manager simular un fallo en la cadena de producción a través del panel de control cuando este se encuentre en una sesión de juego activa.

5.3.16 RF-16: Provocar corte por barra metálica

El sistema deberá permitir a un usuario manager simular un corte en la mano del controlador del jugador a través del panel de control durante una sesión de juego activa.

5.3.17 RF-17: Puntuación partida

El sistema deberá calcular las puntuaciones finales al acabar una partida en base a las reglas de negocio de la aplicación, teniendo en cuenta el tiempo transcurrido y los hitos completados a lo largo de esta.

5.3.18 RF-18: Cancelación de partida

El sistema deberá permitir a un usuario manager cancelar una partida, aunque esta no haya terminado, dándole a elegir si se guardan o no los datos de la partida hasta el momento de la cancelación.

5.3.19 RF-19 Protecciones de manos

El sistema deberá permitir a un usuario jugador ponerse las protecciones de las manos que en este caso serán guantes.

5.3.20 RF-20 Protección de la cabeza

El sistema deberá permitir a un usuario jugador ponerse el casco protector.

5.3.21 RF-21 Protección de los ojos

El sistema deberá permitir a un usuario jugador ponerse las gafas de protección para los ojos.

5.3.22 RF-22 Colocar piezas de carrocería

El sistema deberá permitir a un usuario jugador colocar barras metálicas en el escenario de fabricación durante una partida en el orden correcto.

5.3.23 RF-23 Cambio de fase

El sistema deberá permitir a un usuario jugador cambiar de fase en el proceso de fabricación dentro del escenario de fabricación.

5.3.24 RF-24 Solventar atasque del robot soldador

El sistema deberá permitir a un usuario jugador solventar un atasque en el robot soldador producido por una acción del usuario manager.

5.3.25 RF-25 Solventar fallo de producción en cadena

El sistema deberá permitir a un usuario jugador solventar un fallo en la cadena de producción producido por una acción del usuario manager.

5.3 Requisitos no funcionales

En esta sección se exponen los requisitos no funcionales del sistema identificados a lo largo del proceso de análisis.

5.3.1 RNF-01: Datos en tiempo real

El sistema deberá visualizar los datos y las actividades relativas a un jugador en tiempo real, es decir, con un tiempo de respuesta muy bajo.

5.3.2 RNF-02: Entorno controlado

La aplicación deberá ejecutarse en un espacio controlado y sin alteraciones del entorno para que el tracking de las gafas de realidad virtual no sufra interferencias en la partida.

5.3.3 RNF-03: Aplicación cómoda

El jugador deberá sentirse cómodo a la hora de realizar las actividades de la aplicación, evitando en la medida de lo posible mareos y problemas asociados que conllevan muchas de las aplicaciones de realidad virtual.

5.3.4 RNF-04: Compatibilidad

La aplicación deberá poder ejecutarse en todos aquellos equipos que cumplan las especificaciones mínimas que se requieren para una aplicación de realidad virtual.

5.3.5 RNF-05: Usabilidad

El sistema deberá ser sencillo e intuitivo de utilizar a fin de garantizar que no se requiera invertir una cantidad de tiempo significativa en su aprendizaje.

5.3.6 RNF-06: Adaptabilidad

La aplicación deberá estar adaptada a todo tipo de pantallas de las que pueda disponer un dispositivo apto para la realidad virtual.

5.3.7 RNF-07: Hardware destino

El sistema deberá funcionar tanto con las gafas HTC Rift como con las gafas HTC Vive de una forma similar.

5.3.8 RNF-08: Lenguaje de la base de datos

Se utilizará el lenguaje SQL como lenguaje de desarrollo para la base de datos.

5.4 Requisitos de información

En esta sección se detallan algunos de los requisitos de información resultado del análisis de requisitos del sistema.

5.4.1 RI-01: Datos sobre un jugador

- Nombre (String)
- Apellidos (String)
- DNI (String)
- Edad (Int)

- Experiencia (boleano)

5.4.2 RI-02: Datos sobre un usuario manager

- Nombre (String)
- Apellidos (String)
- DNI (String)
- Administrador (boolean)
- Contraseña (String)
- Correo (String)

5.4.3 RI-03: Datos de la partida

- Tiempo transcurrido (int – en segundos)
- Tipo de escenario (String)
- Puntuación (int)
- Num fallos del jugador (int)
- Eventos del manager (Array de eventos)
- Manager (String)
- Jugador (String)

5.4.7 RI-07: Acciones del manager

- Tipo de acción (Enum)
- Momento (Timestamp)

5.5 Reglas de negocio

5.5.1 RN-01: Cálculo de puntuaciones

El cálculo de puntuaciones que obtiene un jugador cuando finaliza uno de los escenarios de juego dentro de la aplicación se calculará siguiendo la fórmula:

$$puntuación = \frac{(1000) - \left[(tiempo\ en\ segundos) + \left((fallosLeves) * 50\right) + (fallosGraves * 150) + (p * 200)\right]}{10}$$

*Siendo '***p'** una variable que representa a las protecciones del usuario y que puede tomar el valor 0 si el usuario utilizó las protecciones de seguridad y 1 si no las utilizó.

5.5.2 RN-02 Protecciones y fallos graves

Se consideran como **fallos graves** aquellos que tienen que ver con el uso de protecciones y los eventos de manager, las situaciones en que esto ocurre son:

- Se ejecuta la acción de proyección de material sobre el escenario de fabricación y el usuario jugador no lleva puestas las gafas de protección.
- Se ejecuta la acción de corte por barra metálica y el jugador no lleva los guantes de protección.

- Al finalizar la partida el usuario jugador no lleva puesta alguna de las protecciones de seguridad obligatorias (este tipo de fallo además se penaliza como p*200 tal y como viene en la fórmula de cálculo de las puntuaciones).

5.5.3 RN-03 Fallos leves y secuencia de producción

Se considera como un **fallo leve** aquellas ocasiones en las que el jugador se equivoque en la secuencia normal del flujo de trabajo, como, por ejemplo, intentar colocar una barra en la máquina soldadora en un orden incorrecto en el escenario de fabricación.

6. DISEÑO

6.1 Introducción

En este capítulo se expone el diseño de software elaborado para el desarrollo de la aplicación llevada a cabo en este proyecto. Dado que la aplicación construida consiste en el desarrollo de una aplicación basada en técnicas de realidad virtual y desarrollada a través del motor Unreal Engine partiendo de una base de software previamente realizada también en este motor, se han omitido algunas partes del software pertenecientes a este desarrollo previo, centrando la atención únicamente en aquellos elementos desarrollados durante este trabajo, de modo que algunos de los diagramas de secuencia elaborados se presentan en un nivel alto de abstracción tratando como elementos de caja negra a algunos de los componentes previamente desarrollados.

6.2 Casos de uso del sistema

En este apartado se describen los casos de uso del sistema identificados y se describe en detalle cada caso de uso concreto, para ello se divide esta sección en dos partes, conteniendo la primera todo lo referente al análisis de los casos de uso de la aplicación y su diagrama correspondiente, y la segunda parte contiene la descripción y el detalle de los pasos de los casos de uso.

6.2.1 Diagrama de los casos de uso

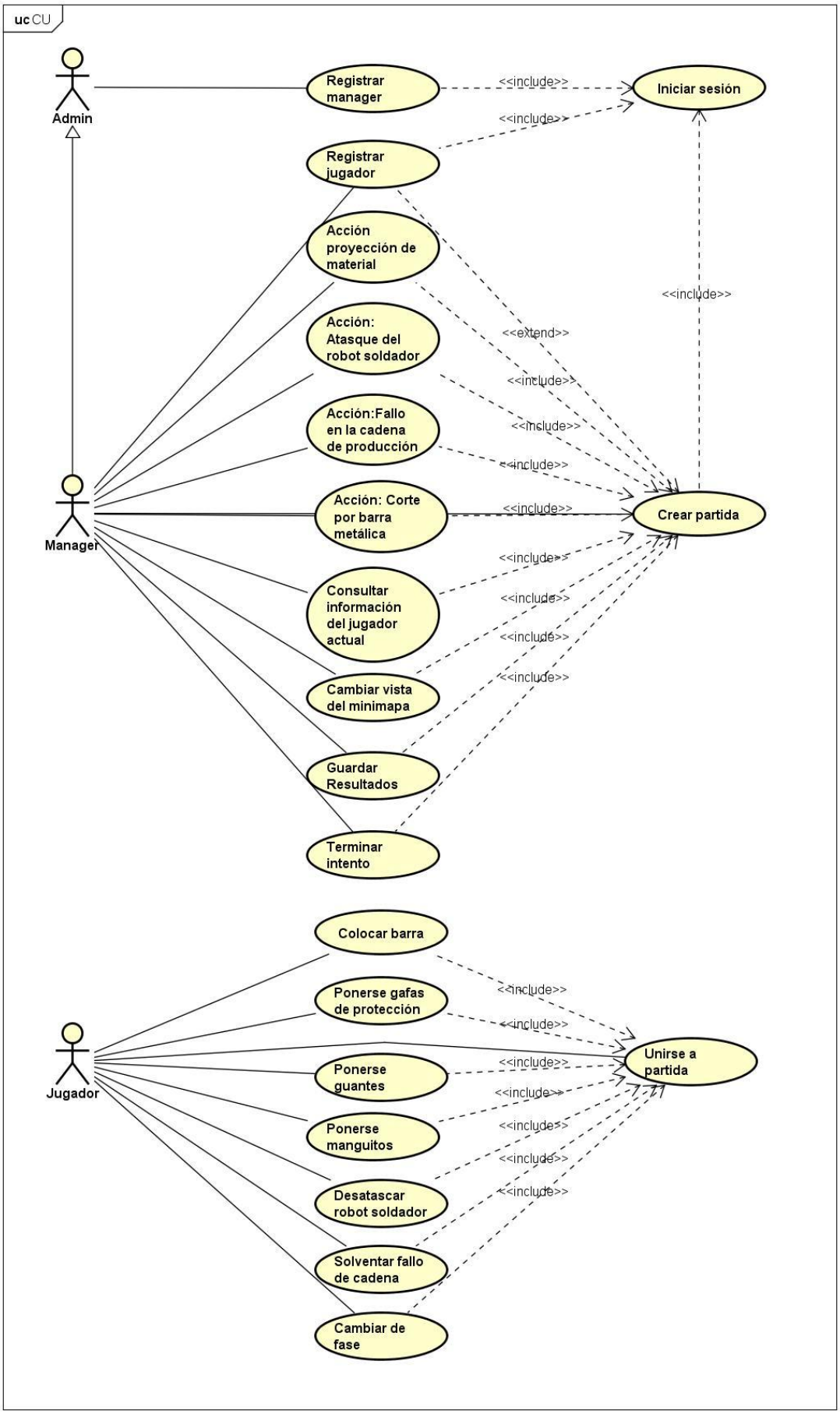


Figura 49: Diagrama de los casos de uso del sistema

6.2.2 Detalle de los casos de uso

CU-01: **Registrar manager**
Precondición El usuario se ha identificado previamente en el sistema y posee privilegios de administrador
Pos condición Se ha creado un nuevo usuario ‘manager’ en la base de datos
Descripción • 1. El usuario administrador desea registrar un usuario manager. • 2. El sistema solicita nombre, apellidos, documento de identidad, correo electrónico, si tiene privilegios de administrador, contraseña y confirmación de esta. • 3. El usuario manager rellena los datos. • 4. El sistema comprueba que son válidos. • 5. El sistema registra el nuevo usuario manager. • 6. El sistema muestra realimentación de éxito al usuario.
Excepciones • 3.1 Si el usuario desea cancelar la operación, el sistema mostrará realimentación al usuario, a continuación, este caso de uso termina. • 4.1 Si los datos introducidos no son válidos el sistema informará al usuario, a continuación, este caso de uso continúa en el paso 2. • 4.2 Si ya existe un usuario manager con un documento de identidad similar entonces el sistema informará al usuario, este caso de uso continúa en el paso 2.

Tabla 10: Caso de uso 01 - "Registrar manager"

CU-02: **Iniciar sesión**
Precondición Ninguna
Pos condición El usuario se ha identificado en el sistema
Descripción • 1. El usuario desea identificarse como manager. • 2. El sistema solicita documento de identidad y contraseña. • 3. El usuario manager rellena los datos. • 4. El sistema comprueba que son válidos. • 5. El sistema muestra realimentación de éxito al usuario.
Excepciones • 3.1 Si el usuario desea cancelar la operación, el sistema mostrará realimentación al usuario, a continuación, este caso de uso termina. • 4.1 Si los datos introducidos no son válidos el sistema informará al usuario, a continuación, este caso de uso continúa en el paso 2.

• 4.2 Si los datos introducidos no coinciden en dni/contraseña se informará al usuario, este caso de uso continuará en el paso 2.

Tabla 11: Caso de uso 02 - "Iniciar sesión"

CU-03: **Crear partida**
Precondición El usuario se ha identificado previamente en el sistema
Pos condición El sistema pasa al estado de crear una partida
Descripción • 1. El usuario manager solicita crear una partida para un jugador. • 2. El sistema solicita el documento de identidad del jugador. • 3. El usuario manager introduce los datos requeridos. • 4. El sistema comprueba que son válidos. • 5. El sistema muestra realimentación de éxito al usuario. • 6. El sistema solicita selección del escenario de juego. • 7. El usuario manager selecciona el escenario de juego. • 8. El sistema crea una sesión en red en el escenario escogido y muestra realimentación al usuario.
Excepciones • 1.1 Si el usuario desea registrar un jugador, se realiza el caso de uso CU-04: Registrar jugador. • 3.1 Si el usuario desea cancelar la operación, el sistema mostrará realimentación al usuario, a continuación, este caso de uso termina. • 4.1 Si los datos introducidos no son válidos el sistema informará al usuario, a continuación, este caso de uso continúa en el paso 2. • 4.2 Si no existe ningún jugador registrado con estos datos, el sistema informa del error y este caso de uso continúa en el paso 2. • 6.1 Si el usuario desea cancelar la sesión de juego, el usuario muestra realimentación, a continuación, este caso de uso termina.

Tabla 12: Caso de uso 03 - "Crear partida"

CU-04: **Registrar jugador**
Precondición El usuario se ha identificado previamente en el sistema y ha seleccionado crear una partida
Pos condición Se ha creado un nuevo usuario 'jugador' en la base de datos
Descripción • 1. El usuario administrador desea registrar un usuario jugador. • 2. El sistema solicita nombre, apellidos, documento de identidad, correo electrónico, teléfono de contacto y si tiene experiencia previa. • 3. El usuario manager rellena los datos. • 4. El sistema comprueba que son válidos. • 5. El sistema se comunica con el sistema gestor de la base de datos y solicita el registro del nuevo usuario jugador con los datos proporcionados. • 6. El sistema muestra realimentación de éxito al usuario.
Excepciones • 3.1 Si el usuario desea cancelar la operación, el sistema mostrará realimentación al usuario, a continuación, este caso de uso termina. • 4.1 Si los datos introducidos no son válidos el sistema informará al usuario, a continuación, este caso de uso continúa en el paso 2. • 4.2 Si ya existe un usuario jugador con un documento de identidad similar entonces el sistema informará al usuario, este caso de uso continúa en el paso 2.

Tabla 13: Caso de uso 04 – "Registrar jugador"

CU-05: **Acción: Proyección de material**
Precondición Existe una partida en curso
Pos condición Se ha actualizado el estado del juego, se ha actualizado el estado del jugador y se ha detenido el flujo del proceso de fabricación del jugador.
Descripción • 1. El usuario manager solicita ejecutar la acción de proyectar material desde el robot soldador. • 2. El sistema realiza la acción sobre la partida actual y se comunican los cambios al cliente que se encuentra en la partida.
Excepciones • 1.1 Si el usuario manager solicita la orden más de dos veces en un corto espacio de tiempo, el sistema únicamente acepta la primera, a continuación, este caso de uso continúa. • 2.1 Si el usuario jugador no lleva las gafas de protección puestas cuando se ejecuta la acción, entonces provocará que este vea disminuida su visión en el juego.

Tabla 14: Caso de uso 05 - "Proyección de material"

CU-06: **Acción: Atasque en el robot soldador**
Precondición Existe una partida en curso
Pos condición Se ha actualizado el estado del juego y se ha detenido el flujo del proceso de fabricación del jugador.
Descripción • 1. El usuario manager solicita ejecutar la acción de provocar un atasque en el robot soldador principal. • 2. El sistema realiza la acción sobre la partida actual y se comunican los cambios al cliente que se encuentra en la partida.
Excepciones • 1.1 Si el usuario manager solicita la orden más de dos veces en un corto espacio de tiempo, el sistema únicamente acepta la primera, a continuación, este caso de uso continúa.

Tabla 15: Caso de uso 06 - "Atasque en el robot soldador"

CU-07: **Acción: Fallo en la cadena de producción**
Precondición Existe una partida en curso
Pos condición Se ha actualizado el estado del juego y se ha detenido el flujo del proceso de fabricación del jugador.
Descripción • 1. El usuario manager solicita ejecutar la acción de provocar un fallo en la cadena de montaje. • 2. El sistema realiza la acción sobre la partida actual y se comunican los cambios al cliente que se encuentra en la partida.
Excepciones • 1.1 Si el usuario manager solicita la orden más de dos veces en un corto espacio de tiempo, el sistema únicamente acepta la primera, a continuación, este caso de uso continúa.

Tabla 16: Caso de uso 07 - "Fallo en la cadena de producción"

CU-08: **Acción: Corte por barra metálica**
Precondición Existe una partida en curso y el jugador está sujetando una de las barras metálicas del juego
Pos condición Se ha actualizado el estado del juego y el jugador pasa al estado ‘cortado’.
Descripción • 1. El usuario manager solicita ejecutar la acción de provocar un fallo en la cadena de montaje. • 2. El sistema realiza la acción sobre la partida actual y se comunican los cambios al cliente que se encuentra en la partida.
Excepciones • 1.1 Si el usuario manager solicita la orden más de dos veces en un corto espacio de tiempo, el sistema únicamente acepta la primera, a continuación, este caso de uso continúa.

Tabla 17: Caso de uso 08 - "Acción: Corte por barra metálica"

CU-09: **Cambiar la vista del minimapa**
Precondición Existe una partida en curso
Pos condición Se ha actualizado la vista actual del minimapa del panel de control del manager
Descripción • 1. El usuario manager solicita cambiar la vista del escenario • 2. El sistema muestra 3 posibilidades de ángulos de visión desde los que observar el transcurso de la partida. • 3. El usuario manager selecciona uno de ellos. • 4. El sistema actualiza la vista actual del panel de control.
Excepciones • Ninguna

Tabla 18: Caso de uso 09 - "Cambiar la vista del minimapa"

CU-10: **Consultar información del jugador**
Precondición Existe una partida en curso
Pos condición Se muestra la información del jugador actual al usuario
Descripción • 1. El usuario manager solicita visualizar los datos del jugador de la sesión en cuso. • 2. El sistema muestra los datos del jugador al usuario.
Excepciones • Ninguna

Tabla 19: Caso de uso 10 - "Consultar información del jugador"

CU-11: **Guardar resultados**
Precondición Existe una partida en curso y se ha completado el juego
Pos condición Se ha creado una nueva partida en la base de datos
Descripción • 1. El usuario manager solicita guardar los datos de una partida terminada. • 2. El sistema solicita confirmación al usuario. • 3. El usuario manager confirma la acción. • 4. El sistema recoge los datos de la partida, calcula las puntuaciones y las guarda en la base de datos.
Excepciones • 3.1 Si el usuario no confirma la acción, este caso de uso queda sin efecto.

Tabla 20: Caso de uso 11 - "Guardar resultados"

CU-12: **Terminar intento**
Precondición Existe una partida en curso
Pos condición Se ha cancelado la sesión de juego
Descripción • 1. El usuario manager solicita finalizar la partida actual del jugador. • 2. El sistema solicita confirmación al usuario. • 3. El usuario manager confirma la acción. • 4. El sistema cierra la sesión de juego y vuelve a la interfaz de creación de partidas.
Excepciones • 3.1 Si el usuario manager no confirma la terminación de la partida, entonces este caso de uso queda sin efecto.

Tabla 21: Caso de uso 12 - "Terminar intento"

CU-13: **Unirse a partida**
Precondición Un usuario manager ha creado una sesión de juego en la misma red (wifi o cable)
Pos condición El usuario jugador se ha unido a la sesión del servidor
Descripción • 1. El usuario jugador solicita realizar una búsqueda de sesiones en la red • 2. El sistema busca partidas activas, si encuentra una se une. • 3. El sistema notifica al usuario que se ha unido a la partida.
Excepciones • 3.1 Si el sistema no encuentra partidas activas, este caso de uso continua en el paso 2.

Tabla 22: Caso de uso 13 - "Unirse a partida"

CU-14: **Colocar barra metálica**
Precondición Existe una partida en curso y el jugador sostiene una de las barras metálicas de la aplicación.
Pos condición Se ha actualizado el estado del juego.
Descripción • 1. El usuario jugador selecciona la opción de colocar una de las piezas metálicas en la máquina soldadora. • 2. El sistema coloca la barra en la máquina dentro del escenrio de fabricación e informa al servidor de la sesión de juego activa de que actualice el estado del juego. • 3. El sistema actualiza el estado del juego y muestra realimentación gráfica al usuario y al servidor.
Excepciones • 1.1 Si la barra seleccionada a colocar no corresponde al orden correcto, el sistema envía una notificación al servidor, haciendo que aumenten los fallos del jugador. • 2.1 Si la conexión entre cliente y servidor falla entonces el sistema muestra un mensaje de error y se intenta reconectar con la partida. Este caso de uso queda sin efecto.

Tabla 23: Caso de uso 14 - "Colocar barra metálica"

CU-15: **Ponerse gafas de protección**
Precondición Existe una partida en curso
Pos condición Se ha actualizado el estado del juego.
Descripción • 1. El usuario jugador solicita ponerse las gafas de protección. • 2. El sistema coloca las gafas al jugador e informa al servidor de la sesión de juego activa de que actualice el estado del juego. • 3. El sistema actualiza el estado del juego y muestra realimentación gráfica al usuario y al servidor.
Excepciones • 2.1 Si la conexión entre cliente y servidor falla entonces el sistema muestra un mensaje de error y se intenta reconectar con la partida. Este caso de uso queda sin efecto.

Tabla 24: Caso de uso 15 - "Ponerse gafas de protección"

CU-16: **Ponerse guantes de protección**
Precondición Existe una partida en curso
Pos condición Se ha actualizado el estado del juego.
Descripción • 1. El usuario solicita ponerse los guantes de protección. • 2. El sistema coloca los guantes al jugador e informa al servidor de la sesión de juego activa de que actualice el estado del juego. • 3. El sistema actualiza el estado del juego y muestra realimentación gráfica al usuario y al servidor.
Excepciones • 2.1 Si la conexión entre cliente y servidor falla entonces el sistema muestra un mensaje de error y se intenta reconectar con la partida. Este caso de uso queda sin efecto.

Tabla 25: Caso de uso 16 - "Ponerse guantes de protección"

CU-17: **Ponerse casco de protección**
Precondición Existe una partida en curso
Pos condición Se ha actualizado el estado del juego.
Descripción • 1. El usuario jugador solicita ponerse el casco de protección. • 2. El sistema coloca el casco al jugador e informa al servidor de la sesión de juego activa de que actualice el estado del juego. • 3. El sistema actualiza el estado del juego y muestra realimentación gráfica al usuario y al servidor.
Excepciones • 2.1 Si la conexión entre cliente y servidor falla entonces el sistema muestra un mensaje de error y se intenta reconectar con la partida. Este caso de uso queda sin efecto.

Tabla 26: Caso de uso 17 - "Ponerse casco de protección"

CU-18: **Desatascar robot soldador**
Precondición Existe una partida en curso y el usuario manager ha activado el atasque del robot soldador desde el panel de control.
Pos condición Se ha actualizado el estado del juego.
Descripción • 1. El usuario jugador solicita desatascar el robot soldador del escenario de fabricación. • 2. El sistema desatasca el robot permitiendo seguir con la jugabilidad e informa al servidor de la sesión de juego activa de que actualice el estado del juego. • 3. El sistema actualiza el estado del juego y muestra realimentación gráfica al usuario y al servidor.
Excepciones • 2.1 Si la conexión entre cliente y servidor falla entonces el sistema muestra un mensaje de error y se intenta reconectar con la partida. Este caso de uso queda sin efecto.

Tabla 27: Caso de uso 18 - "Desatascar robot soldador"

CU-19: **Solventar fallo de cadena**
Precondición Existe una partida en curso y el usuario manager ha activado previamente la acción de fallo de cadena de producción.
Pos condición Se ha actualizado el estado del juego.
Descripción • 1. El usuario jugador solicita resolver el fallo en la cadena de producción. • 2. El sistema reactiva la producción en cadena permitiendo seguir con la jugabilidad e informa al servidor de la sesión de juego activa de que actualice el estado del juego. • 3. El sistema actualiza el estado del juego y muestra realimentación gráfica al usuario y al servidor.
Excepciones • 2.1 Si la conexión entre cliente y servidor falla entonces el sistema muestra un mensaje de error y se intenta reconectar con la partida. Este caso de uso queda sin efecto.

Tabla 28:Caso de uso 19 - "Solventar fallo de cadena"

CU-20: **Cambiar de fase**
Precondición Existe una partida en curso y el usuario manager ha activado previamente la acción de fallo de cadena de producción.
Pos condición Se ha actualizado el estado del juego.
Descripción • 1. El usuario jugador acerca uno de los controladores del juego hacia el botón que alerta del fallo de cadena. • 2. El sistema reactiva la producción en cadena permitiendo seguir con la jugabilidad e informa al servidor de la sesión de juego activa de que actualice el estado del juego. • 3. El sistema actualiza el estado del juego y muestra realimentación gráfica al usuario y al servidor.
Excepciones • 1.1 Si la acción se lleva a cabo en un momento incorrecto, el sistema informa al servidor de que el usuario jugador ha cometido un fallo leve. • 2.1 Si la conexión entre cliente y servidor falla entonces el sistema muestra un mensaje de error y se intenta reconectar con la partida. Este caso de uso queda sin efecto.

Tabla 29: Caso de uso 20 - "Cambiar de fase"

6.1 Modelo de dominio

En este apartado se analizan las clases que contendrá la aplicación y su interconexión entre ellas a través de un modelo de dominio del sistema, dado que estamos trabajando con una herramienta de software como es el motor Unreal Engine 4, algunas de las clases y utilidades que se utilizarán en la aplicación real no estarán reflejadas de forma directa en el modelo de dominio principal que se detalla más abajo, sino que se ha intentado realizar de la forma más abstracta posible el diseño de las entidades principales que compondrán la aplicación.

A continuación, se expone una abstracción del sistema a través del diagrama de dominio de la aplicación:

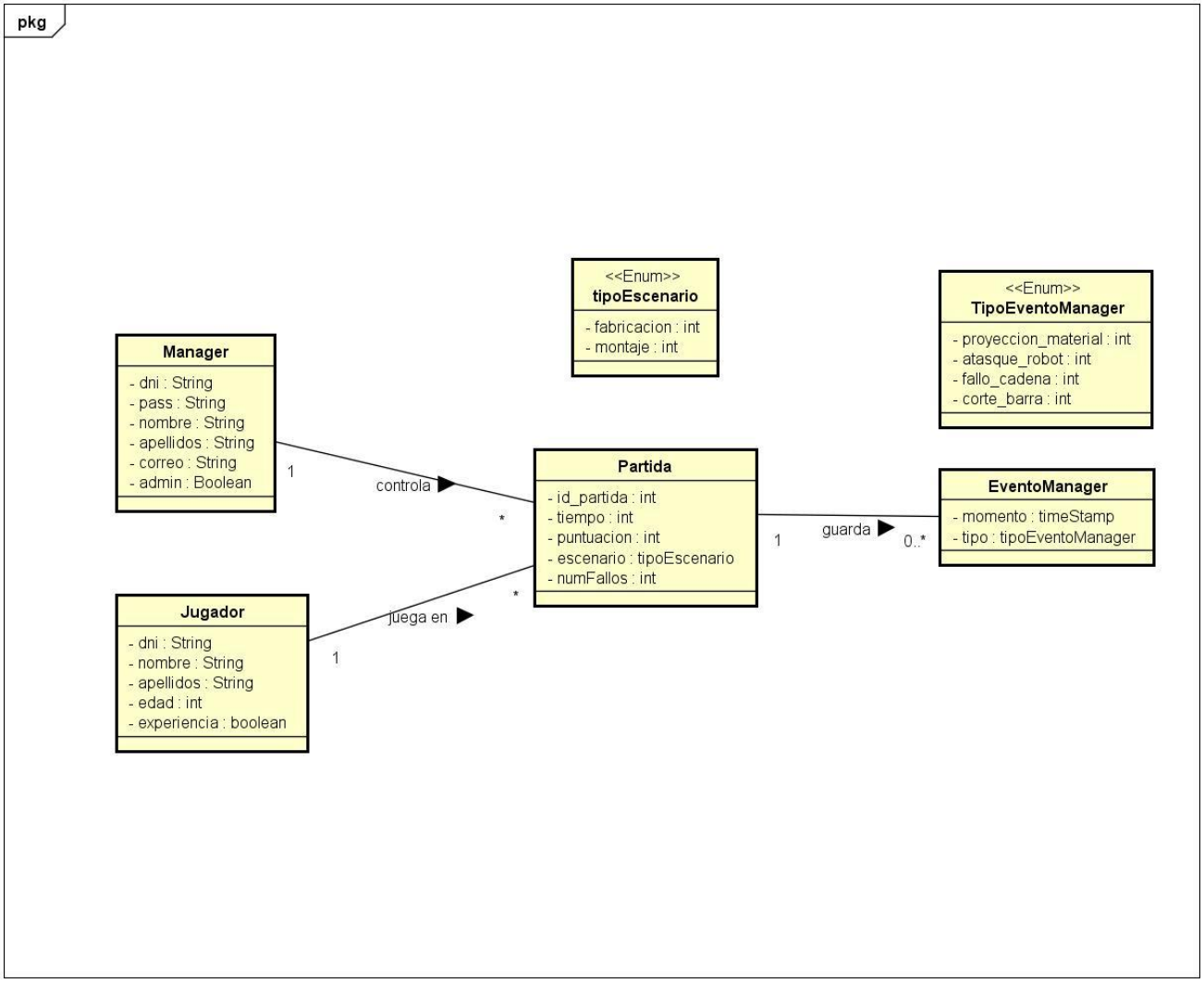


Figura 50: Modelo de dominio del sistema

6.2 Visión general del sistema

En esta sección se describe la arquitectura del sistema y tanto la vista global del mismo organizado por paquetes de funcionalidad, habiendo adaptado el modelo MVC a la estructura y formas de trabajo en el motor Unreal Engine 4, como el detalle de cada uno de los paquetes.

6.3.1 Arquitectura del sistema y adaptación del patrón MVC

Se realizarán dos aplicaciones de escritorio que comparten arquitectura y funcionalidades, una que será utilizada en un dispositivo por el usuario manager y la otra que será utilizada por el jugador, ambas se comunicarán vía red. La diferencia fundamental reside en que una aplicación se desplegará con algunos ajustes del motor que se explican en la sección de Implementación de la aplicación y la otra cambiando esos ajustes, por tanto, la aplicación es en esencia la misma.

Se ha aplicado el **patrón MCV** [44] en la forma de construcción del software en la medida de lo posible dado que el desarrollo se realiza sobre una herramienta que tiene sus propios flujos de trabajo y que no necesariamente sigue este patrón, por tanto, ha sido necesario utilizar un **único controlador para las interfaces** del panel de control del usuario manager que será del tipo '**PlayerState**'.

Esto se hace porque es la clase más adecuada según la forma de trabajo que se sigue en el motor Unreal Engine 4. También existen otros dos tipos de controladores fundamentales, una de ellas es una clase de tipo '**GameInstance'** y que almacenará toda la información relativa al estado del juego, como las interfaces creadas, cuál de estas es la que está activa o el estado del jugador. Esta instancia es la única que **persiste entre un cambio de niveles** y que únicamente existe en el servidor, haciéndola indispensable para almacenar toda la información relevante de la partida, dado que habrá cambios de niveles. Otra de las clases que actuarán como controladoras del sistema, será una clase de tipo '**Pawn**' y que será idéntica tanto en la aplicación del servidor como en la del cliente. Esta clase contendrá toda la funcionalidad que puede realizar el usuario y la única capaz de **enviar mensajes RPC al servidor**.

A mayores también podríamos considerar como una clase controlador los **blueprint del nivel** de juego, como cada nivel tiene el suyo, en este caso tendríamos 3 niveles, el nivel de inicio del servidor, el de inicio del jugador y el escenario de fabricación.

Además, en el caso del usuario se hace prácticamente imposible seguir una estructura puramente MVC porque la interacción del usuario no pasa por interfaces gráficas, sino que **sus acciones son enviadas y procesadas a través de la red hacia el servidor** y es este el único que es capaz de modificar el estado del juego, y por tanto, el único que es capaz de comunicarse con las clases del modelo, por ello, el usuario jugador únicamente se comunica con sus clases controlador y estas a su vez con los controladores del servidor.

La razón por la que se ha utilizado el patrón MVC es porque de esta forma conseguimos separar la lógica del negocio de los datos y las interfaces de usuario, además, dado que la aplicación se basa en un software previamente desarrollado, se han tenido que adaptar algunas partes del contenido del mismo para seguir este patrón.

El sistema se representa **organizado por capas**, cada una de estas representa uno de los componentes de alto nivel del patrón MVC, es decir, tendremos el paquete **Vista**, el paquete **Controlador**, el paquete **Modelo** y la **Persistencia** además de otros paquetes de **Utilidades**. Para cada uno de estos paquetes se detallan las clases y funcionalidades que contienen en los apartados 6.3.3, 6.3.4 y 6.3.5, además se detallan las clases del motor de Unreal Engine utilizadas [42].

6.2.2 Vista lógica del sistema

En la **figura 51** se observa la composición del sistema estructurada por capas y organizada en paquetes. Es importante recalcar que se ha optado por mostrar tanto el sistema y componentes del cliente como los del servidor en una única vista unificada del sistema, aunque el sistema real esté formado por dos aplicaciones ligeramente diferentes, aunando las clases y componentes que interactúan vía red entre ambas en el paquete 'Controlador' que se detalla en el apartado 6.3.4.

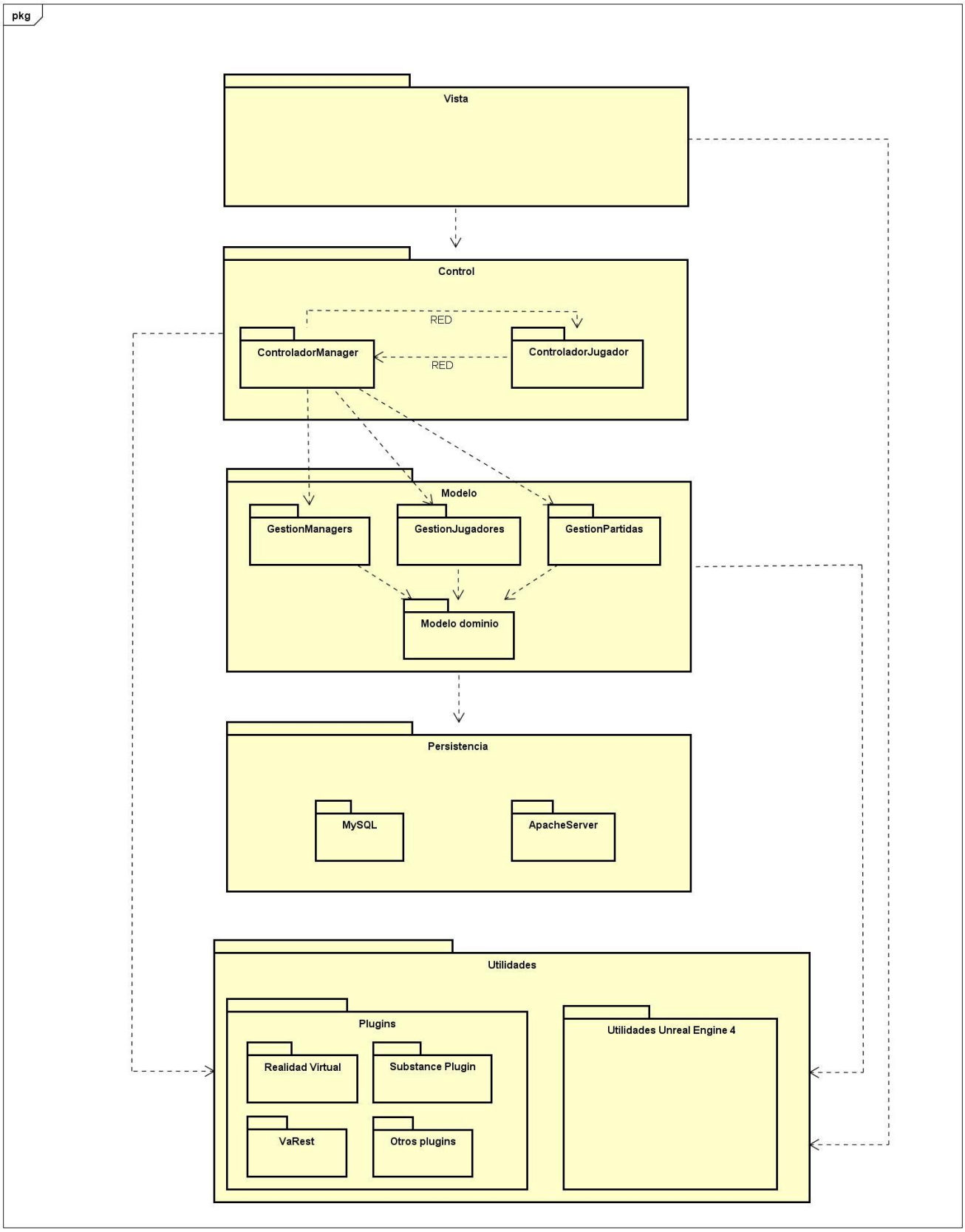


Figura 51: Vista lógica del sistema

6.2.3 Paquete Vista

Contiene el conjunto de interfaces y pantallas que utilizará el usuario manager en la aplicación, se especifican únicamente las clases desarrolladas en la aplicación, explicando únicamente las clases del motor de las que derivan. Para obtener más información detallada acerca de estas clases puede consultarse la documentación oficial de Unreal Engine 4 accesible desde [46].

Dado que el motor Unreal Engine 4 permite utilizar '***Eventos'*** en las clases de proyectos constituidos por blueprints y que funcionan de una forma parecida a las funciones de una clase, como el lenguaje UML no permite modelar este tipo de componentes de la forma más adecuada, en su lugar **se expresan estos eventos como funciones con argumentos**.

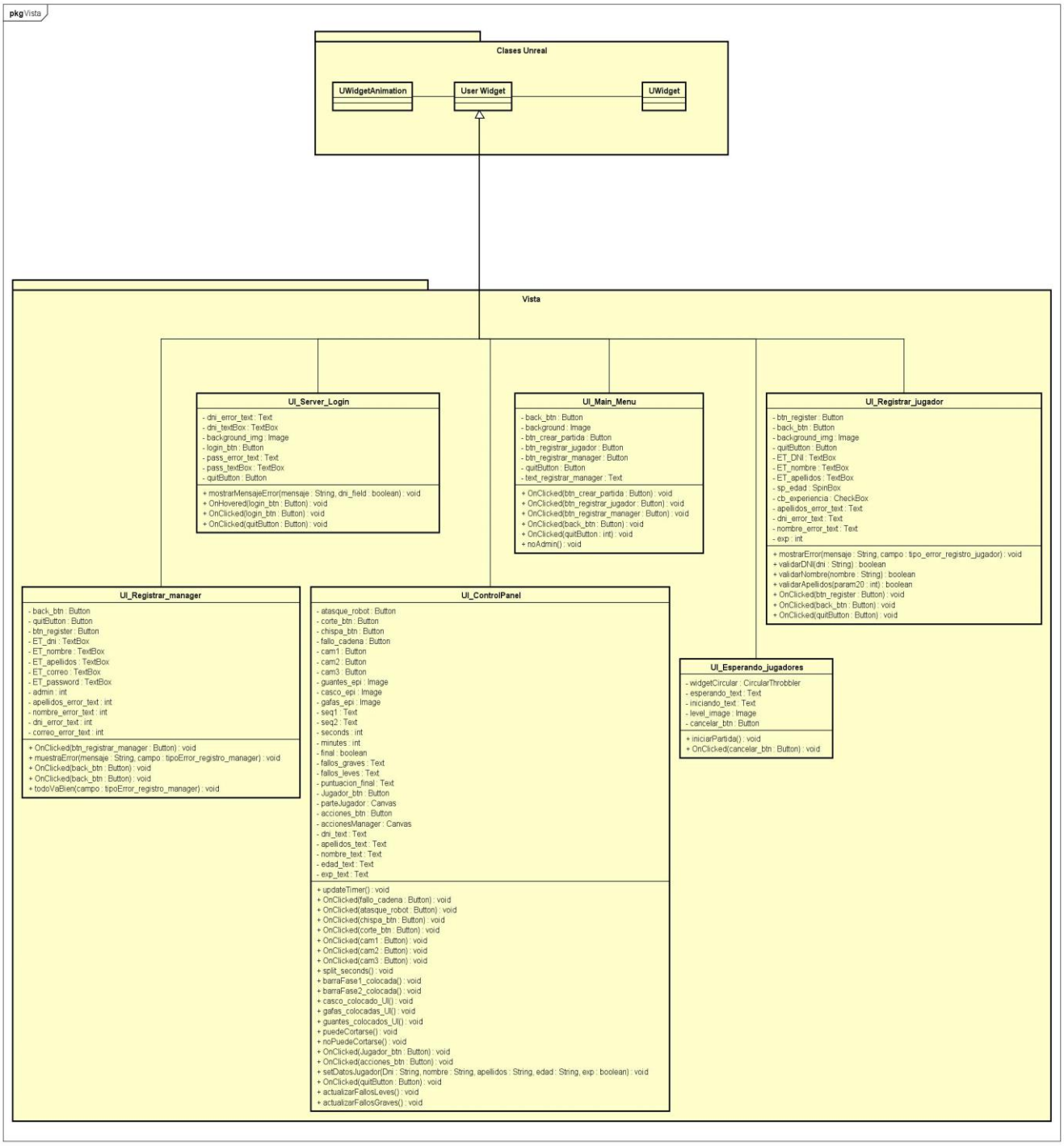


Figura 52: Detalle del paquete 'Vista' del sistema

6.2.3.1 UI_Server_Login

Se encarga de procesar las acciones de un usuario manager cuando este se identifica en el sistema. Sólo es utilizado en el lado servidor.

6.2.3.2 UI_Main_Menu

Clase encargada de procesar las acciones del usuario después de que esta se haya identificado en el sistema.

6.2.3.3 UI_Registrar_Jugador

Se encarga de procesar las acciones del usuario cuando este registra a un jugador en el sistema.

6.2.3.4 UI_Registrar_manager

Clase encargada de procesar las interacciones del usuario cuando este registra a un usuario manager.

6.2.3.5 UI_Esperando_Jugadores

Cumple la función de procesar las acciones del usuario manager cuando los jugadores se están uniendo a la sesión de juego.

6.3.3.6 UI_ControlPanel

Es el corazón de la aplicación y se encarga de procesar todas las acciones del usuario desde que se inicia una partida multijugador hasta que esta finaliza.

6.2.4 Paquete Controlador

El paquete de clases, funcionalidades y componentes que cumplen la función de controladores dentro del sistema **se compone por clases del motor Unreal Engine** y por otras **clases que son las desarrolladas en la aplicación**. Hay que recordar que en el motor Unreal Engine tanto cliente como servidor poseen una copia de la mayoría de clases de la aplicación, pero tienen diferentes privilegios, por ejemplo, un servidor tendrá una instancia de la clase ‘Pawn’ y el cliente tendrá otra instancia de la misma clase pero ambas son copias locales y la modificación en el lado cliente no implica la modificación en el lado servidor, sin embargo, existen otras clases como aquellas que derivan de la clase ‘GameMode’ que únicamente existen en el lado del servidor y que persisten a lo largo de la partida.

6.3.4.1 Clases del motor Unreal Engine 4

Se representan algunas de las clases más relevantes a la hora de entender la aplicación, la funcionalidad de estas clases puede consultarse en la documentación oficial de Unreal Engine 4 accesible desde la página oficial de referencia de la API [46].

6.3.4.2 Clases de la aplicación

Se trata de las clases que han sido implementadas en la aplicación o a las que se le han añadido nuevas funcionalidades. Estas clases tienen la característica de que heredan de otras clases básicas del motor y que se ha considerado oportuno el representarlas en la figura XX, además en la figura XXI puede verse con mayor detalle las clases desarrolladas en la aplicación y la relación entre ellas.

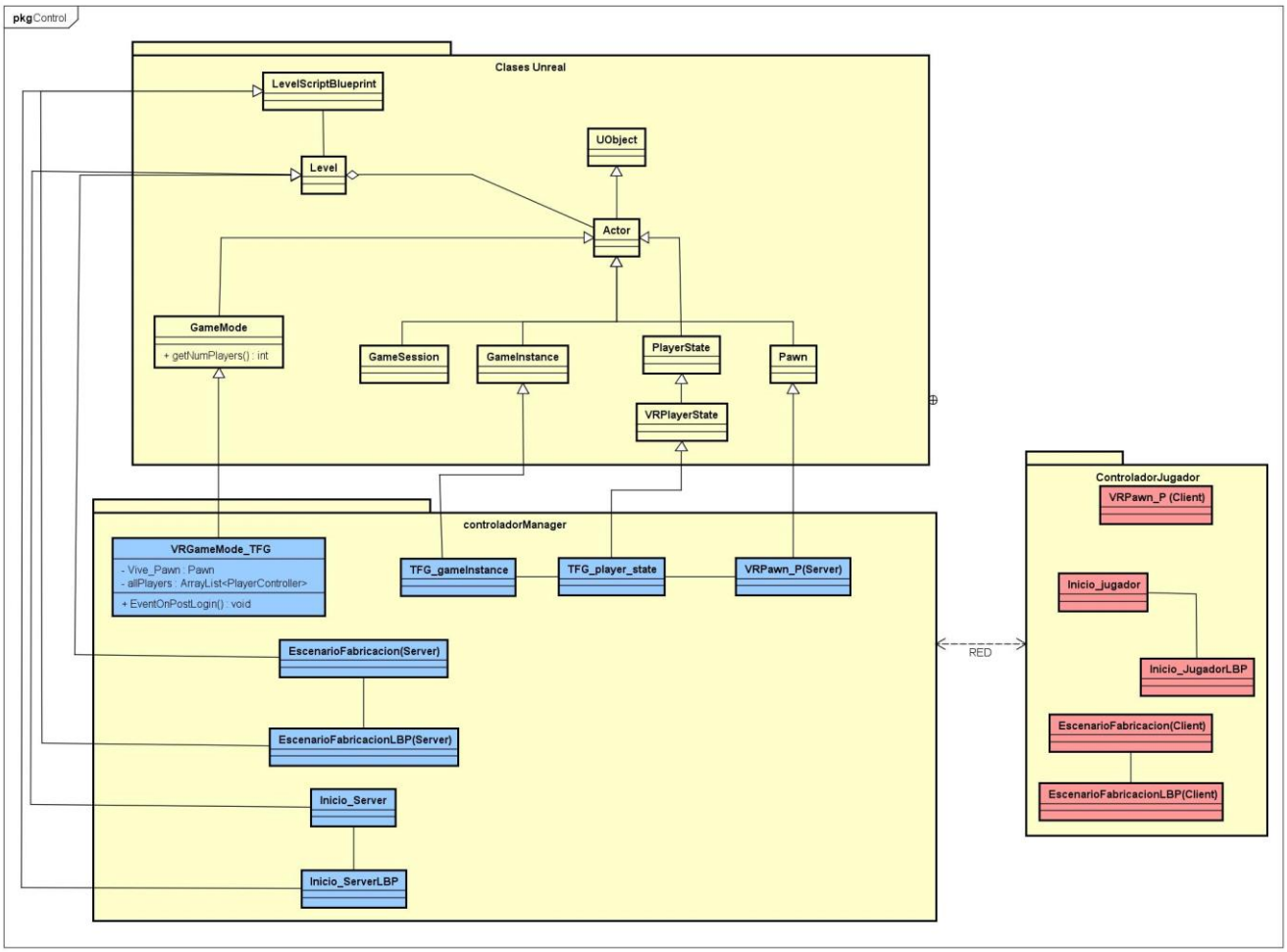


Figura 53: Clases del paquete 'Controlador' sin detalle junto con las clases correspondientes del motor

- **VRGameMode_TFG**: se trata de una clase de tipo ‘GameMode’, sólo existe en el lado del servidor y sirve como punto de entrada inicial para determinar qué debe hacerse cuando un cliente entra en la sesión a partir del evento ‘**OnPostLogin’**.

- **TFG_player_state**: es una clase que desciende de ‘**Player_State** y actúa como el **controlador principal de las clases de la vista** de la aplicación servidor. Aunque a priori rompe un poco el esquema MVC, ya que se utilizamos un controlador para múltiples vistas, es necesario hacerlo en entornos multijugador, ya que de hacer varios controladores para las vistas estaríamos sacrificando gran parte del rendimiento de la aplicación en pos de forzar al motor a seguir este tipo de patrón y favorecer un posterior mantenimiento.

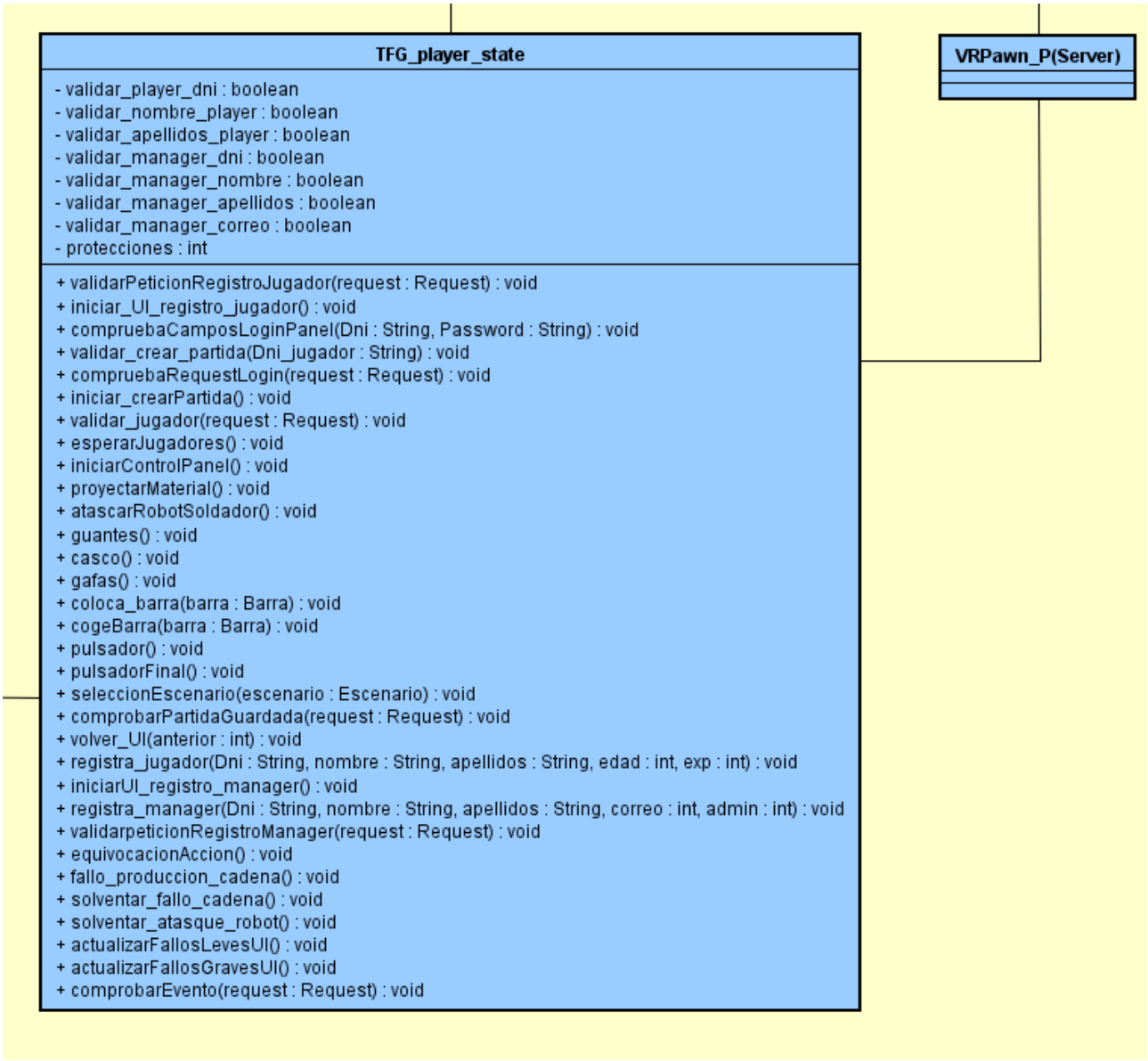


Figura 54: Detalle de la clase 'TFG_player_state'

- **VRPawn_P (Servidor)**: Se trata del pawn que utilizará un usuario manager una vez se esté ejecutando la aplicación y que desciende directamente de la clase ‘Pawn’ del motor. Será la encargada de recibir los mensajes que se envíen desde el cliente al servidor (mensajes RPC) ya que el motor está configurado de tal forma que tan sólo permite la comunicación inversa, es decir, sólo pueden enviarse mensajes desde el cliente hacia el servidor en componentes que desciendan de la clase ‘Pawn’. Tanto el Pawn del cliente como el pawn del servidor tienen las mismas operaciones y funcionalidades.

- **TFG_game_instance**: es la clase que deriva de ‘**GameInstance’** y representa a una instancia el propio juego, se encarga de guardar todas aquellas variables que indiquen un cambio de estado en el usuario y que son relevantes durante el uso de la aplicación. Es la única clase que persiste entre niveles y sólo existe en el lado del servidor.

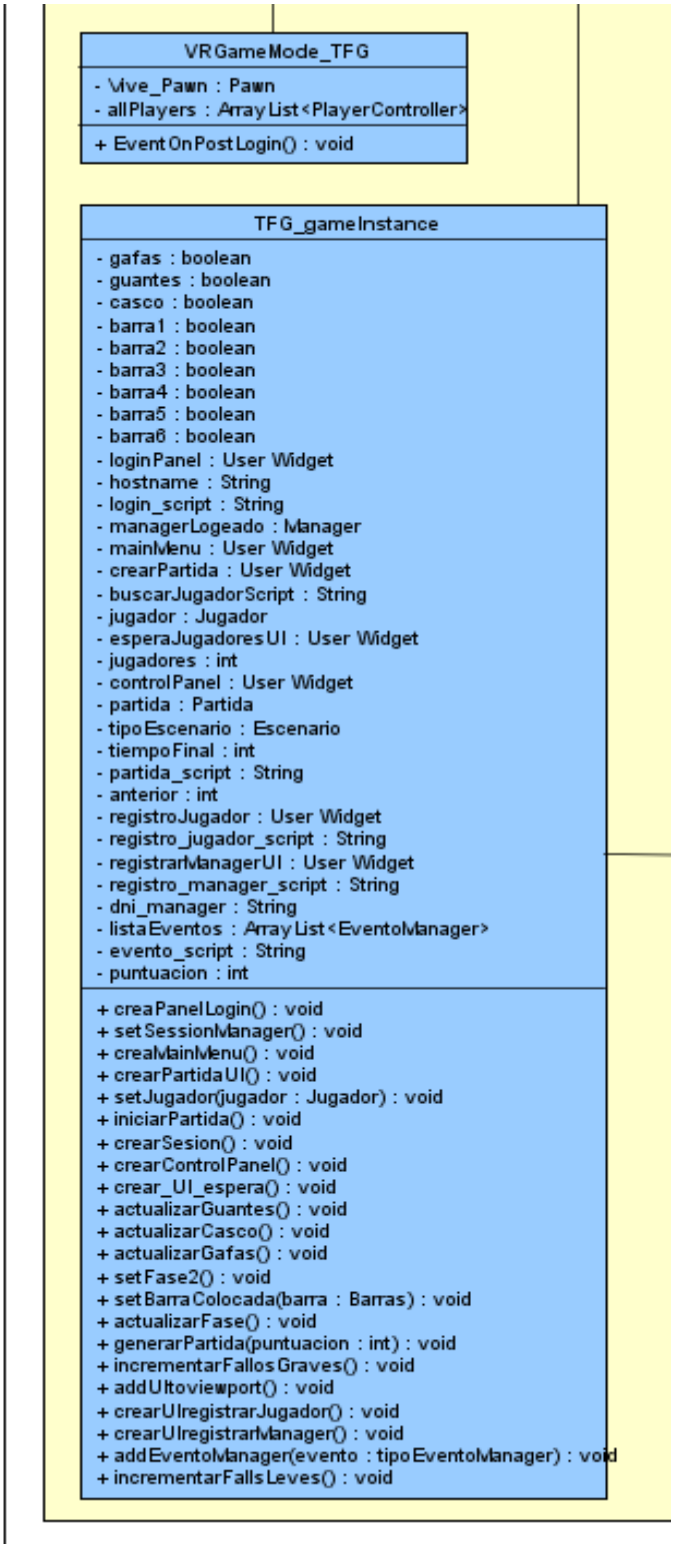


Figura 55: Detalle de las clases 'TFG_gameInstance' y de la clase 'VRGameMode_TFG'

- **VRPawn_P (Cliente):** Utiliza la misma clase base de tipo 'Pawn' que utiliza el servidor y comunica aquellos elementos de relevancia para el servidor mediante **mensajes RPC**. Contiene toda la **funcionalidad del usuario jugador** en el escenario. Es una copia del Pawn que utiliza el servidor.

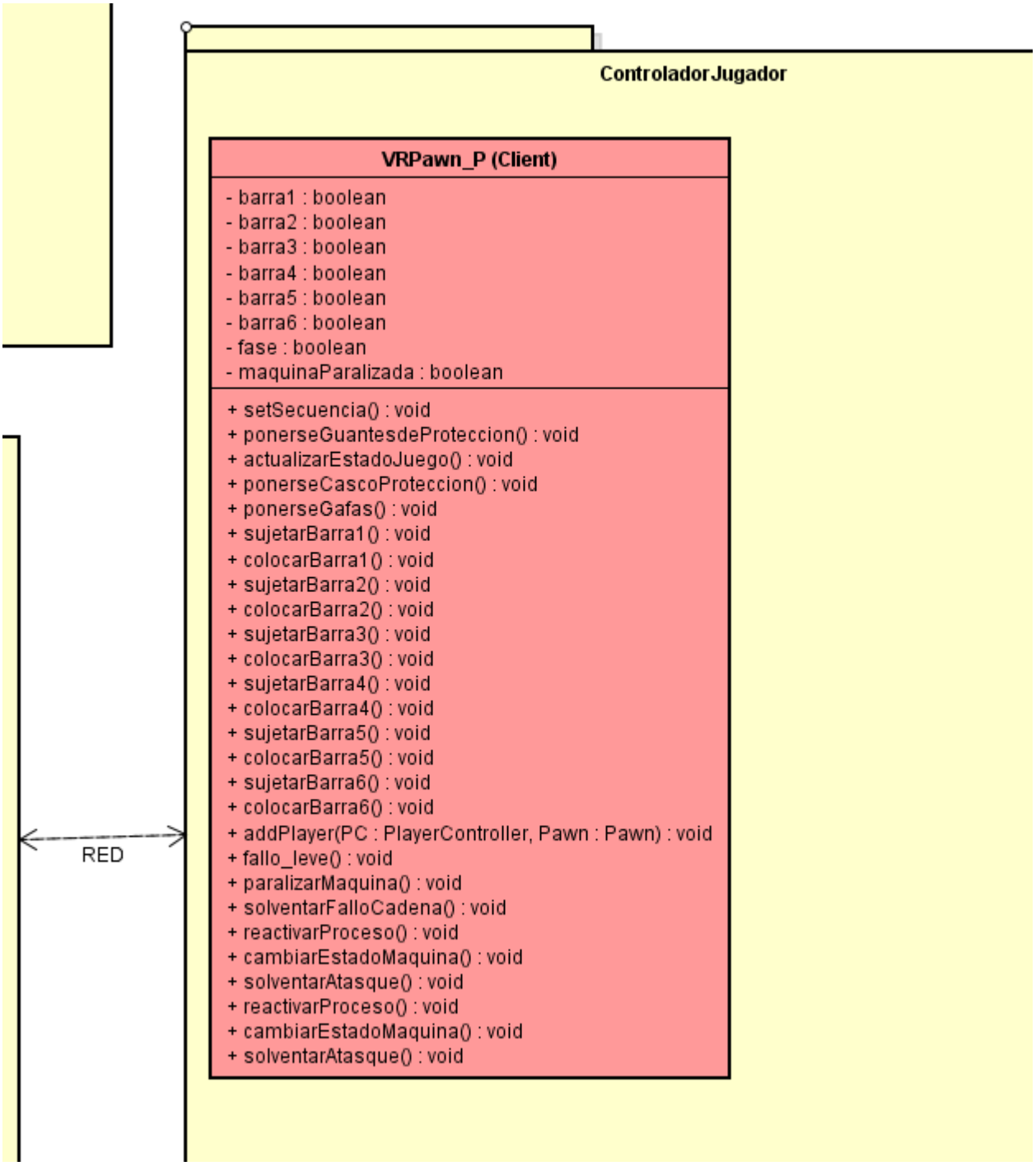


Figura 56:Detalle de la clase VRPawn_P del lado cliente

- **VRPlayerController_P**: Esta clase se presenta tanto en el cliente como en el servidor y representa una interfaz entre el objeto 'Pawn' y el usuario real y que persiste durante todo el juego en la aplicación.

- **Inicio_Server**: Se trata del nivel de inicio de la aplicación cliente. Está formado por un espacio vacío y los actores básicos de un nivel. Sirve como punto de entrada de la aplicación para poder navegar desde este a otros niveles como el escenario de fabricación o a otros futuros niveles.

- **Inicio_Jugador**: Es el nivel de inicio del jugador, desde este se proporciona un entorno gráfico que informa al jugador de si se están buscando sesiones en red e informar de cuando se une a una de ellas.

- **EscenarioFabricacion**: Es un nivel de juego que contiene todos los actores que constituyen el escenario de fabricación de la aplicación, y que será uno de los escenarios donde se desarrolle la acción principal de la aplicación.

- **Inicio_ServerLBP**: Se trata del blueprint del nivel 'Inicio_Server' y contiene la funcionalidad necesaria para iniciar la creación de ventanas gráficas en el servidor.

- **Inicio_JugadorLBP**: Contiene la funcionalidad necesaria para hacer que un usuario jugador busque sesiones vía red y pueda unirse a estas.

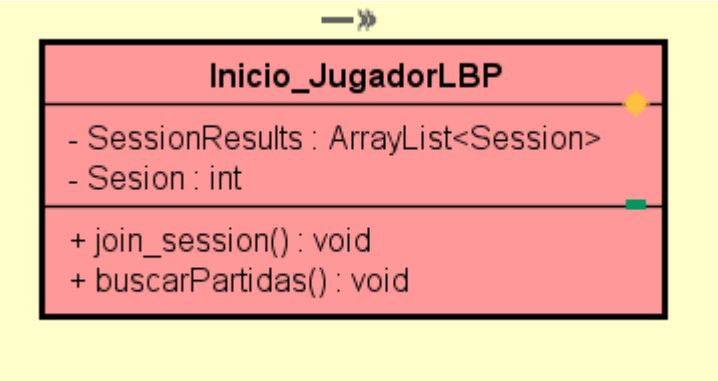


Figura 57: Detalle de blueprint del nivel 'Inicio_Jugador'

- **EscenarioFabricacionLBP**: Es un nivel de juego donde un usuario manager puede controlar a un usuario jugador a través del panel de control. Este nivel ha sido previamente desarrollado en el software sobre el que se basa esta aplicación y al que se le han añadido y modificado nuevas funcionalidades.

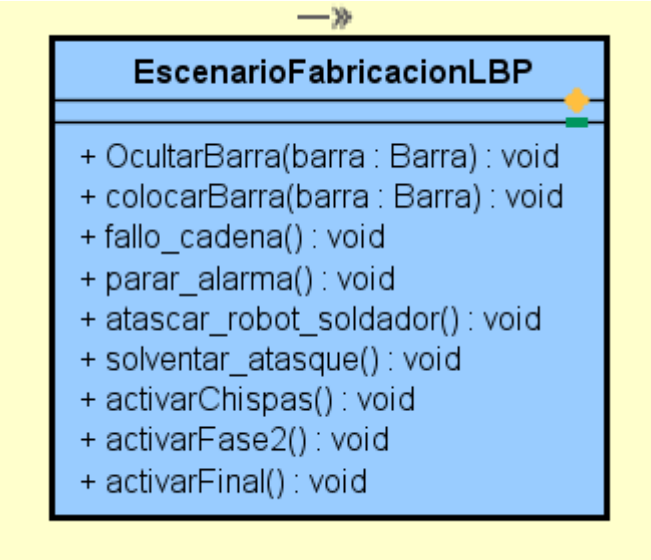


Figura 58: Detalle del blueprint del nivel 'EscenarioFabricacion'

6.2.5 Paquete Modelo

A continuación, se resumen las clases y componentes pertenecientes a la capa del modelo que intervienen en la aplicación. Es importante apuntar que las clases del dominio en la aplicación son **representadas a través de 'Structs' en lenguaje C++** y, por tanto, carecen de operaciones como tal, sin embargo, se considera a estas clases como **Entidades del sistema**, ya que representan a las clases del dominio y son utilizadas por el resto de capas. Además, los gestores de la aplicación son representados mediante clases que derivan de '**Function Library**' y que tampoco son clases como tal, sino que únicamente albergan funciones. También en este paquete encontramos algunas **clases enumerables** que serán de ayuda a la hora de realizar la implementación de la aplicación.

- **gestorManagers**: esta clase se encarga de la gestión de la clase '**Manager'** del dominio de la aplicación. A través de este controlador se accede a la base de datos mediante peticiones por URL.
- **gestorJugadores**: se encarga de la gestión de operaciones referentes a la clase '**Jugador'**. Es utilizada por las capas superiores para el acceso a persistencia.
- **gestorPartidas**: clase encargada de gestionar las operaciones relacionadas con las clases **'Partida' y 'EventoManager'**, dado que un evento del manager depende de una entidad partida, por ello se ha optado por utilizar un único gestor que reúne las funcionalidades de las dos clases del dominio.

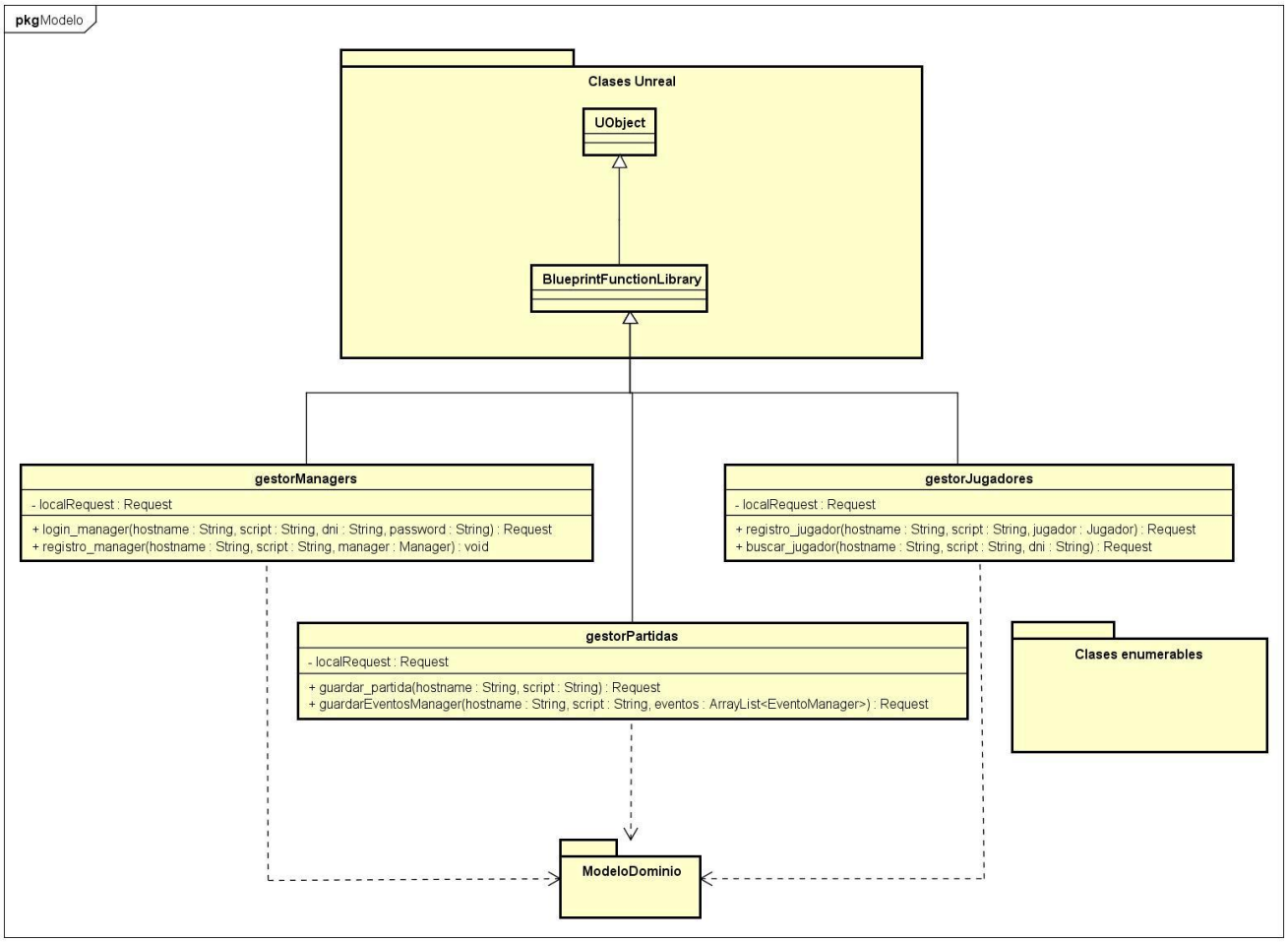


Figura 59: Detalle del paquete 'Modelo' del sistema

El sistema también contiene algunas clases **enumerables** en este paquete:

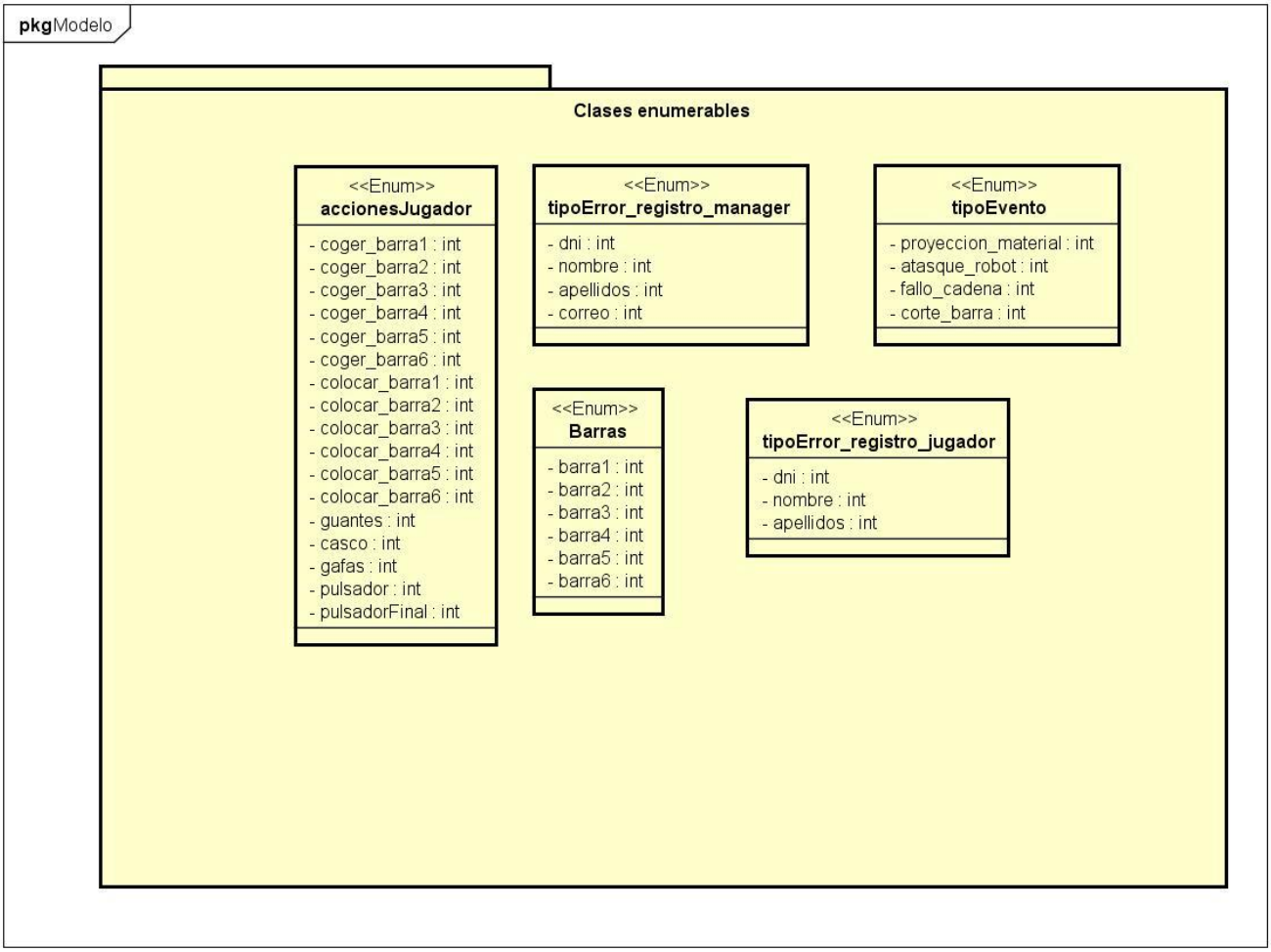


Figura 60: Detalle de las clases enumerables de apoyo del sistema

6.5 Modelo de Entidad – Relación

Las entidades que aparecen en el modelo son:

- **Manager**: Representa a un usuario que actúa como servidor y que utiliza el panel de control habiéndose registrado como tal en la aplicación. Se identifica a través de su documento de identidad (DNI). Una entidad manager debe guardar información sobre:
  - DNI
  - Contraseña
  - Nombre
  - Apellidos
  - Correo
  - Si tiene permisos de administrador

- **Jugador**: Representa a un jugador de la aplicación, este también se identifica en la base de datos a través del documento de identidad o dni (clave primaria). Una entidad 'jugador' debe guardar información acerca de:
  - Dni
  - Nombre

    - Apellidos
    - Edad
    - Si tiene experiencia previa

- **Partida**: Representa a una partida online llevada a cabo entre cliente y servidor en máquinas distintas. Una partida únicamente tiene un jugador, mientras que un jugador puede tener múltiples partidas, además una partida también tiene un solo manager, mientras que un manager puede controlar múltiples partidas. Una partida debe se identifica a través de un identificador único que actúa como clave primaria, 'id_partida' y guarda los siguientes datos:
    - El tiempo transcurrido desde que inicia la partida hasta que termina.
    - La puntuación obtenida por el jugador.
    - La fecha en que se realizó la partida.
    - El número de fallos del jugador durante la partida

- **Evento Manager**: Representa a una acción o evento que realiza el manager a través del panel de control y que queda registrado en el historial de la partida, se identifica a través de un identificador 'id_evManager' y además con el identificador de la partida en la que se ejecutó la orden 'id_partida', es decir que su clave primaria está formada por su identificador y la clave foránea de la partida, además un evento del manager debe contener información acerca de:
    - Momento en que se realiza
    - El tipo de evento realizado, de tipos:
        - Proyección de material
        - Atasque en el robot soldador
        - Fallo de la cadena de producción
        - Corte por barra

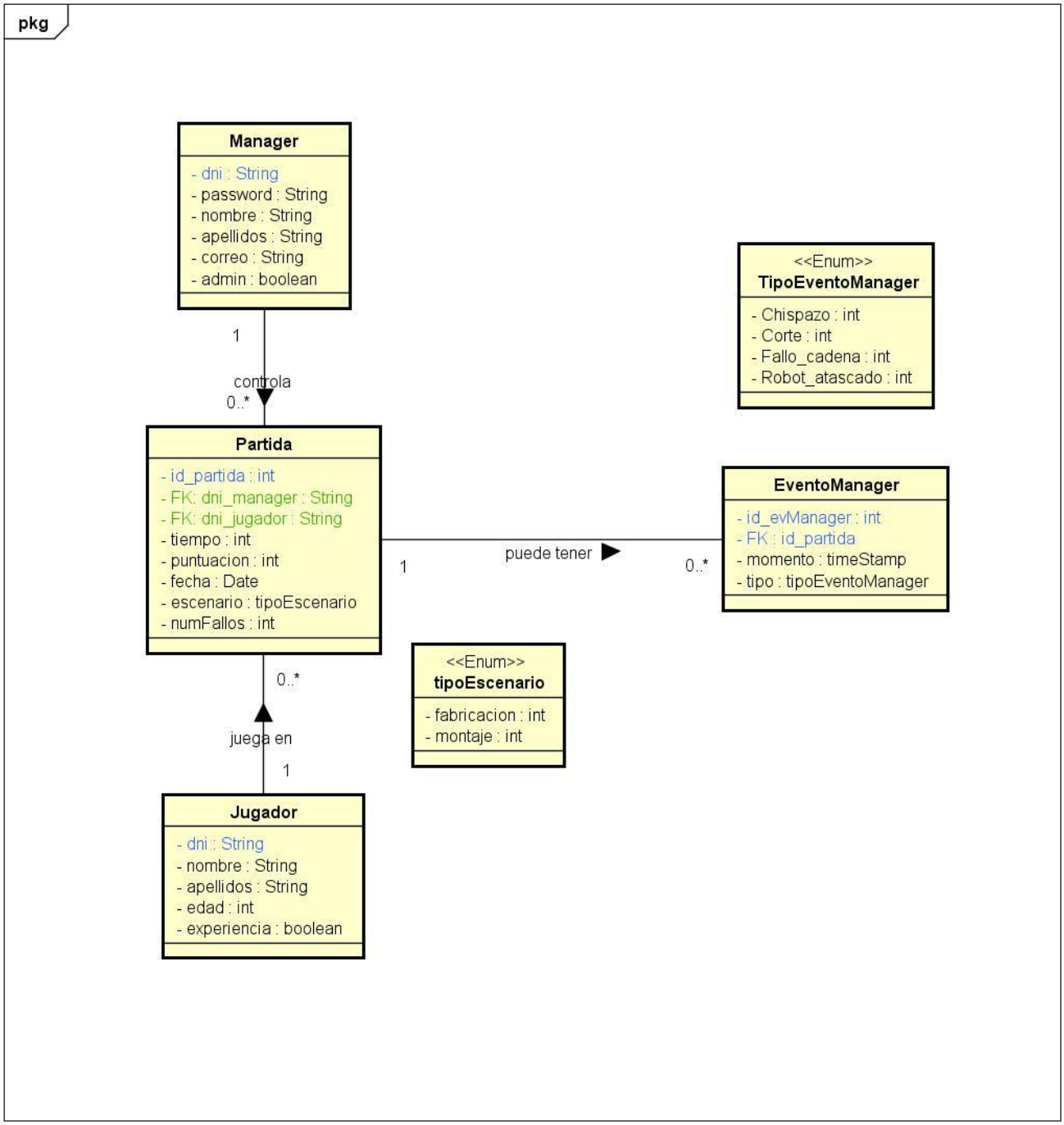


Figura 61: Modelo Entidad - Relación de la base de datos

6.6 Modelos de secuencia

En esta sección, se muestran algunos de los diagramas de secuencia que se han considerado más relevantes a la hora de comprender el funcionamiento de la aplicación. Se expondrán los diagramas para los casos de uso más relevantes:

6.6.1 CU-01: Registrar manager

Un usuario administrador puede registrar a un usuario manager a través de la interfaz:

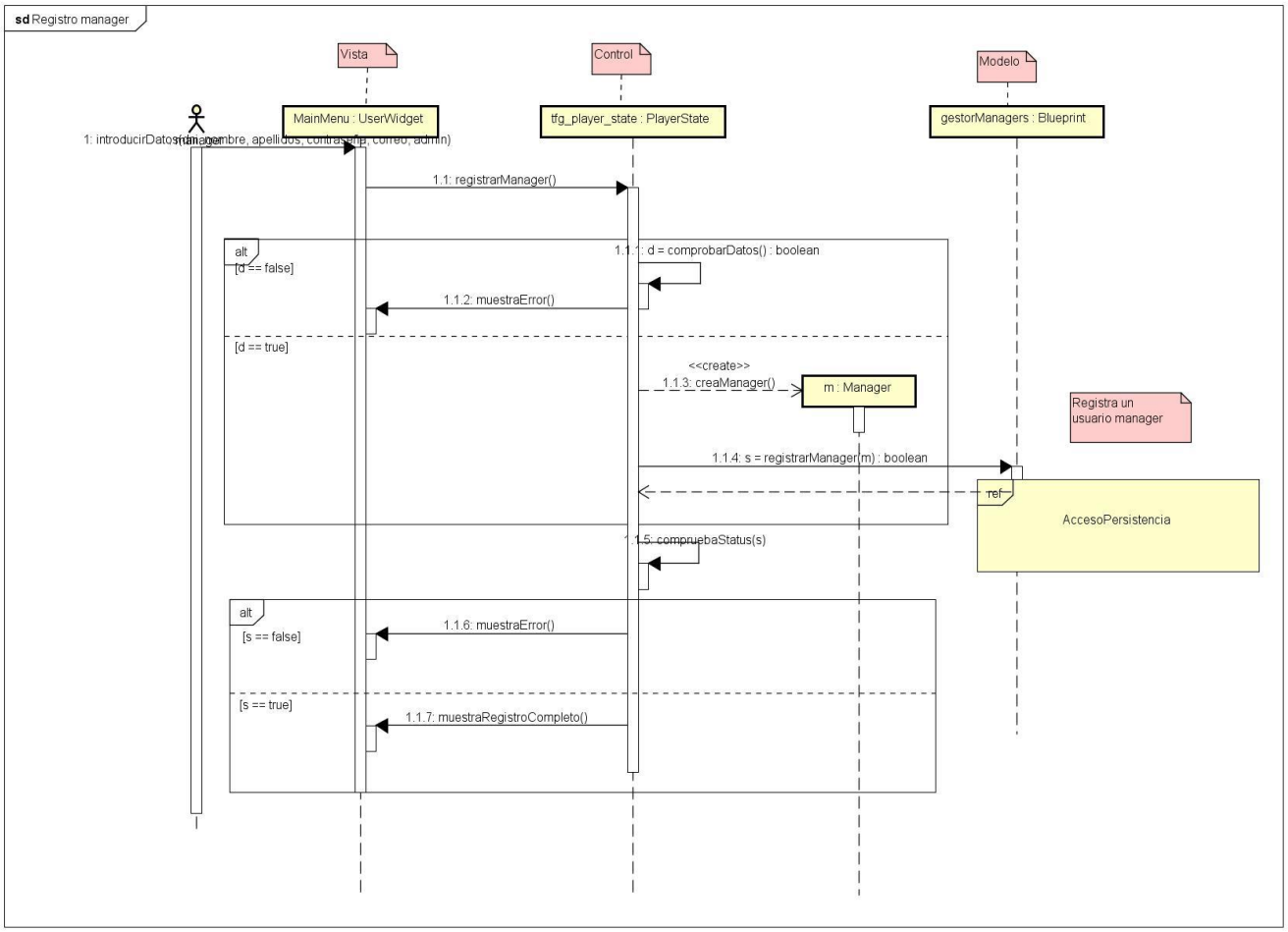


Figura 62: Diagrama de secuencia del caso de uso 'CU-01: Registrar manager'

6.6.2 CU-02: Iniciar Sesión

En este caso de uso un usuario manager inicia sesión en el sistema a través de la interfaz inicial de la aplicación:

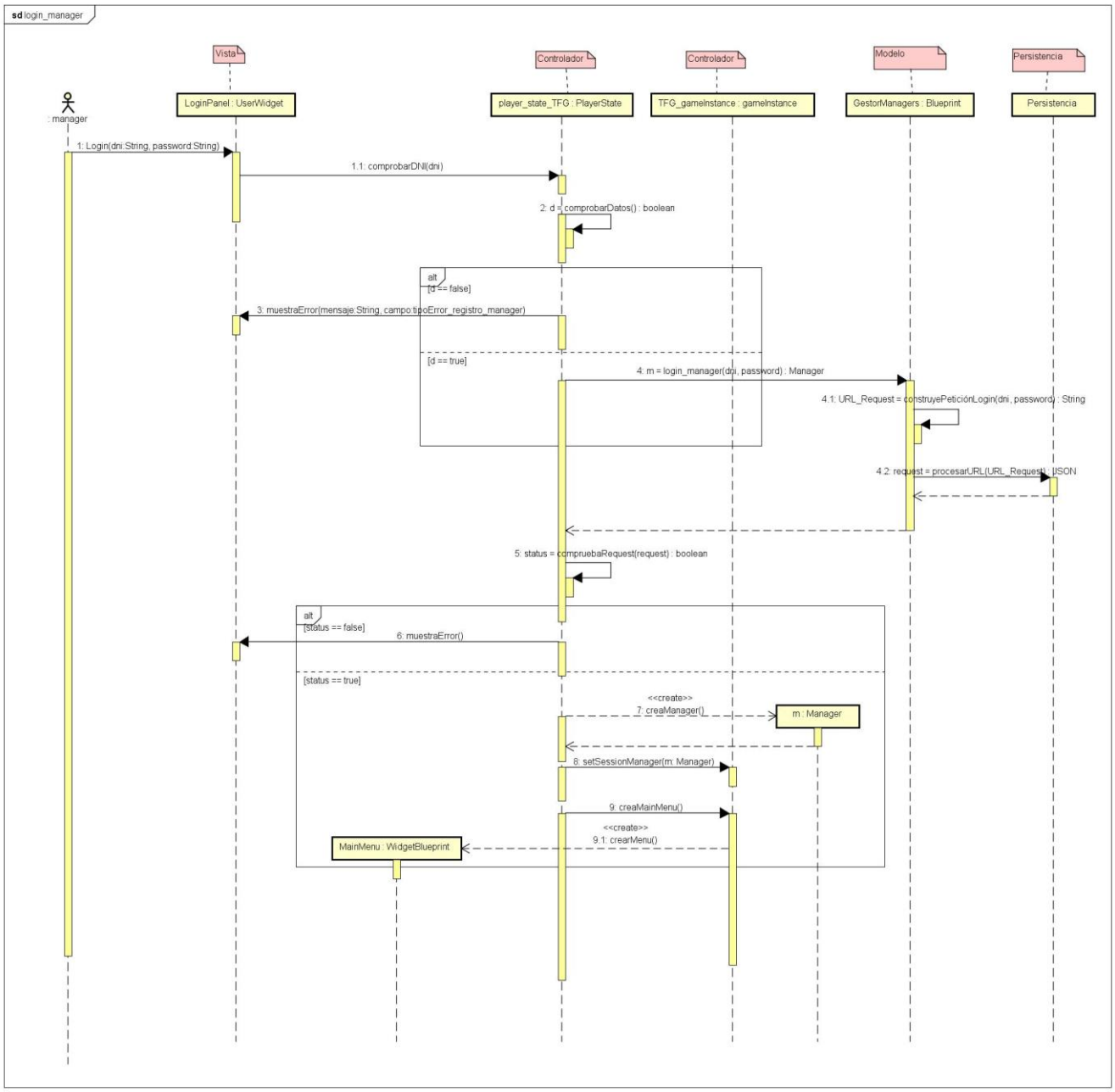


Figura 63: Diagrama de secuencia del caso de uso 'CU02 - Iniciar sesión'

6.6.3 CU-03: Crear partida

Este caso de uso se describen las operaciones que realiza el sistema cuando un usuario manager decide crear una partida.

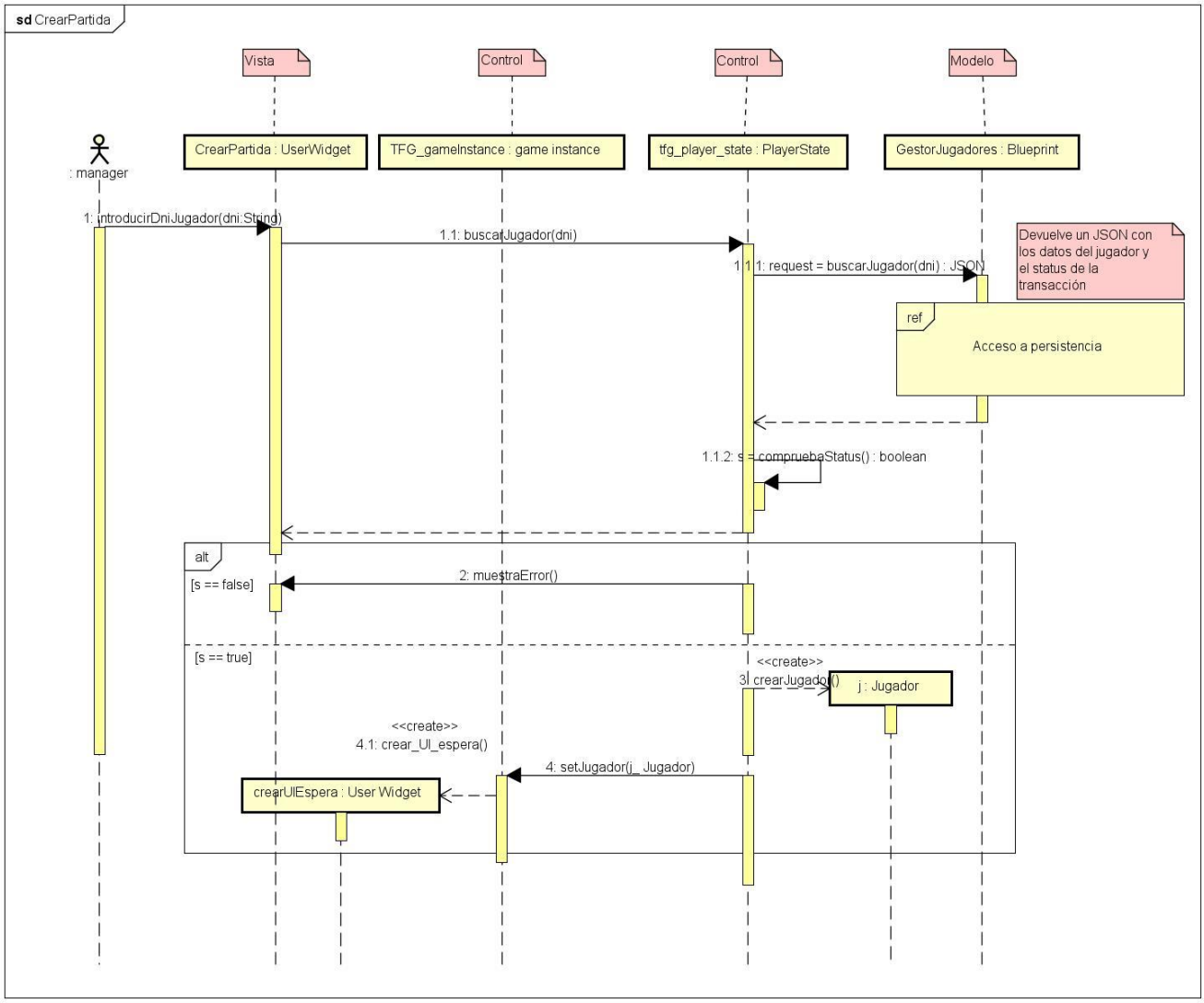


Figura 64: Diagrama de secuencia del caso de uso 'CU-03: Crear partida'

6.6.4 CU-04: Registrar jugador

En este diagrama de secuencia se describe el caso en el que el usuario manager registra a un jugador.

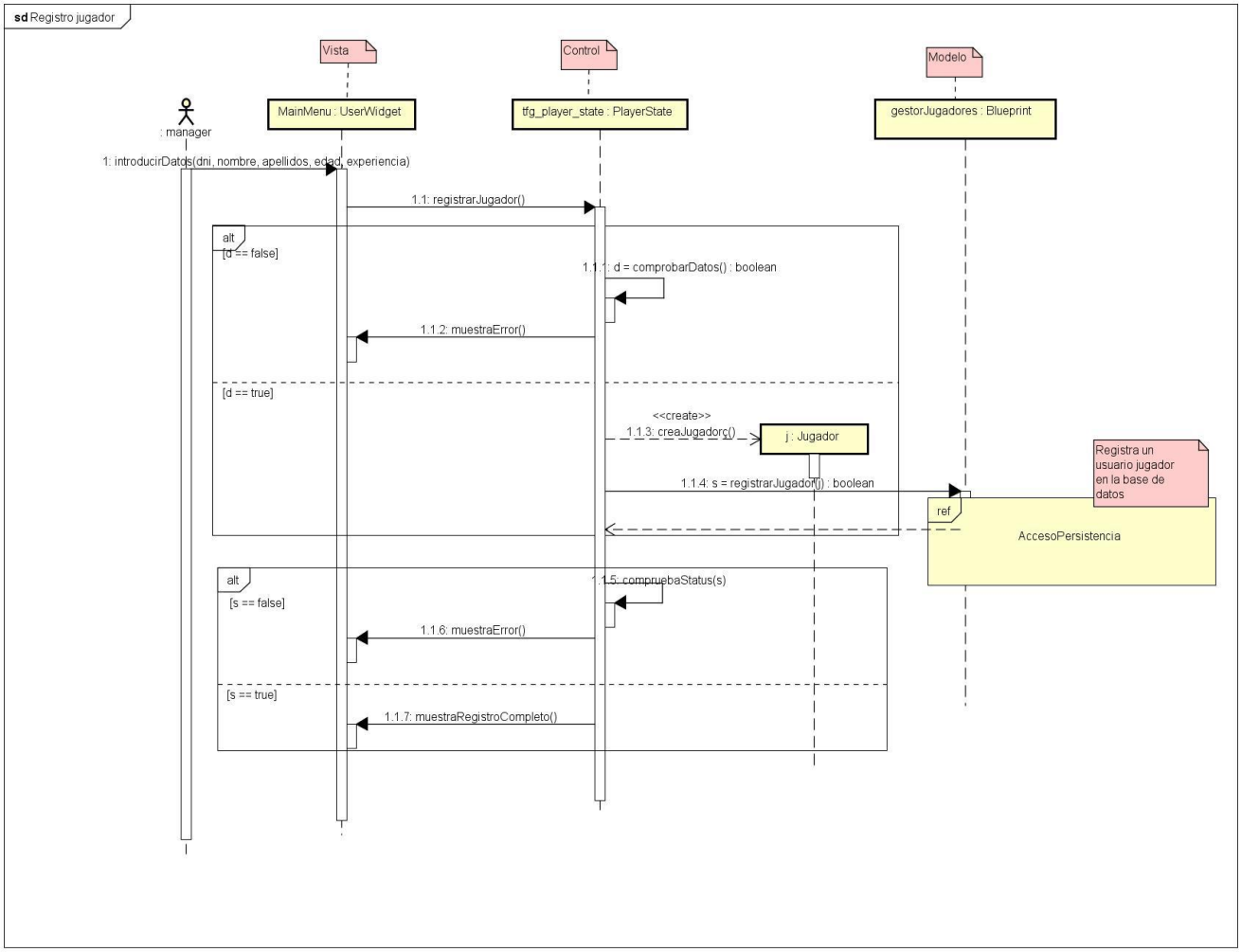


Figura 65: Diagrama de secuencia del caso de uso 'CU-04: Registrar jugador'

6.6.3 CU-05: Acción Proyección de material

En este caso de uso el usuario manager realiza la acción de saltar proyección de material de uno de los robots soldadores de la máquina principal del escenario. En el evento referenciado se liga el event dispatcher ‘spawnSpark’ desde el blueprint del nivel del escenario de fabricación y se crean las partículas sobre el robot.

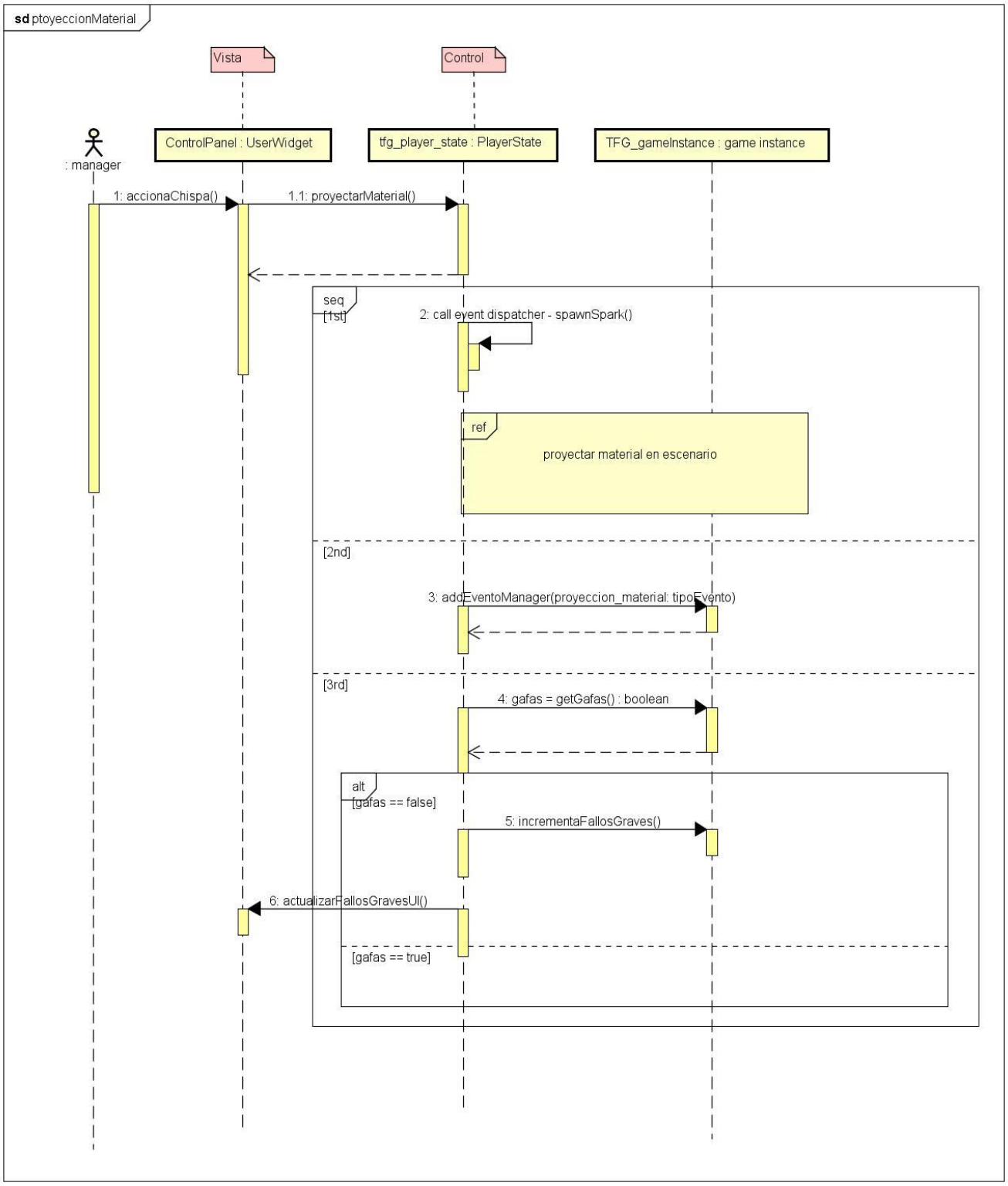


Figura 66: Diagrama de secuencia del caso de uso 'CU-05: Acción: Proyección de material'

6.6.6 CU-06: Atasque en el robot soldador

En este caso de uso el usuario manager acciona un atasque en el escenario de fabricación, impidiendo que el jugador pueda seguir con el proceso de fabricación hasta que solvente el fallo. En el evento referenciado se liga el event dispatcher ‘stopMachine’ desde el blueprint del nivel del escenario de fabricación y muestra el atasque en la máquina, actualizando el estado del juego.

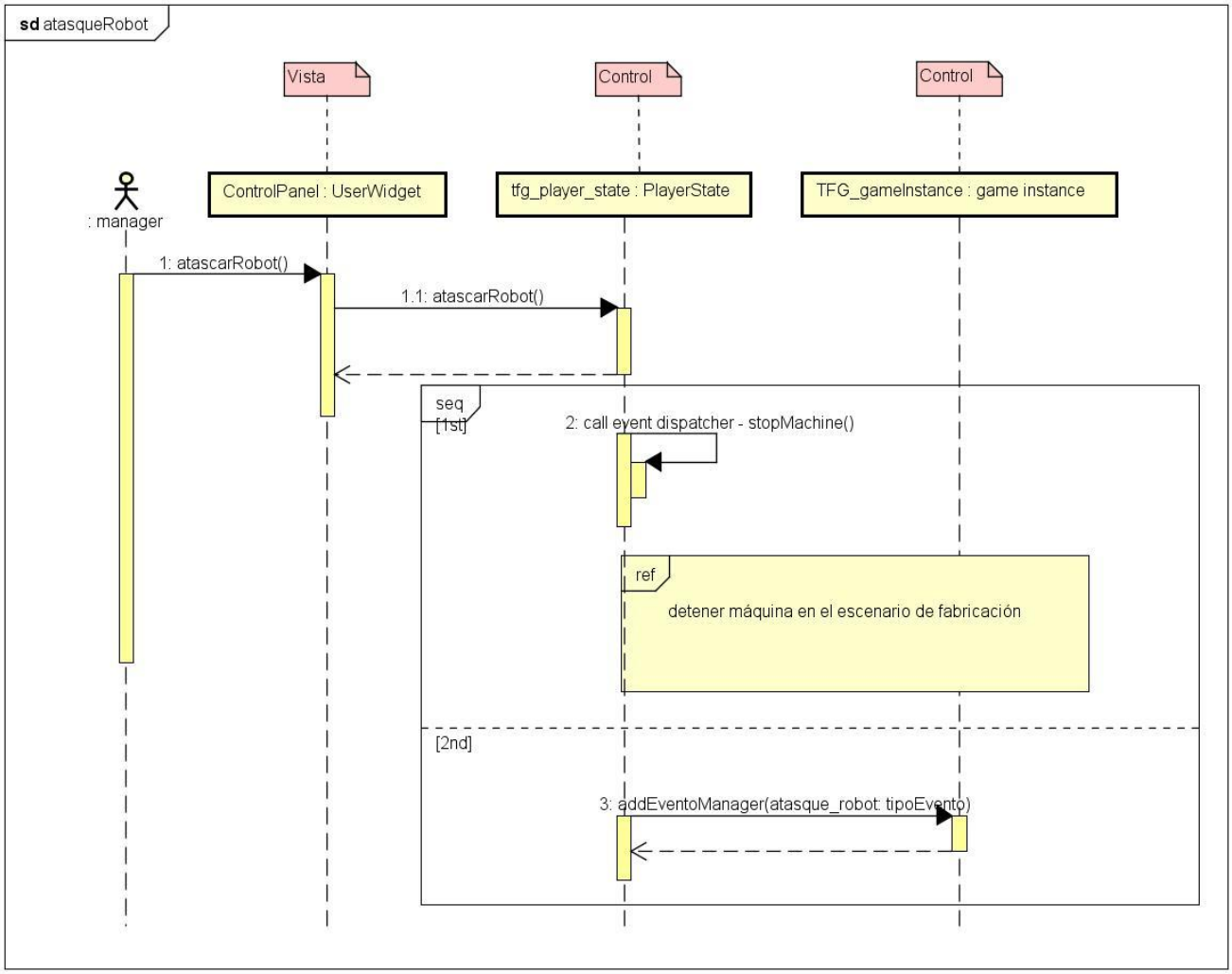


Figura 67: Diagrama de secuencia del caso de uso 'CU06 - Atasque en el robot soldador'

6.6.7 CU-07: Fallo en la cadena de producción

En este caso de uso el usuario manager acciona un fallo en la cadena de producción del escenario de fabricación, haciendo que el jugador que se encuentra en la partida activa tenga que desactivarlo antes de poder seguir con el proceso de fabricación. En el evento referenciado se liga el *event dispatcher* 'fallo_cadena_ED' en el *level blueprint* del escenario de fabricación que activa la luz parpadeante del escenario y actualiza el estado del juego.

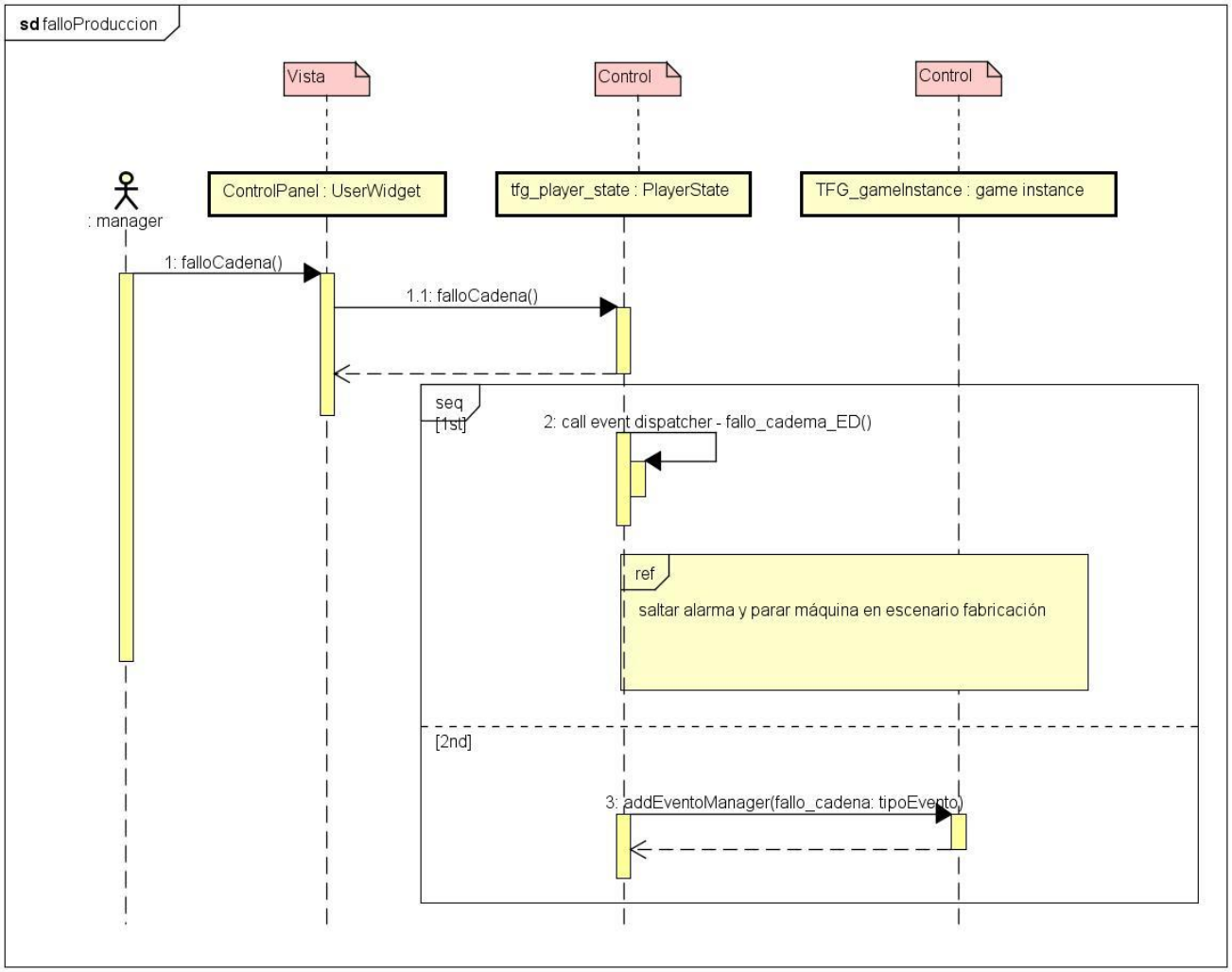


Figura 68: Diagrama de secuencia del caso de uso 'CU07 - Acción: Fallo en la cadena de producción'

6.6.8 CU-08: Corte por barra metálica

En este caso de uso, el usuario manager ejecuta la acción de corte sobre el jugador en una sesión activa, teniendo que en el evento referenciado se liga el event dispatcher del controlador tfg_player_state en el VRPawn_P del jugador, cambiando este la textura de la malla de la mano izquierda del componente gráfico del Pawn.

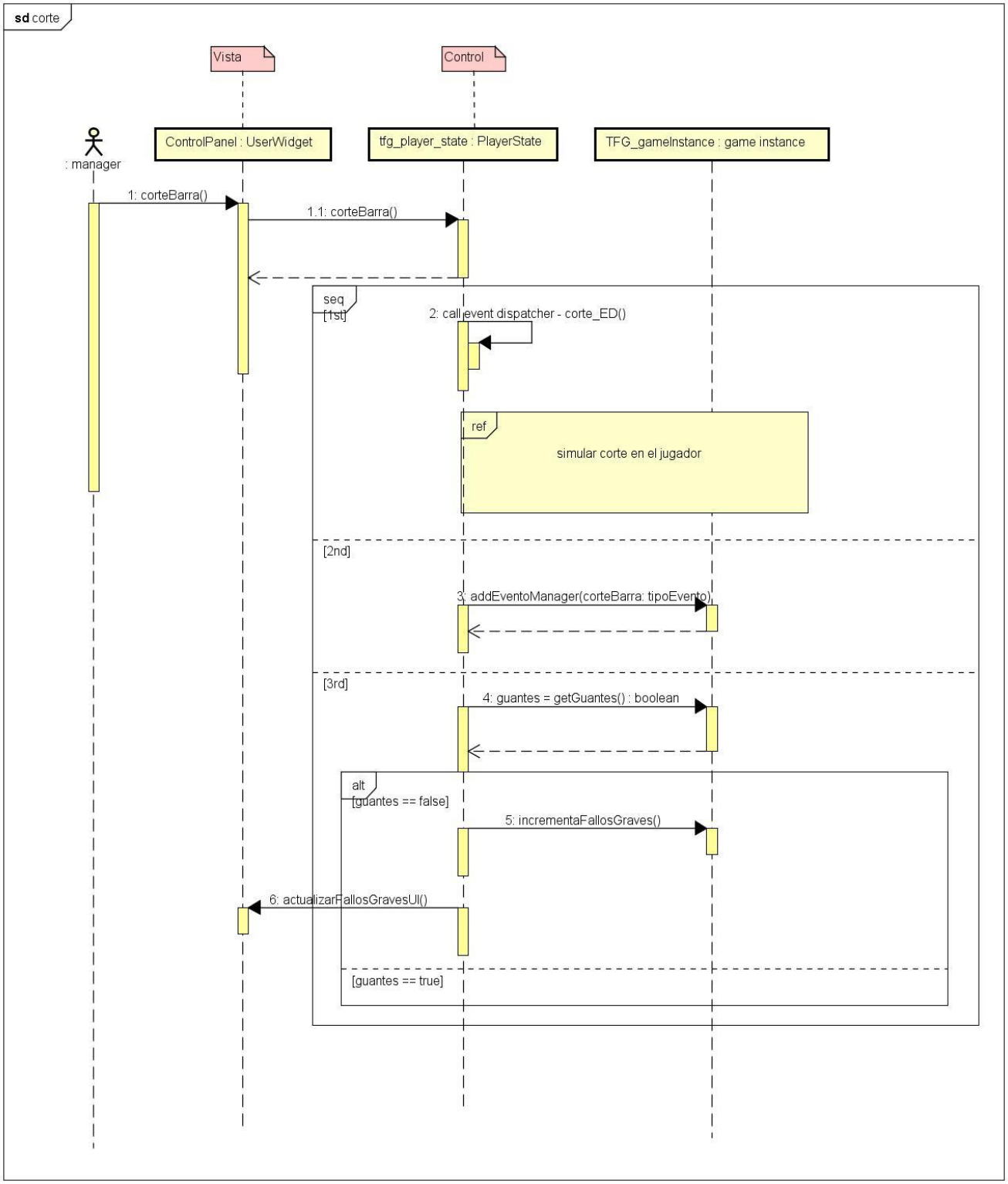


Figura 69: Diagrama de secuencia del caso de uso 'CU08 – Acción: Corte por barra metálica'

6.6.9 CU-09: Cambiar la vista del minimapa

Este caso de uso consiste en que el usuario manager cambie la perspectiva desde la que observa el escenario en el que juega el jugador a través del panel de control. Se explicará su implementación detallada en el capítulo 7 ‘Implementación’ ya que están implicadas muchas clases y componentes del propio motor.

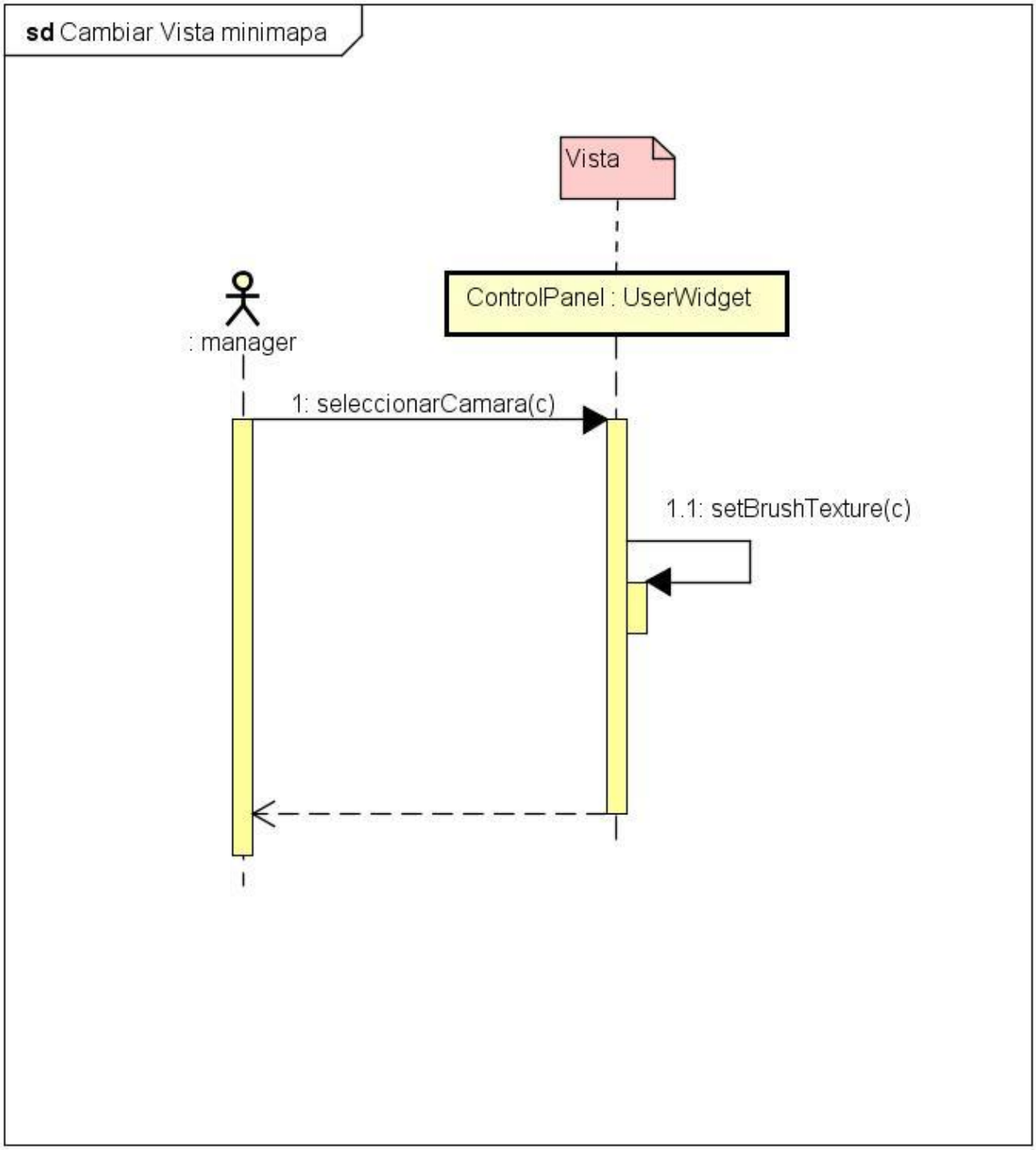


Figura 70: Diagrama del caso de uso 'CU09 - Cambiar la vista minimapa'

6.6.10 CU-10: Consultar información del jugador

En este caso de uso el usuario manager consulta la información de un jugador que se encuentra en una partida activa.

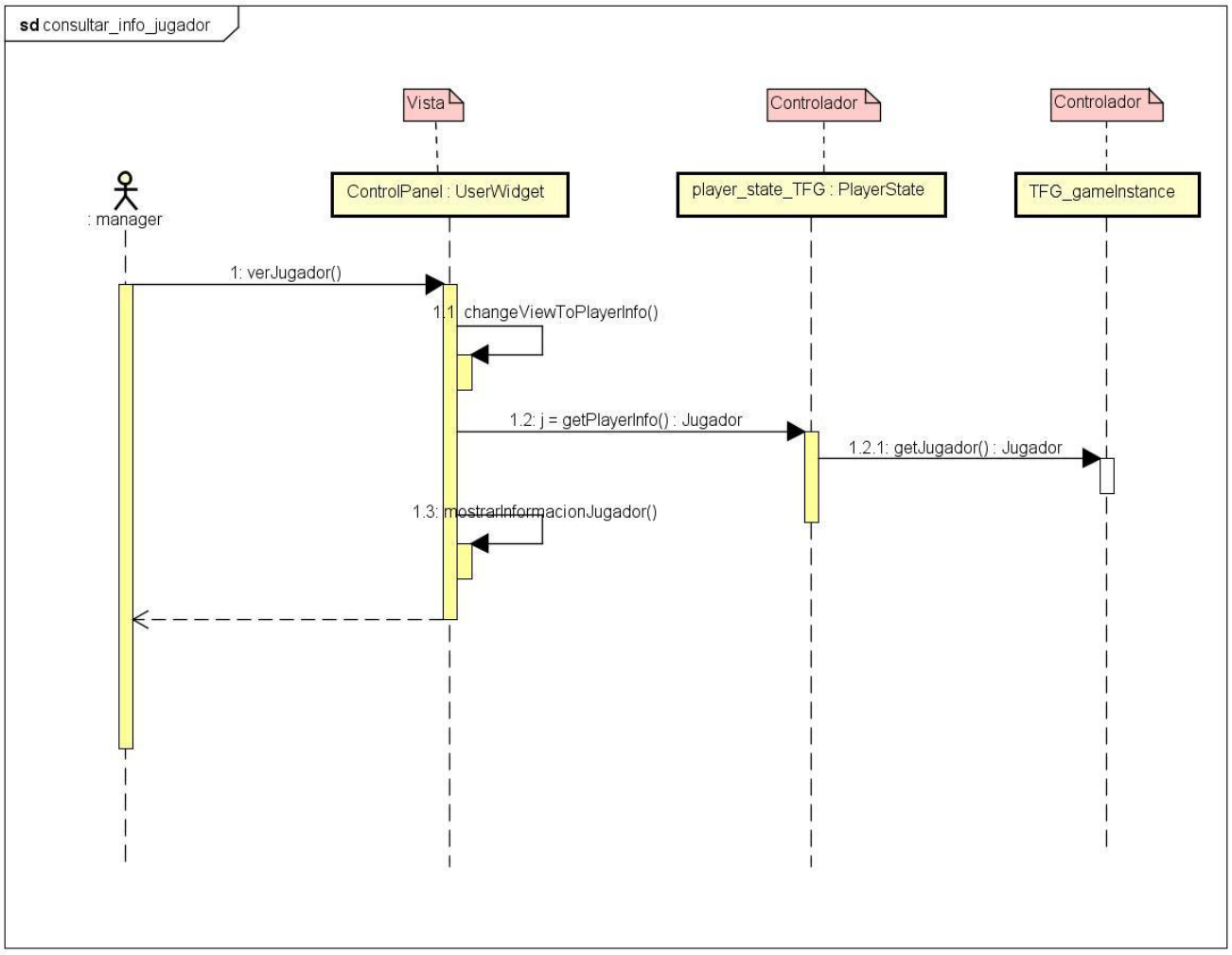


Figura 71: Diagrama de secuencia del caso de uso 'CU10 - Consultar información del jugador'

6.6.11 CU-11: Guardar resultados

En este caso de uso una vez el usuario termina la partida el sistema pregunta al manager si desea guardar la partida realizada por el jugador, este confirma la acción y se registra la partida en la base de datos.

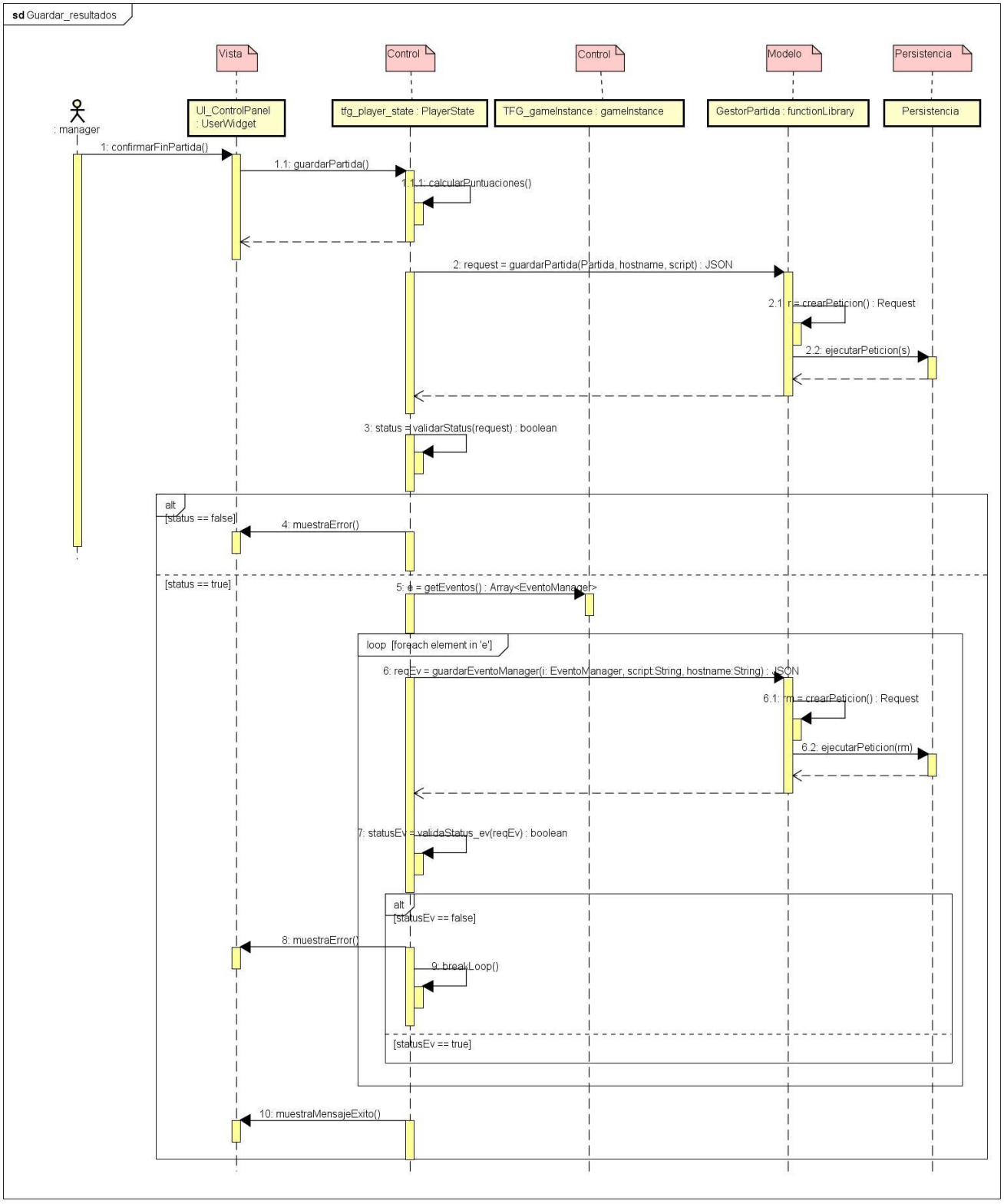


Figura 72: Diagrama de secuencia del caso de uso 'CU11 – Guardar resultados'

6.6.12 CU-12: Terminar intento

En este caso de uso el manager finaliza una sesión de juego activa antes de que el jugador haya terminado el proceso de fabricación.

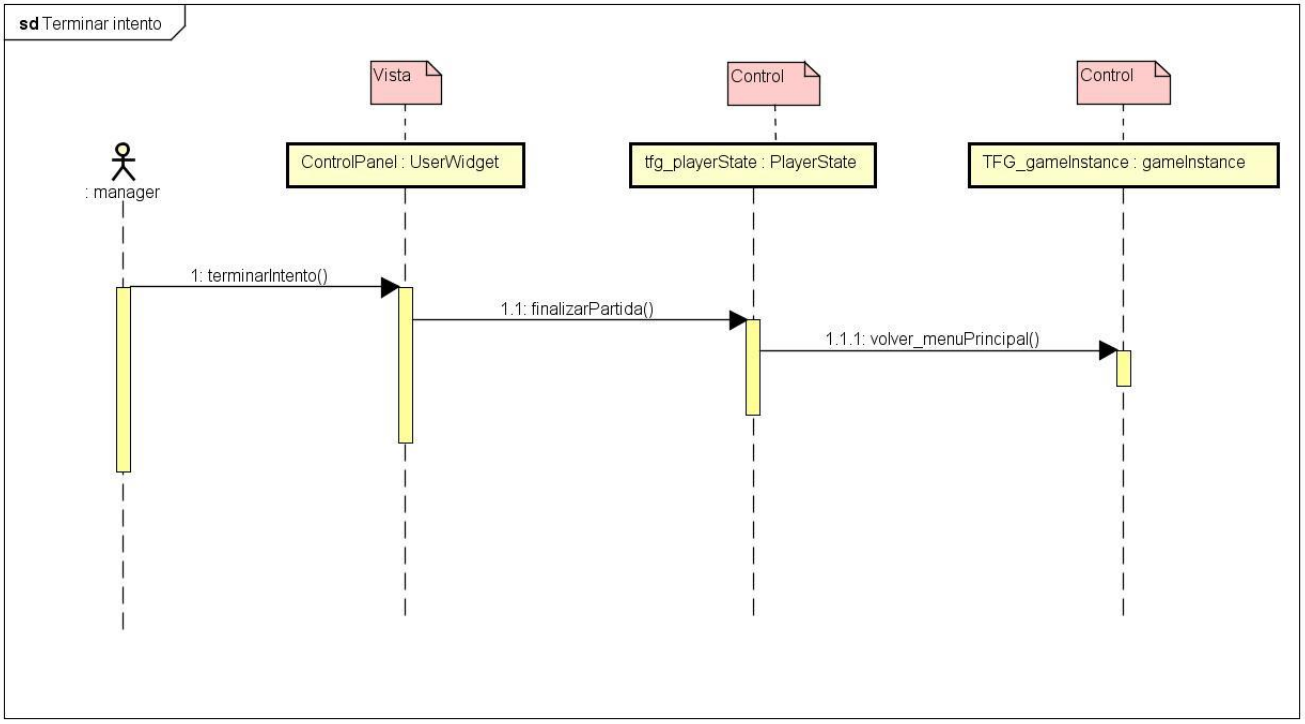


Figura 73: Diagrama de secuencia del caso de uso 'CU12 - Terminar intento'

6.6.13 CU-13: Unirse a partida

En este caso de uso un jugador se use a una sesión de juego activa.

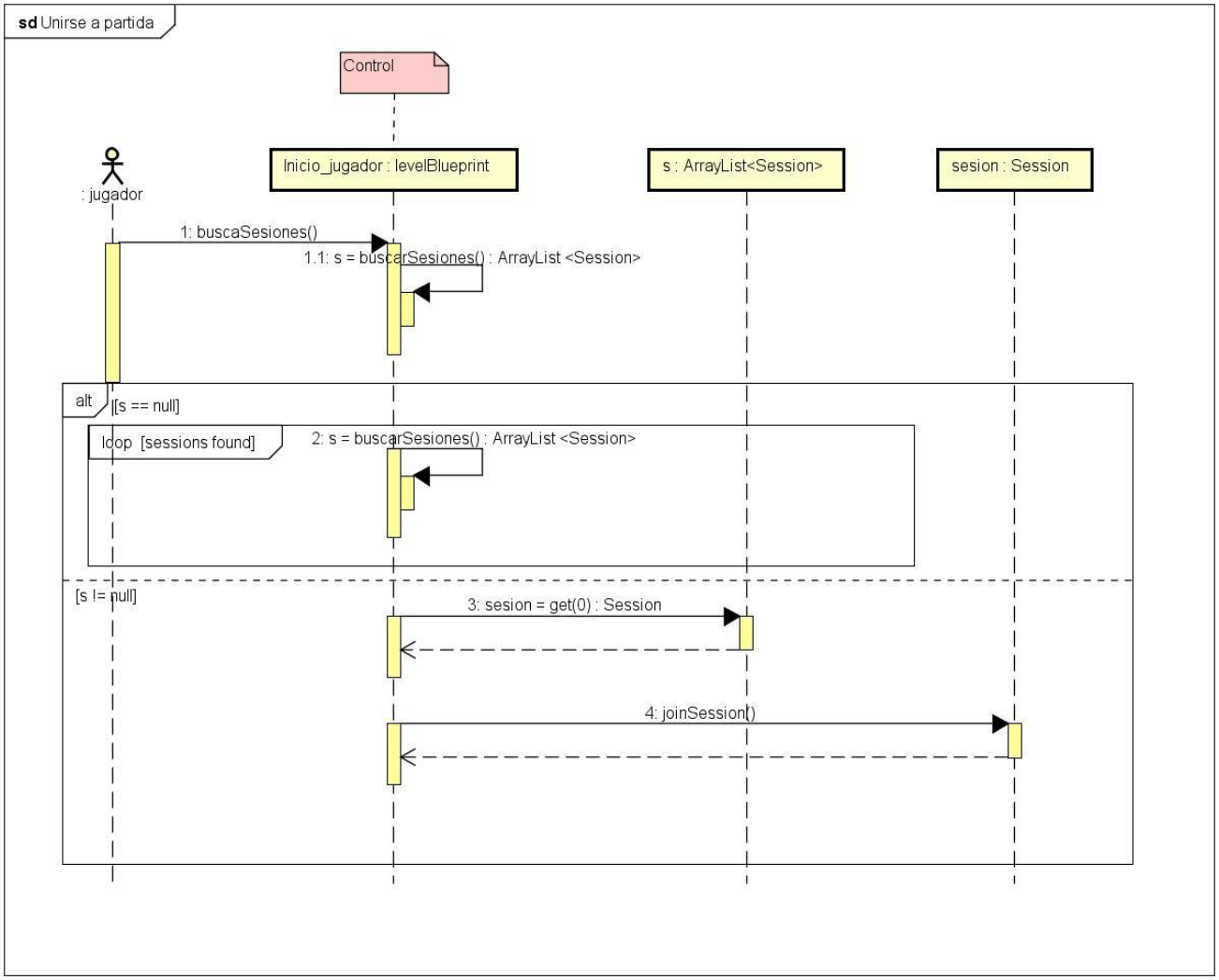


Figura 74: Diagrama de secuencia del caso de uso 'CU13 – Unirse a partida'

6.6.14 CU-14: Colocar barra metálica

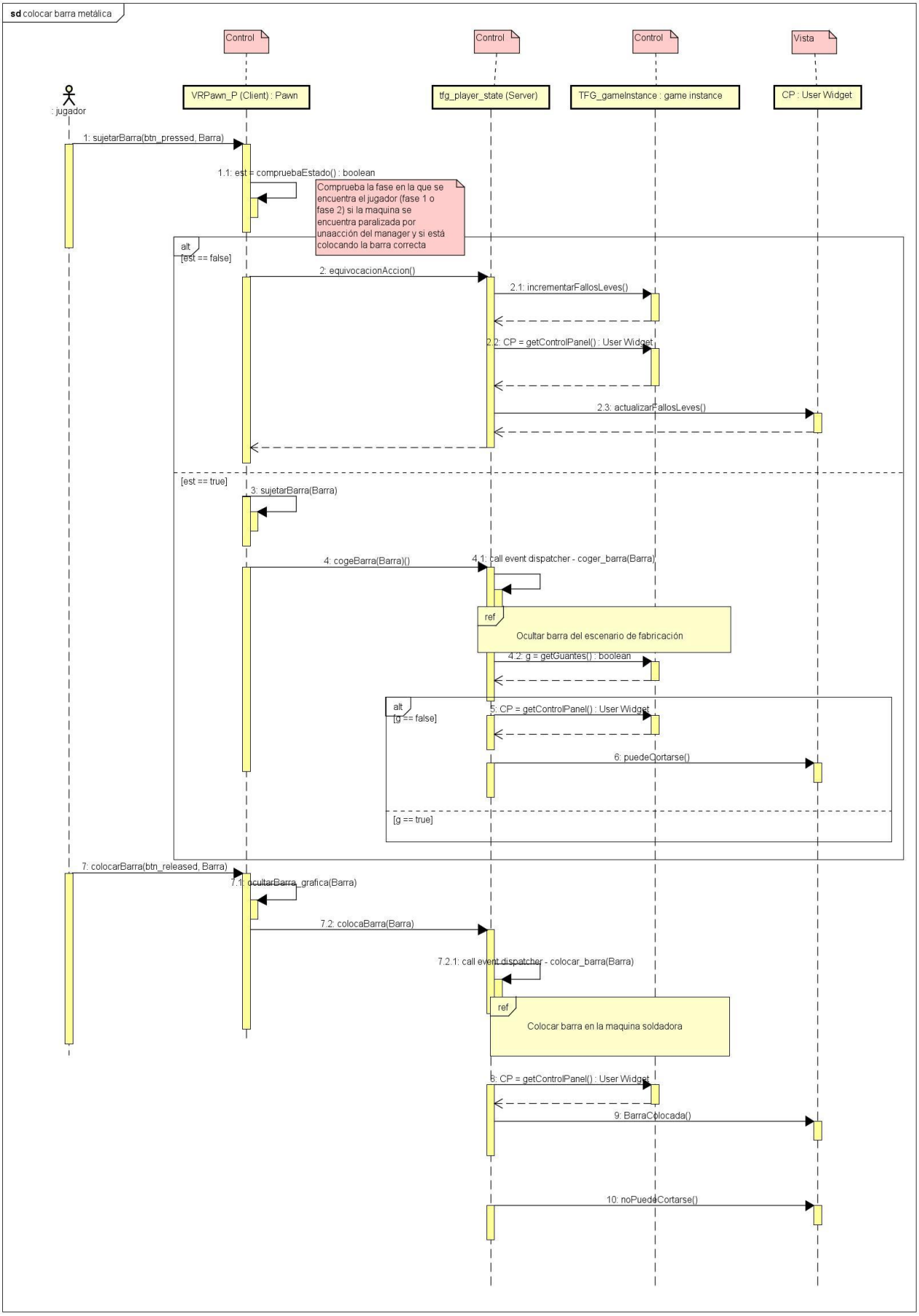


Figura 75: Diagrama de secuencia del caso de uso 'CU14 – Colocar barra metálica'

6.6.15 CU-15: Ponerse gafas de protección

En este caso de uso el usuario jugador se pone las gafas de protección del escenario de fabricación. Además, se activa un **componente *PostProcess*** que hace que cambie la vista del jugador para que simule que lleva puestas unas gafas.

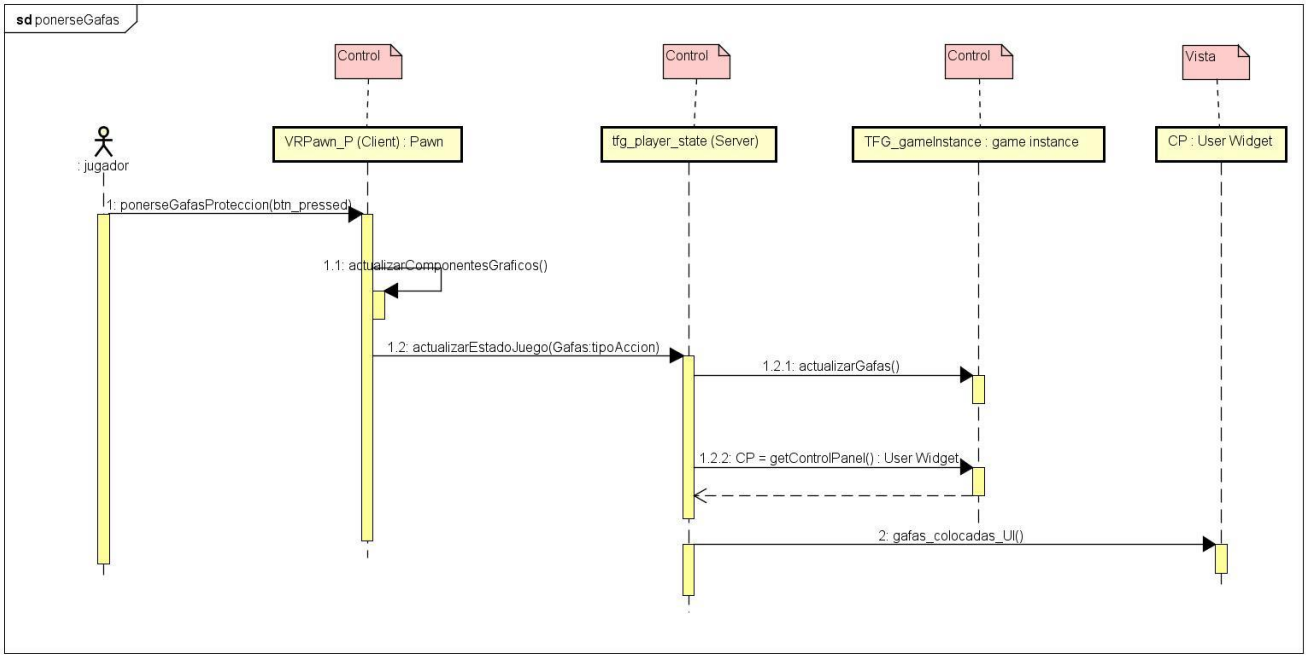


Figura 76: Diagrama del caso de uso 'CU15 - Ponerse gafas de protección'

6.6.16 CU-16: Ponerse guantes de protección

En este caso de uso el jugador se pone los guantes de protección para evitar cortes.

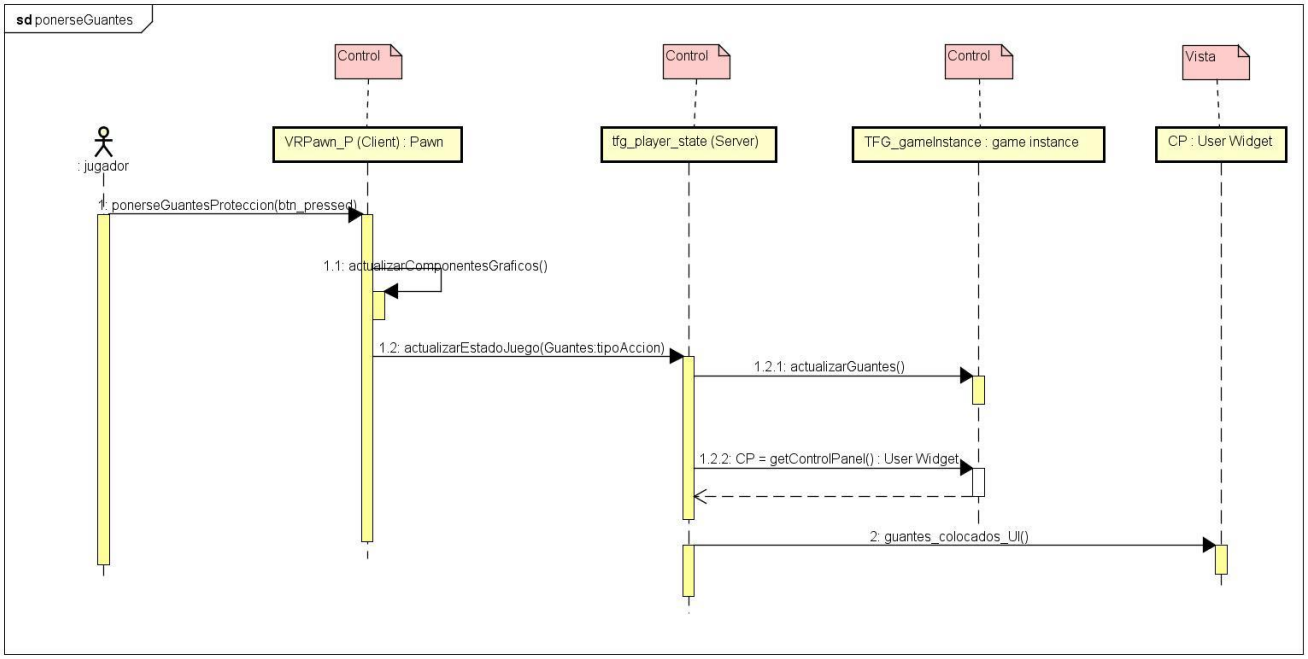


Figura 77: Diagrama de secuencia del caso de uso 'CU16 - Ponerse guantes de protección'

6.6.17 CU-17: Ponerse casco de protección

En este caso de uso el usuario jugador se pone el casco de protección en el escenario de fabricación.

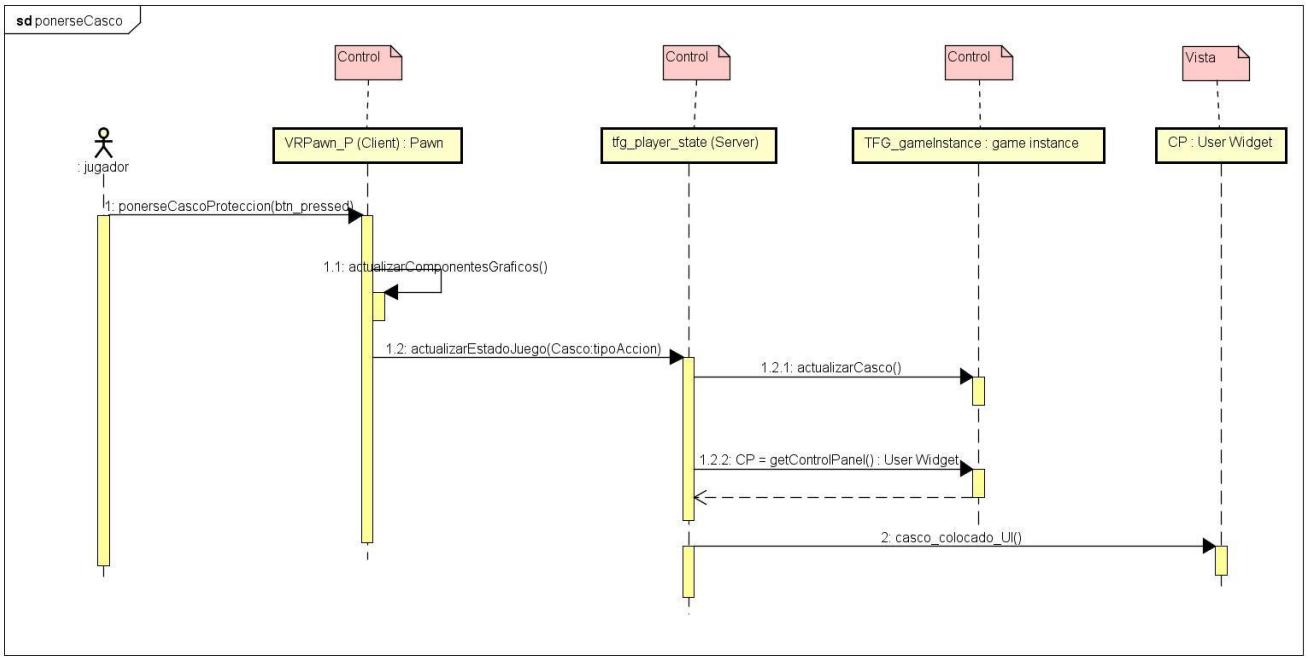


Figura 78: Diagrama de secuencia del caso de uso 'CU17 - Ponerse casco de protección'

6.6.18 CU-18: Desatascar robot soldador

En este caso de uso el usuario jugador desatasca un fallo en el robot soldador cuando un usuario manager ha activado el fallo en una partida activa.

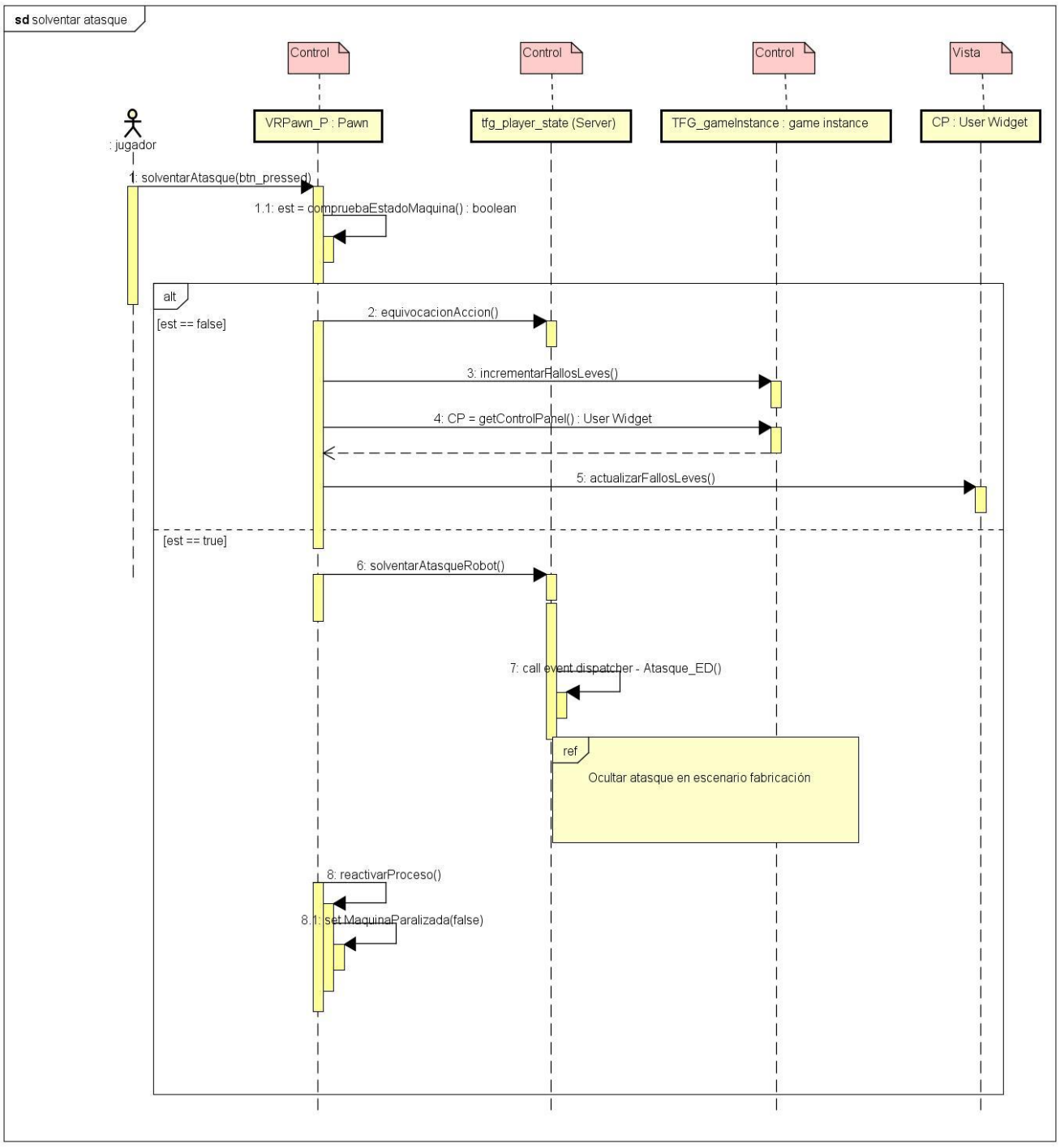


Figura 79: Diagrama de secuencia del caso de uso 'CU18 – Desatascar robot soldador'

6.6.19 CU-19: Solventar fallo de cadena

En este caso de uso un usuario jugador solventa un fallo en la cadena de producción cuando un usuario manager ha activado previamente el fallo en una sesión de juego activa.

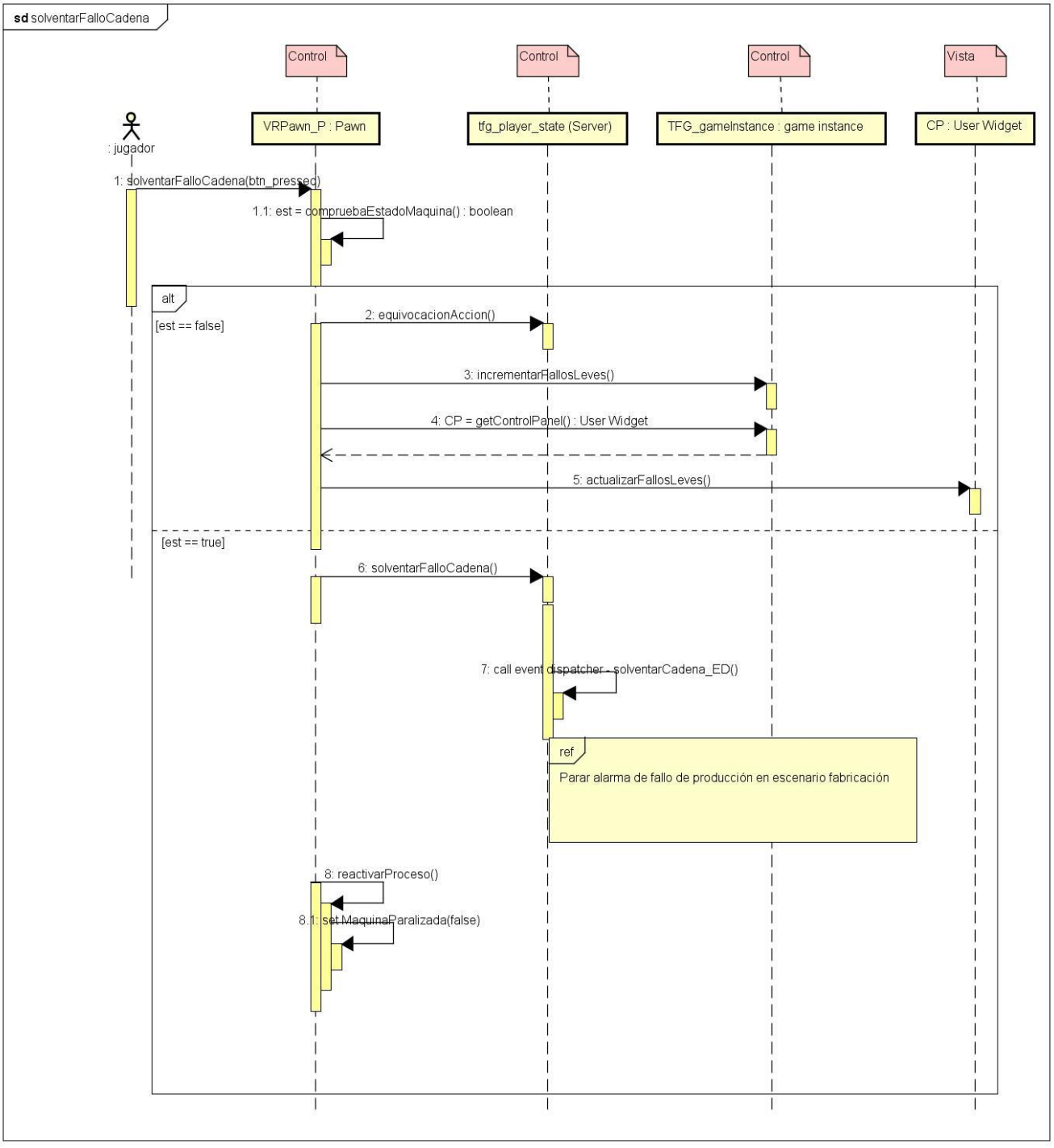


Figura 80: Diagrama de secuencia del caso de uso 'CU19 – Solventar fallo de cadena'

6.6.20 CU-20: Cambio de fase

En este caso de uso un usuario jugador puede cambiar de fase en el escenario de fabricación para continuar con el proceso de fabricación.

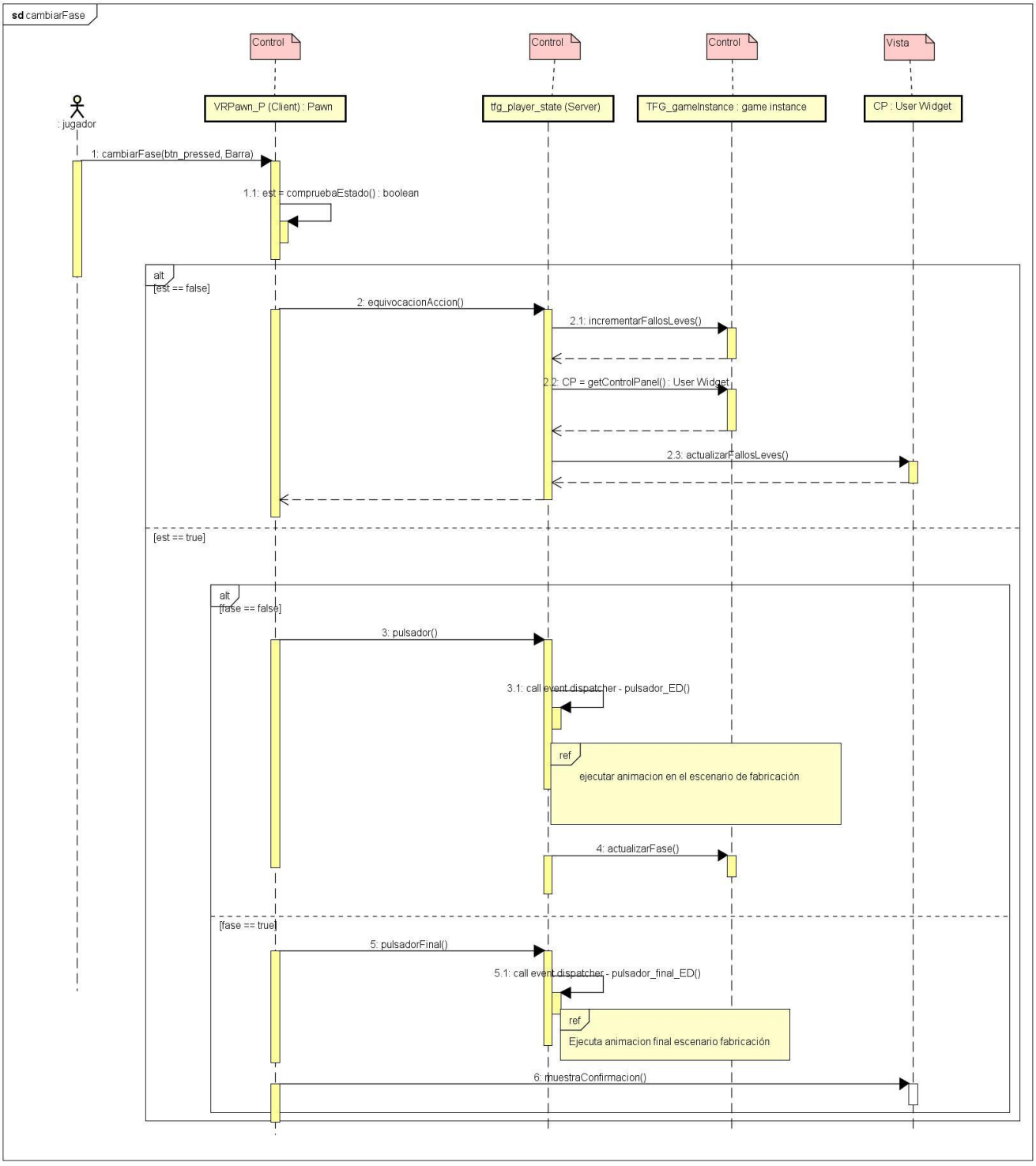


Figura 81: Diagrama de secuencia del caso de uso 'UC20 - Cambio de fase'

6.7 Diagrama de despliegue

Los nodos representados en el modelo de despliegue del sistema son los que se detallan a continuación:

- **Gafas de realidad virtual HTC VIVE** junto con 2 controladores y dos sensores lighthouse que se conectan a través de USB y un puerto HDMI al dispositivo del jugador representado como 'PC User', además de estas dos conexiones también es necesario disponer de una fuente de alimentación para este dispositivo.
- **Un PC que utilizará el jugador** y que es recomendable que disponga de la plataforma Windows 10 debido a diversos factores que se comentaban en el apartado 3.6 de esta memoria.
- **Un PC que utilizará el usuario servidor** y que es recomendable que utilice Windows 10 también como plataforma de ejecución de la aplicación. El equipo del jugador y el equipo del servidor se conectarán mediante red.
- **Un servidor Apache local al servidor y una base de datos MySQL** en la que se guardarán todos los datos relevantes de la aplicación y de cada partida jugada.

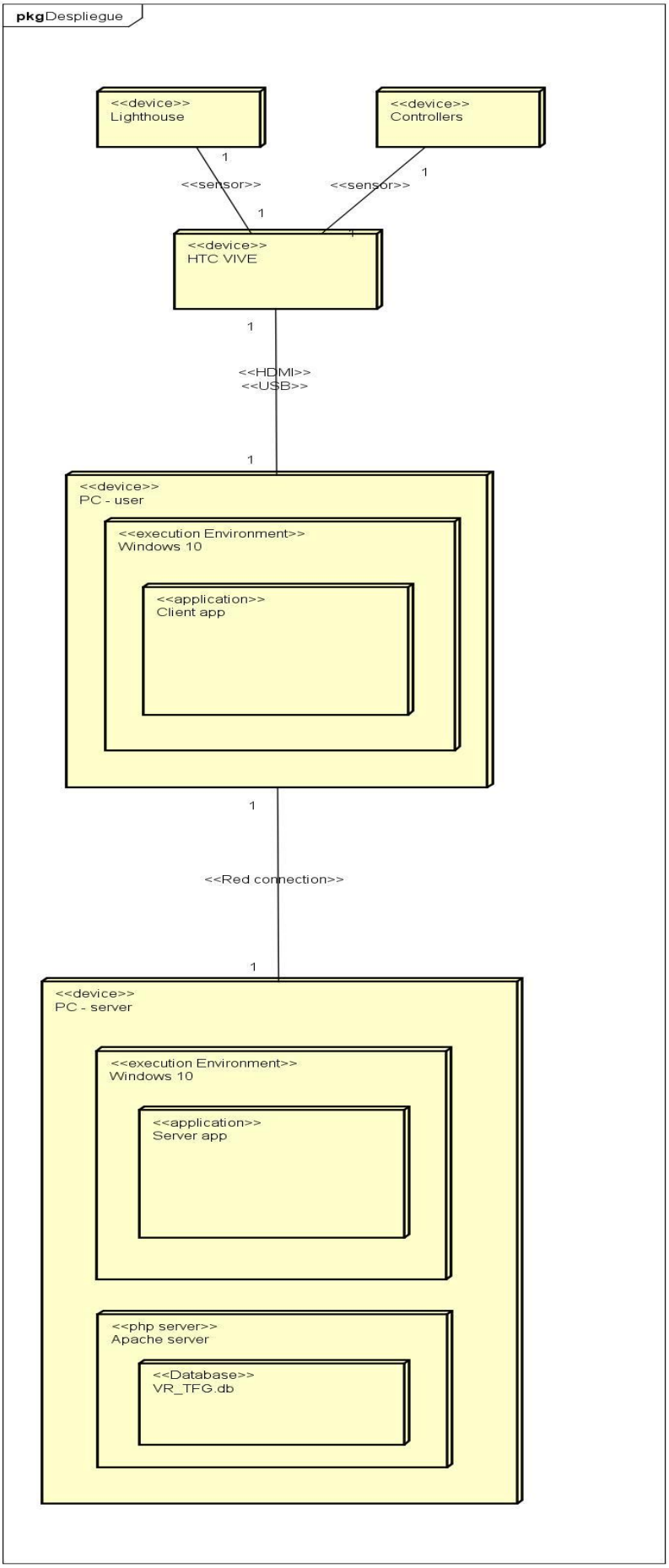


Figura 82: Diagrama de despliegue del sistema

6.8 Diseño de interfaces de usuario

En este apartado se describe el proceso de diseño de las interfaces gráficas que nos encontramos en la aplicación, explicando cada una de los componentes de las interfaces del servidor y también por qué se ha elegido el diseño de cada uno de los elementos.

6.8.1 La interfaz del servidor

A continuación, se detallan los diseños realizados para la interfaz del servidor que utilizará el usuario manager en la aplicación final y cómo se interconectan entre ellas a través de un mapa de navegación entre interfaces que servirá para explicar mejor qué función cumple cada una.

En el mapa de navegación se ha omitido la mayor parte de la navegabilidad hacia atrás, es decir, todas aquellas opciones alternativas de cancelación de la operación actual que te devuelven a una interfaz o estado anterior.

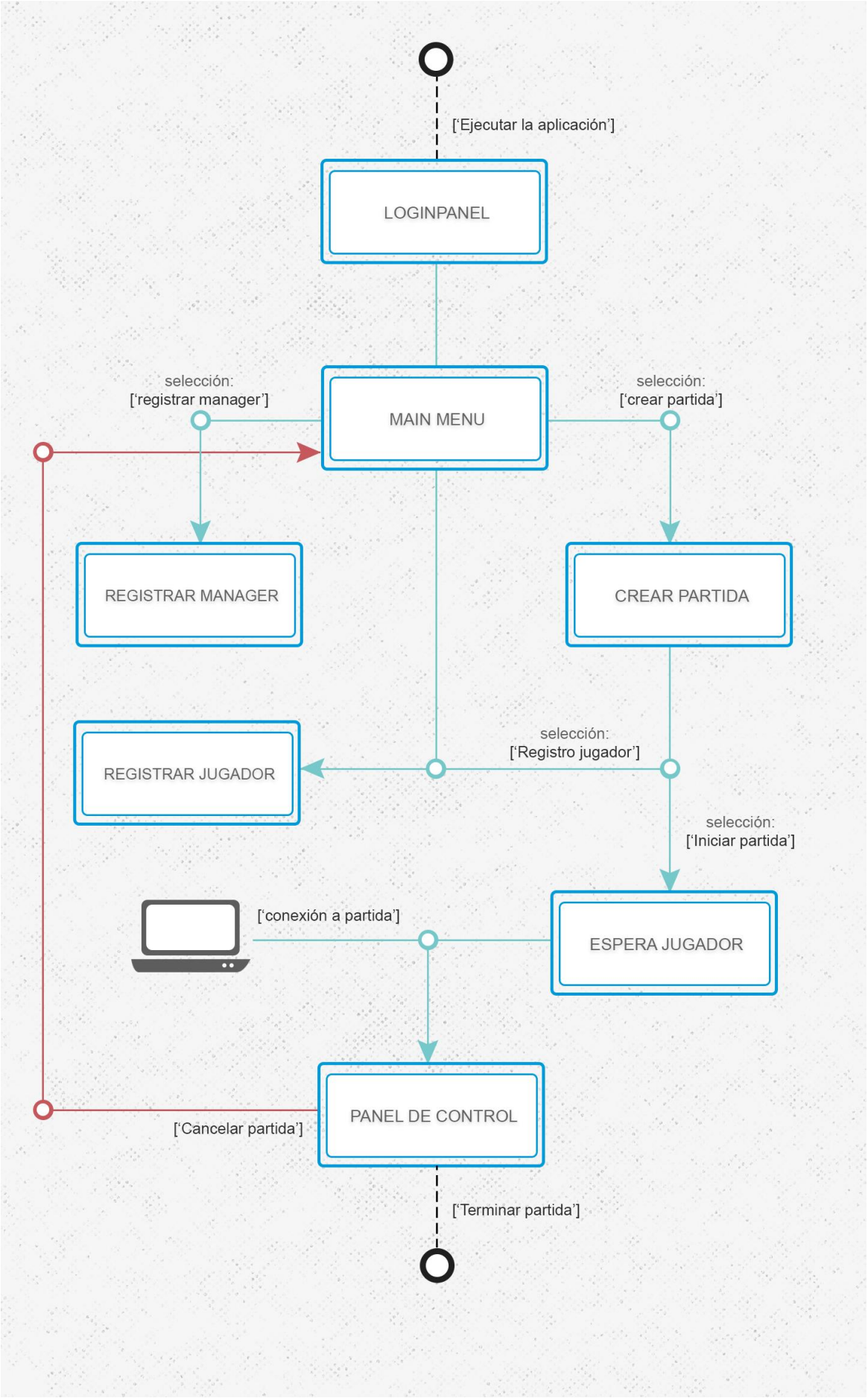


Figura 83: mapa de navegación entre las interfaces del usuario manager

6.8.1.1 Interfaz principal o login del manager

Es la primera interfaz del usuario manager al ejecutar la aplicación, como cualquier interfaz, esta se encargará de procesar las interacciones del usuario con la aplicación, se muestran dos campos que rellenar: DNI y Contraseña. Si el usuario introduce unas credenciales pertenecientes a un usuario manager en la base de datos, entonces se pasa a la interfaz 'Crear partida' detallada en el apartado siguiente 6.8.1.2.

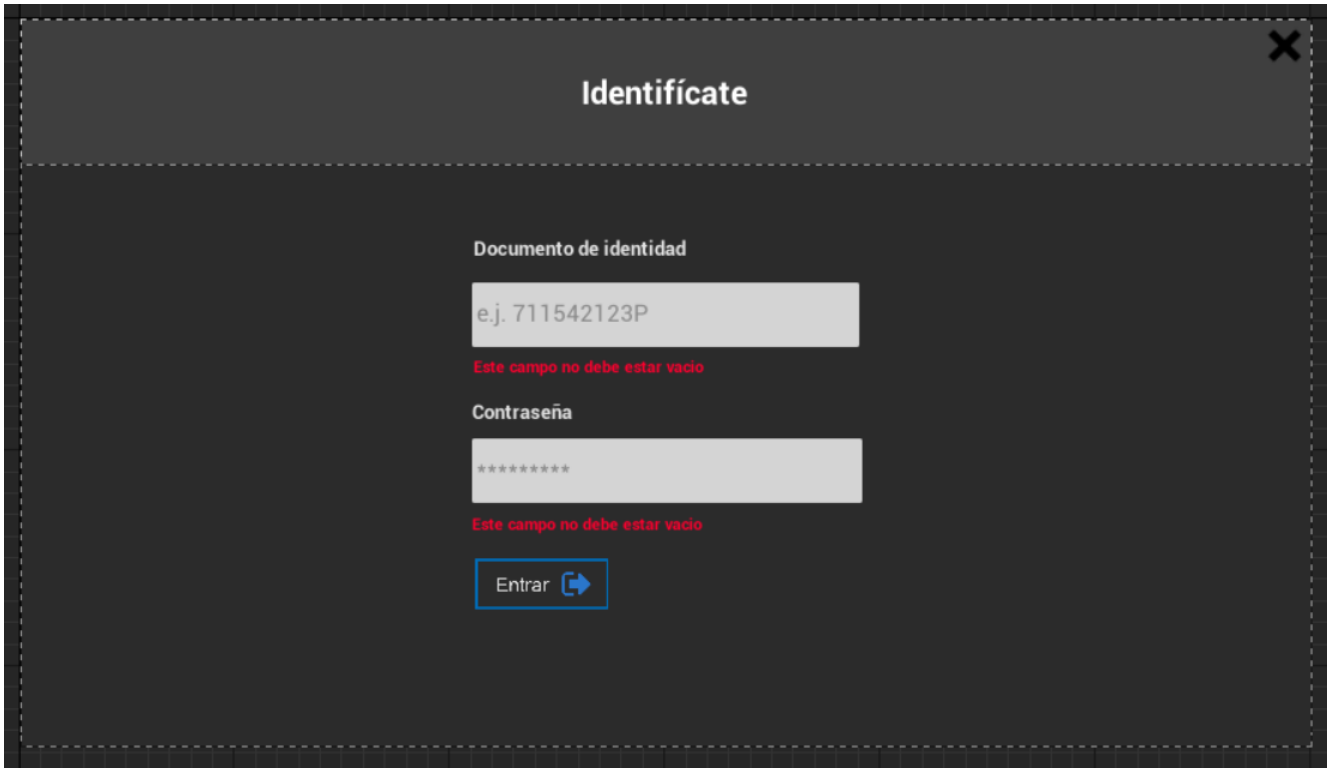


Figura 84: Interfaz de login del manager

6.8.1.2 Interfaz del menú principal

A través de esta interfaz podremos iniciar una partida, registrar un manager o un jugador, constituye la interfaz que actúa como base de operaciones para poder realizar diversas tareas.

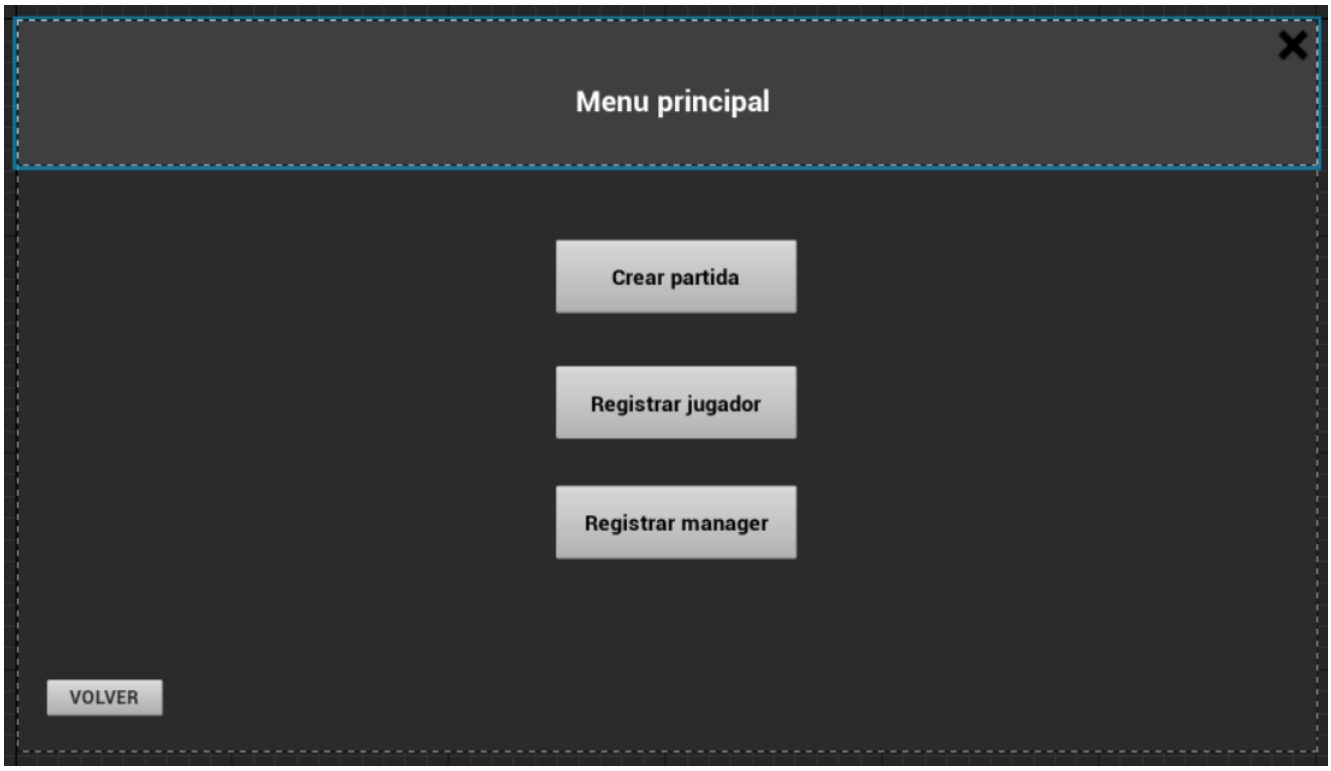


Figura 85: Interfaz de menñu principal del usuario manager

6.8.1.3 Interfaz de registro de managers

Desde esta interfaz un usuario que se haya identificado previamente en la aplicación puede registrar un nuevo usuario managers proporcionando para ello los datos:

- Nombre
- Apellidos
- DNI
- Correo electrónico válido

Si el registro en la base de datos ha ido bien, se muestra al usuario un mensaje indicándolo y se vuelve a la interfaz principal explicada previamente en el apartado 6.8.1.2.

Figura 86: Interfaz de registro de manager

6.8.1.4 Interfaz de registro de jugadores

Desde esta interfaz un usuario manager que previamente se ha identificado en el sistema puede registrar a un jugador indicando:

- Nombre
- Apellidos
- DNI
- Edad
- Experiencia previa

Si el registro se produce correctamente entonces el sistema muestra un mensaje de éxito y vuelve a la interfaz de creación de partidas.

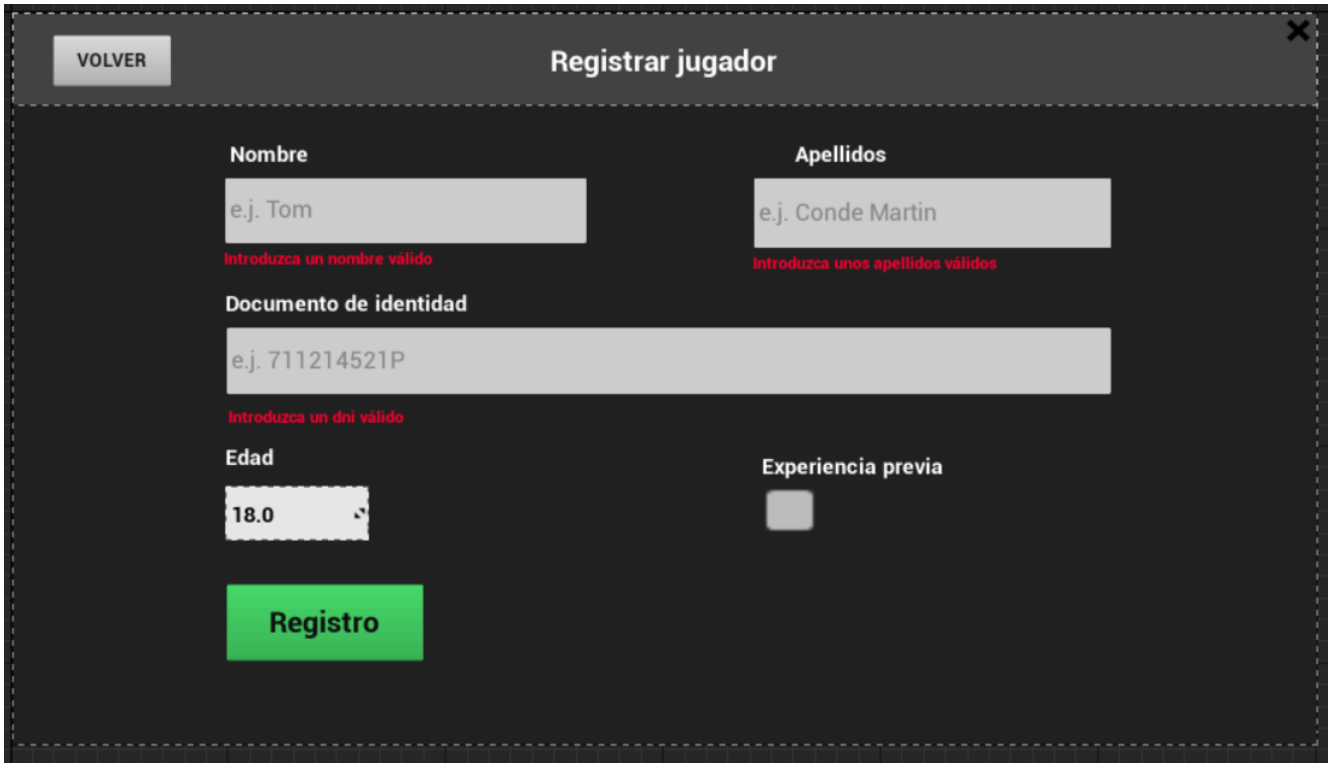


Figura 87: Interfaz de registro de jugadores

6.8.1.5 Interfaz de creación de partidas

Se accede a ella a través de la interfaz de 'main menu' y desde la que un usuario manager puede iniciar una partida multijugador, para ello debe introducir un dni de un jugador previamente registrado o también puede registrar un jugador, lo que le llevaría a la interfaz de registro de jugadores.

Una vez introducido el dni del jugador, el usuario manager selecciona un escenario (**únicamente disponible el escenario de fabricación situado a la izquierda**) y ya puede iniciar la partida.

Figura 88: Interfaz de creación de partida

6.8.1.6 Interfaz de espera

Esta interfaz se muestra cuando un usuario manager decide empezar una partida y debe esperar a que el usuario jugador ejecute a su vez la aplicación y se establezca la conexión entre cliente – servidor. Una vez esto ha ocurrido y se encuentran tanto manager como jugador 'conectados' a través de la red entonces el sistema muestra la interfaz del panel de control detallada en el siguiente apartado 6.1.1.6.

Figura 89: Interfaz de espera de jugadores

6.8.1.7 Interfaz panel de control

Se trata de la interfaz principal del usuario manager desde la que podrá observar el transcurso de la partida del jugador en realidad virtual y monitorizar datos como el tiempo de juego y en qué estado se encuentra, además de poder visualizarlo a través de un '**minimapa**' desde el que se puede cambiar la perspectiva a través de diferentes cámaras, también tiene otras opciones para poner a prueba al jugador, como que le salte una proyección de material del robot soldador o que este se atasque para que el jugador intente resolverlo.

Cuando el usuario jugador ha terminado la secuencia se calculan las puntuaciones del jugador en base al registro llevado durante la partida y se guardan los datos en una base de datos, quedando así registrado el paso del jugador por la aplicación. Si el manager decide terminar el intento y cancelar la sesión antes de que el jugador finalice su prueba entonces el sistema guardará el intento del jugador como abortado.

Al pasar el cursor por algunas de las acciones del panel de control estas se iluminan debido a que cuentan con otro diseño de imagen que se intercambia cuando ocurre este evento, lo mismo pasa cuando el usuario logra alguno de los hitos que se marcan en el juego, como ponerse las protecciones o *EPIS*.

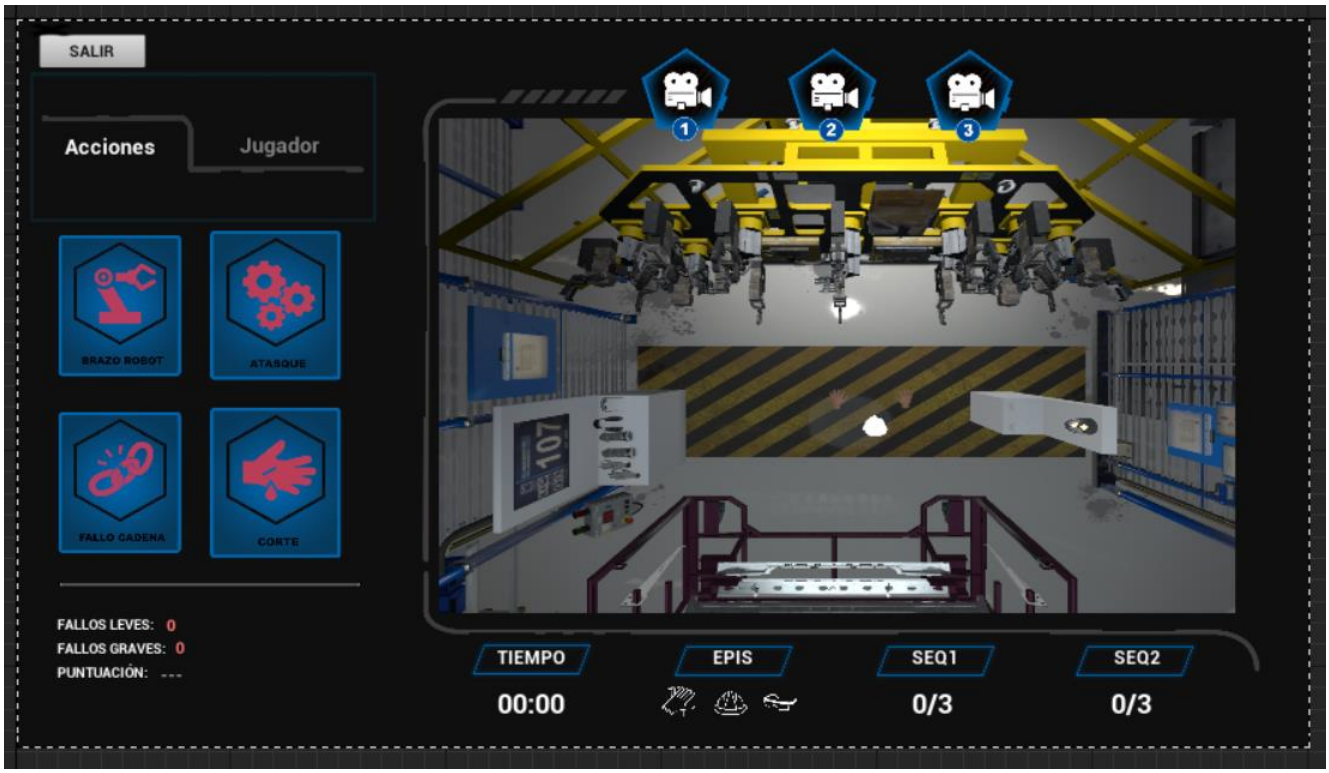


Figura 90: Interfaz del panel de control del usuario manager

6.8.2 La interfaz del jugador

La interfaz del cliente se basa en la visualización en realidad virtual del propio entorno creado en la aplicación y en el software previamente desarrollado, en el que dispone de dos controladores que harán las veces de manos dentro del juego con las que podrá interactuar con los diferentes objetos a su alcance, de modo que tendremos dos escenarios o fases:

6.8.2.1 Fase 1: Espera al servidor

El usuario jugador aparece en una sala blanca en la que se le informa de que está buscando partidas a las que conectarse, si el servidor previamente ha creado una sesión entonces se unirá a esta notificándosele previamente. El tiempo que puede pasar en esta 'sala de espera' se prolongará hasta que el servidor haya creado la sesión de juego.

6.8.2.2 Fase 2: Escenario de fabricación

Una vez cliente y servidor se encuentran en una misma sesión de juego el jugador entra en el mapa del servidor y puede comenzar a utilizar la aplicación en un escenario constituido por una máquina de soldado de piezas, unas protecciones opcionales, unas piezas metálicas y otros elementos que juntos componen la estructura de la aplicación.

7. IMPLEMENTACIÓN

7.1 Introducción

En este capítulo se detalla el proceso de desarrollo e implementación del panel de control de realidad virtual a partir de los artefactos obtenidos de las fases anteriores de planificación, análisis de los requisitos y diseño del software.

En primer lugar, se detalla el proceso llevado a cabo para conseguir una conexión vía red LAN estable entre cliente y servidor, de modo que el cliente se ejecute en un entorno de realidad virtual, mientras que el servidor tenga a su disposición un panel de control en el monitor desde el que pueda monitorizar y controlar al cliente y ambos puedan comunicarse entre sí.

Después se especifican algunas de las tareas llevadas a cabo dentro del entorno virtual relacionadas con la funcionalidad directa del jugador. Más tarde se explica el funcionamiento e implementación de las interfaces y su conexión con los elementos del entorno.

Una vez las interfaces funcionan de forma correcta se detallan los pasos seguidos para la recogida de datos de la partida y su posterior guardado. Por último, se detallan los procesos seguidos para el despliegue de la aplicación tanto para cliente como para servidor.

7.2 Fase 1: Conexión LAN entre jugador – manager

En esta sección se detalla la implementación realizada correspondiente a la configuración inicial del proyecto Unreal Engine 4 ampliando además algunos de los conceptos sobre los que trabaja el motor explicados en el apartado '**3.3 Conceptos y nociones básicas**' y, además, la implementación y funcionamiento del **modelo cliente - servidor** dentro del proyecto [45].

7.2.1 Configuración del proyecto

Se ha tomado como proyecto inicial el software previamente desarrollado en el que un usuario puede experimentar las actividades de un entorno de trabajo en dos posibles escenarios, pudiendo escoger este entre un escenario de fabricación de carrocería de vehículos o de montaje final de piezas de carrocería en realidad virtual.

Para poder desarrollar correctamente una aplicación multijugador basada en técnicas de realidad virtual, se han llevado a cabo las siguientes implementaciones y configuraciones:

- **Instalación y configuración de los dispositivos de realidad virtual.** Se han instalado y configurado la suite de programas necesarios para poder trabajar en el proyecto:
  - **VIVE Desktop App**. Imprescindible para hacer instalar y hacer funcionar unas gafas HTC VIVE en un dispositivo compatible.
  - **Steam + Steam VR**. Sin este programa no podremos visualizar correctamente el contenido creado desde el motor.
  - **Lanzador Epic Games + UE4 4.13.2**. Se ha instalado tanto el lanzador de Epic Games disponible desde [17] y el motor con la versión 4.13.2 directamente desde el lanzador.
  - **Visual Studio 2015**. Se ha instalado Visual Studio versión 2015 seleccionando además las opciones de 'Common Tools' a la hora de la instalación.

- **Configuración del subsistema online**. Por defecto, Unreal Engine 4 no permite realizar algunas de las funcionalidades requeridas durante el desarrollo de una aplicación

multijugador, y por ello se ha actualizado el archivo del proyecto 'DefaultEngine.ini' que se encuentra en la ruta 'IVECO/Config/DefaultEngine.ini' añadiendo las siguientes líneas para evitar problemas en las comunicaciones multijugador:

```
[OnlineSubsystem]
DefaultPlatformService=Null
```

Figura 91: Código necesario para las pruebas multijugador en local

- **Preparación del sistema de niveles**. Se definen los niveles y escenarios que existirán en la aplicación y se eliminan todos aquellos archivos y contenido sobrante del software previo que no se requiere dentro de la aplicación desarrollada, revisando para ello todo el código y comprendiendo el funcionamiento de este para optimizar el contenido del proyecto sin eliminar ni modificar nada especialmente relevante para el funcionamiento de la aplicación. Se ha optado por la creación de 3 niveles:

    - **Nivel de Inicio del manager**. Cuando un usuario manager inicia la aplicación se sitúa sobre un entorno sin contenido 3D sobre el que se situarán las interfaces del usuario en 2D, de esta forma si tuviéramos que navegar hacia múltiples escenarios podemos pasar desde este nivel al seleccionado mediante las interfaces.
    - **Nivel de Inicio del jugador**. Este nivel si tiene contenido 3D y muestra un entorno sencillo desde el que el jugador pueda observar si se están buscando sesiones activas y que cuando se han encontrado le informe de que se está uniendo a la sesión o del estado del juego en general.
    - **Nivel del escenario de fabricación.** Este nivel se ha tomado del software previo como punto de partida y muestra un entorno que simula un puesto de trabajo de fabricación de piezas de carrocería, se han modificado y eliminado todas aquellas funcionalidades que no son de utilidad para la aplicación que se ha desarrollado y que se describe en este trabajo. Mientras en el software anterior esta aplicación era completamente guiada dejándole poco margen de actuación libre al usuario, la aplicación objeto de esta memoria se centra en no guiar al jugador y observar y recoger sus aciertos y fallos a través de un panel de control.

- **Configuración del modo de juego y clases asociadas.** Se crean y configuran las clases de control del proyecto, teniendo:
    - **GameState**: gameState_TFG
    - **Pawn**: VRPawn_P
    - **PlayerController**: VRPlayerController_P
    - **Game Instance Class**: TFG_GameInstance

- **Modificación del software previo**. Se modifica gran parte de la funcionalidad de la aplicación para permitir que un usuario jugador pueda moverse libremente por el escenario y, además, que pueda convertirse en una aplicación cliente-servidor.

7.2.2 Conceptos adicionales

En este apartado ampliamos algunas de las nociones básicas sobre algunos componentes del motor relevantes para explicar la implementación de modo que pueda comprenderse de mejor manera el código desarrollado en el proyecto:

7.2.3 Creación de la sesión

A continuación, se describe la implementación realizada para construir la funcionalidad de creación de sesión en la parte de la aplicación que utilizará el usuario manager y que cumple el rol de servidor.

La funcionalidad que permite la **creación de la sesión** se encuentra en el objeto ***tfg_game_instance*** que hereda de la clase del motor *UGameInstance* y a la que podemos acceder desde cualquier otro actor o componente del juego cuando la aplicación se está ejecutando.

Cuando el usuario manager selecciona crear una partida, la vista se comunica con el controlador *TFG_player_state* y este a su vez con la instancia *TFG_gameInstance* que crea una sesión de juego en la red. Si esta se crea con éxito entonces se inicia el nivel *EscenarioFabricación* que se comunica con el controlador *TFG_player_state* para que inicie la espera de jugadores.

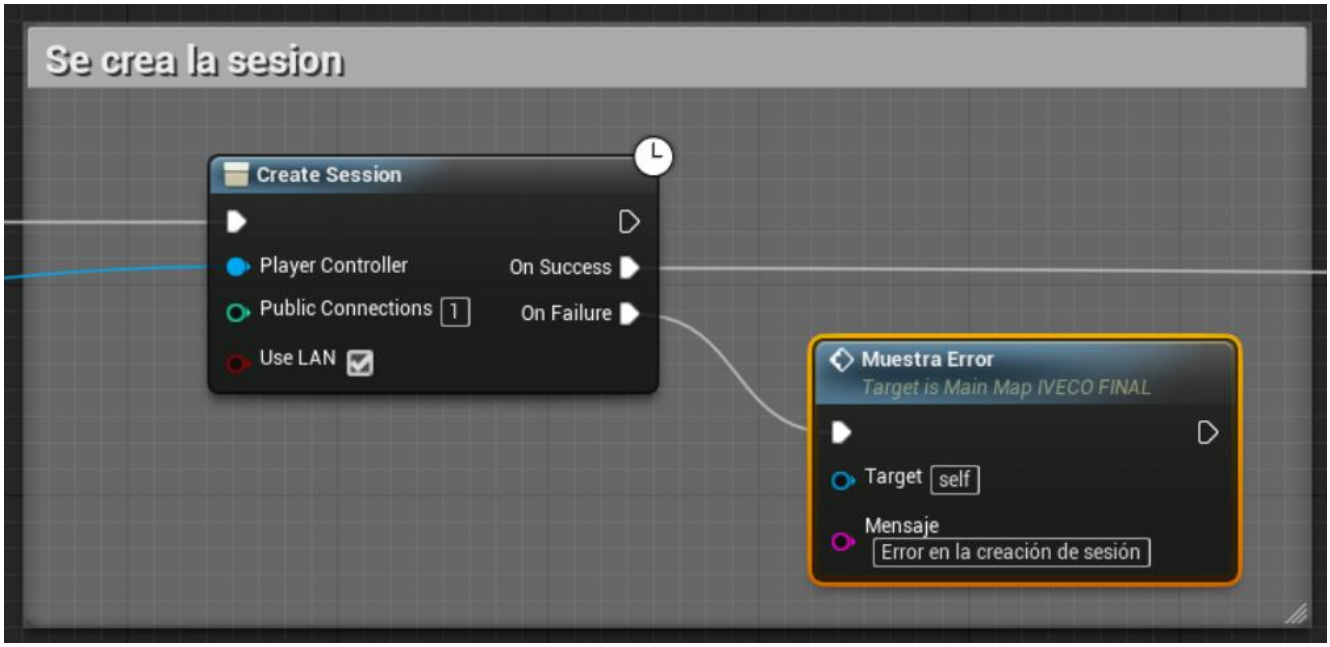


Figura 92: Nodo de creación de sesión

Una vez creada la sesión y abierto el escenario de juego, se espera hasta que se una a la sesión el jugador a través de la red, para obtener el **número de jugadores activos** se accede al blueprint ***GameMode*** de la aplicación que es la clase encargada de guardar esta información y que tan sólo es accesible desde el servidor en aplicaciones multijugador. **Mientras el número de jugadores no sea igual a dos o mayor** (contando al servidor) no se inicia la partida.

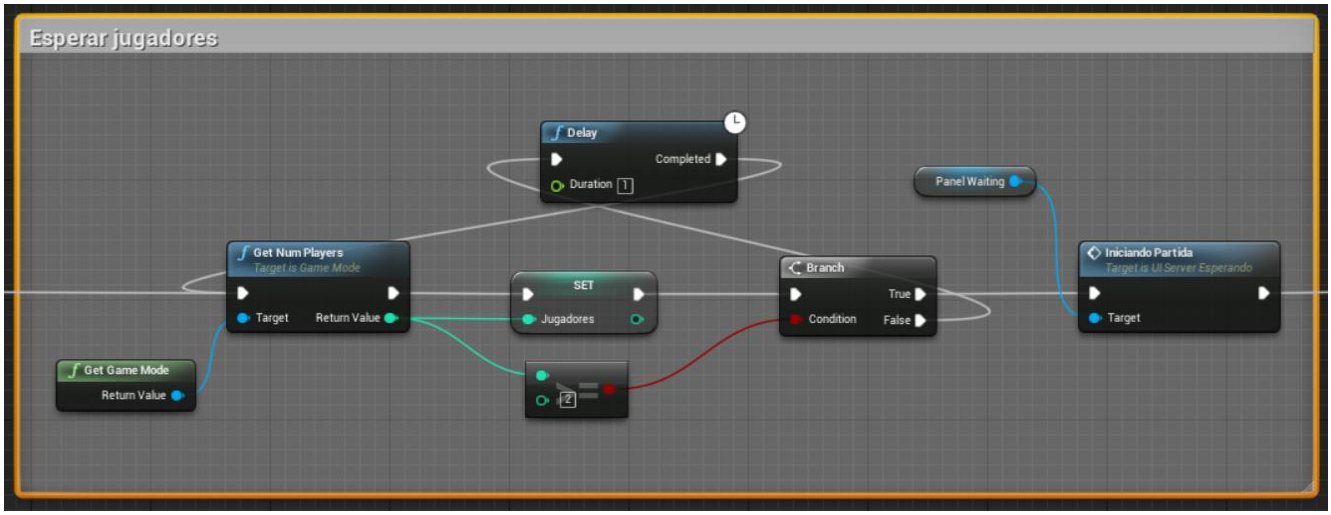


Figura 93: Código de espera de jugadores

Cuando el jugador se une a la sesión de juego, aumenta el número de jugadores en el *Game Mode* y es entonces cuando se elimina el widget actual de la vista y **se crea el panel de control** del usuario manager en el controlador ***TFG_gameInstance*** desde el que podrá gestionar y monitorizar la partida del jugador.

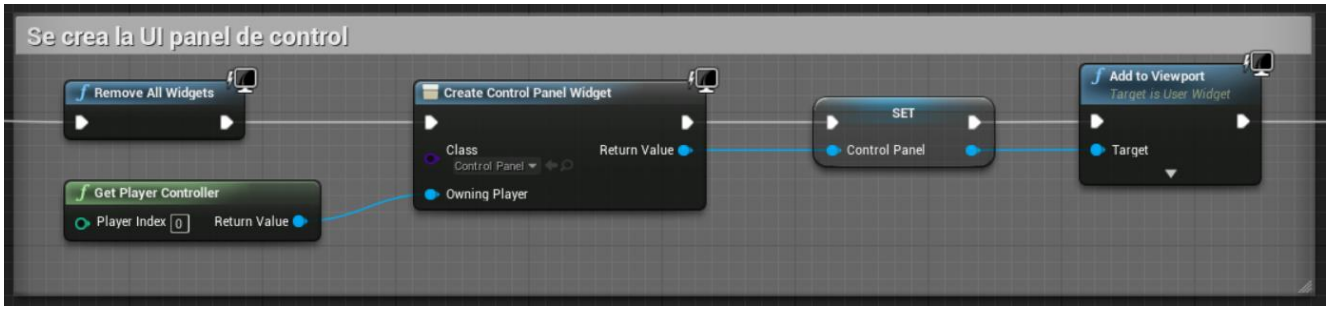


Figura 94: Creación del widget Control Panel

7.2.4 Unirse a una sesión

Un usuario jugador, al iniciar la aplicación, inicia por defecto el nivel ‘Inicio_jugador’ que automáticamente buscará sesiones en la red en la que esté conectado el dispositivo desde el que se ejecuta la aplicación. Por tanto, es el level blueprint del nivel ‘Inicio_jugador’ quein se encarga de buscar sesiones.

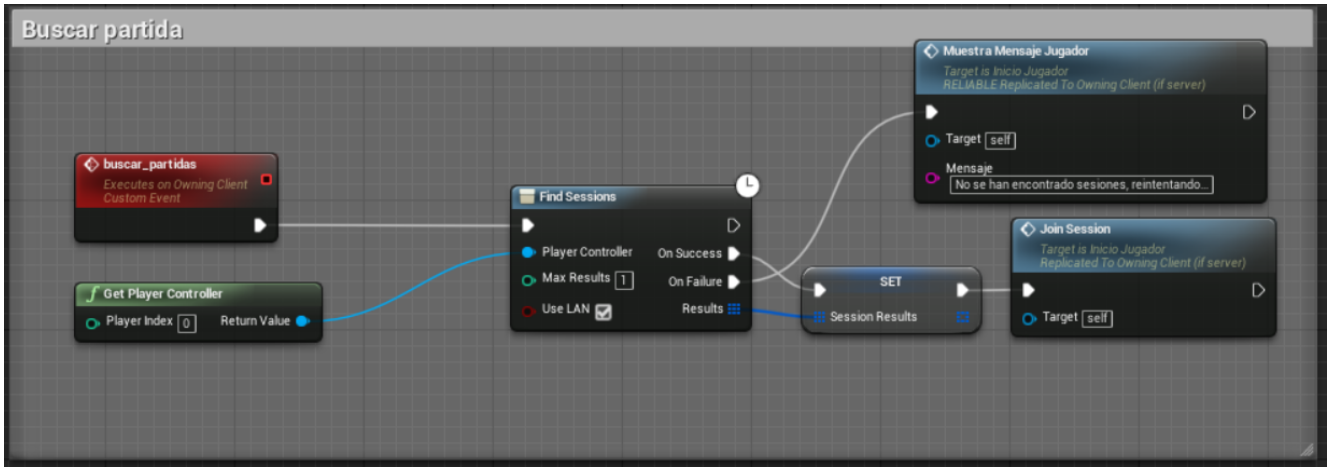


Figura 95: Búsqueda de sesiones en el blueprint del nivel 'Inicio_jugador'

Una vez se han encontrado sesiones de juego, entonces se actualiza el texto del nivel del jugador mostrándole que se han encontrado partidas y se une a la primera sesión almacenada en el array de sesiones mediante el nodo *Join Session* que hace que se cargue en el cliente el mismo nivel del servidor al que se está uniendo en la sesión.

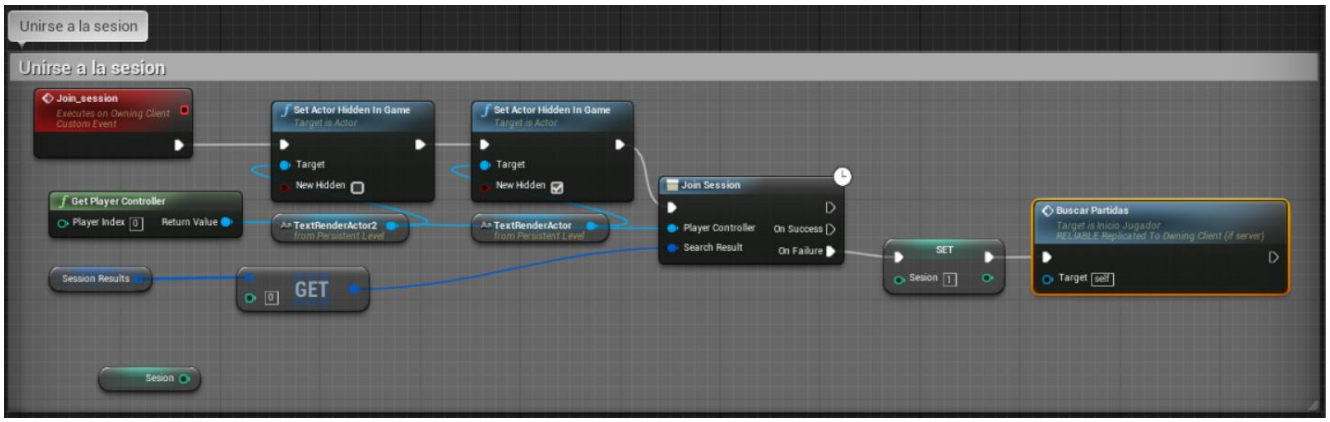


Figura 96: Unirse a una sesión en el blueprint del nivel 'Inicio_jugador'

7.3 Fase 2: Interfaces funcionales

En este apartado se describen las interfaces del usuario manager y se detalla la implementación desarrollada para estas.

7.3.1 Interfaz Login Panel

Es la interfaz principal que se crea cuando un usuario manager ejecuta la aplicación, básicamente consiste en una interfaz que permite el inicio de sesión de un usuario manager introduciendo un número de **documento de identidad** (DNI) y una **contraseña** perteneciente a un usuario administrador o a un usuario manager que haya sido registrado en la base de datos con anterioridad.

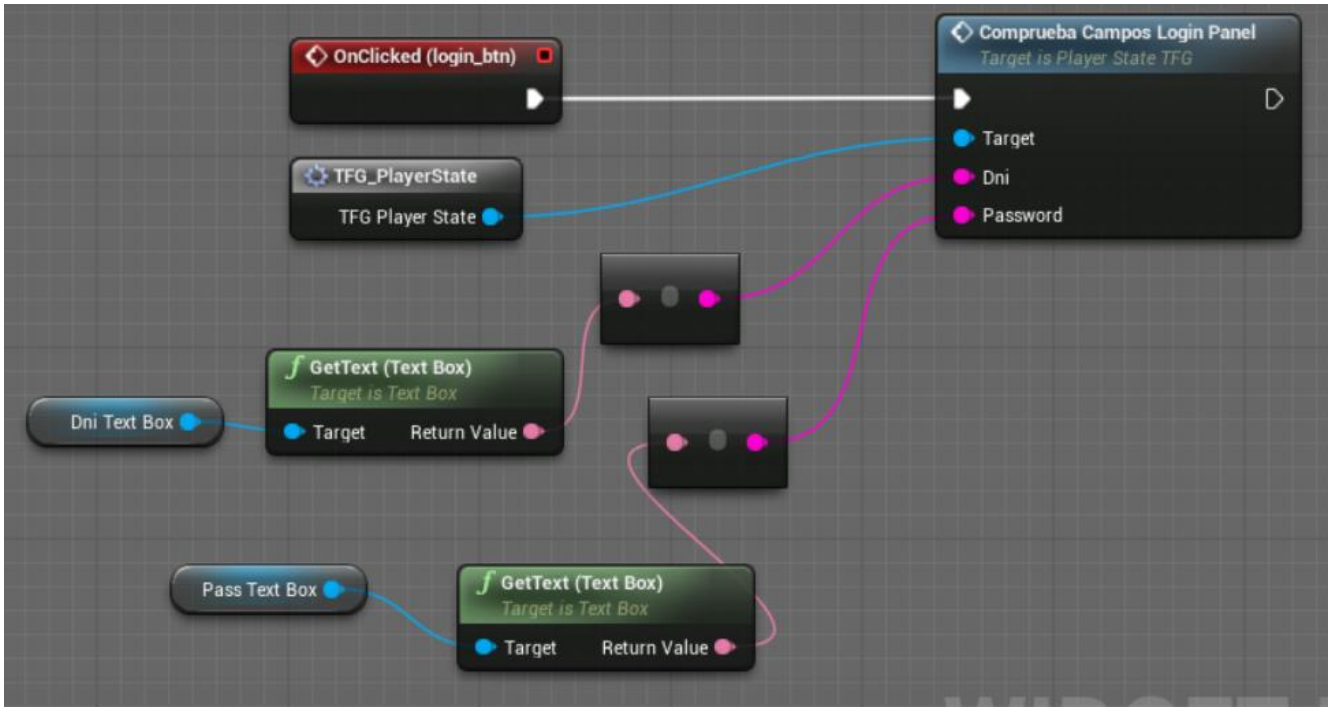


Figura 97: Recogida de datos del formulario en la interfaz de login y llamada al controlador TFG_player_state

Cuando el usuario introduce los datos correspondientes a los campos de DNI y contraseña, la interfaz se comunica con el controlador *TFG_player_state* y comprueba que los datos introducidos son válidos.

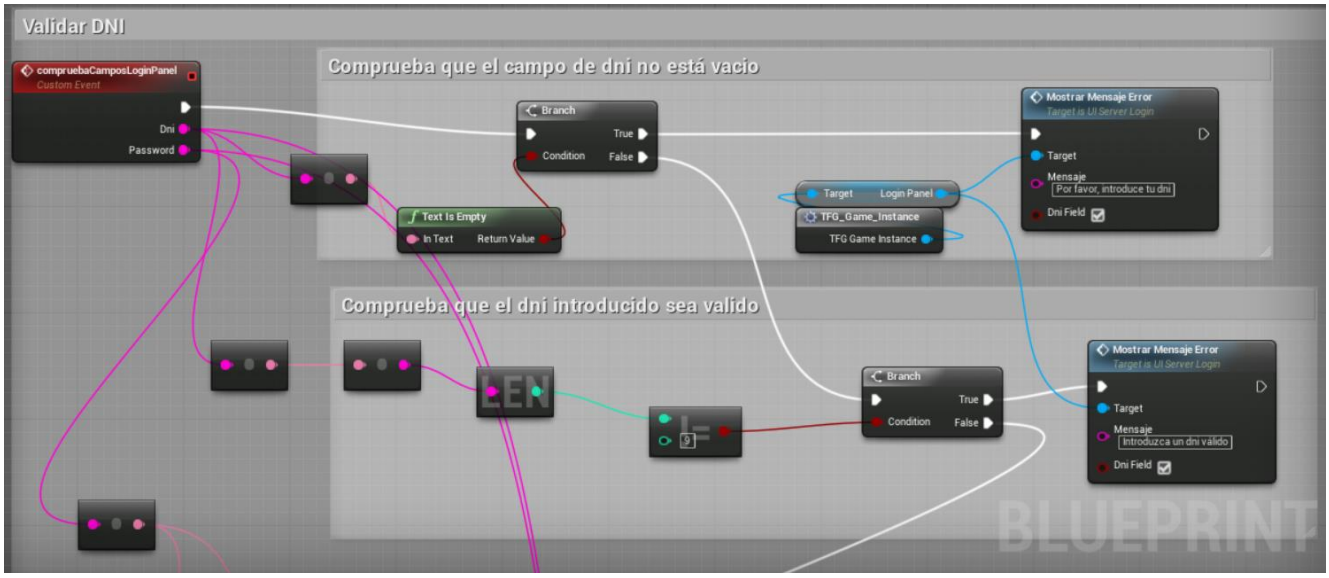


Figura 98: Fragmento del evento 'compruebaCamposLoginPanel' del controlador TFG_player_state

Si los datos introducidos son válidos, entonces se comunica con la **clase del modelo *gestorManager*** que construye una petición a la base de datos y se comunica con un script PHP que accede a la base de datos local y que devuelve el status de la transacción en un objeto JSON.

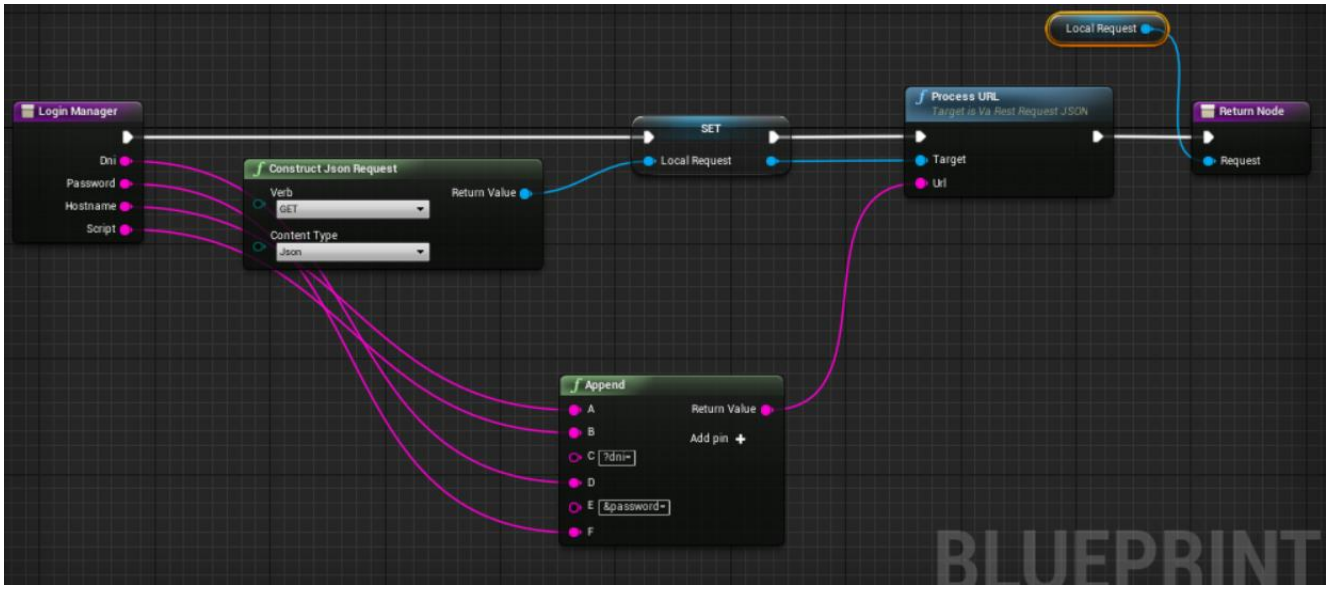


Figura 99: Función de identificación de un usuario manager en el gestor de managers

Una vez ejecutada la solicitud, el controlador *TFG_player_state* comprueba si el status de la transacción contenido en el objeto JSON ha sido satisfactorio, si es así, entonces el controlador informa a la vista de que el usuario se ha identificado ante el sistema y comunica al controlador *TFG_gameInstance* que cree la interfaz de '**MainMenu'** y además crea una **variable de tipo 'Manager'** que guardará los datos del manager identificado durante el resto de la partida.

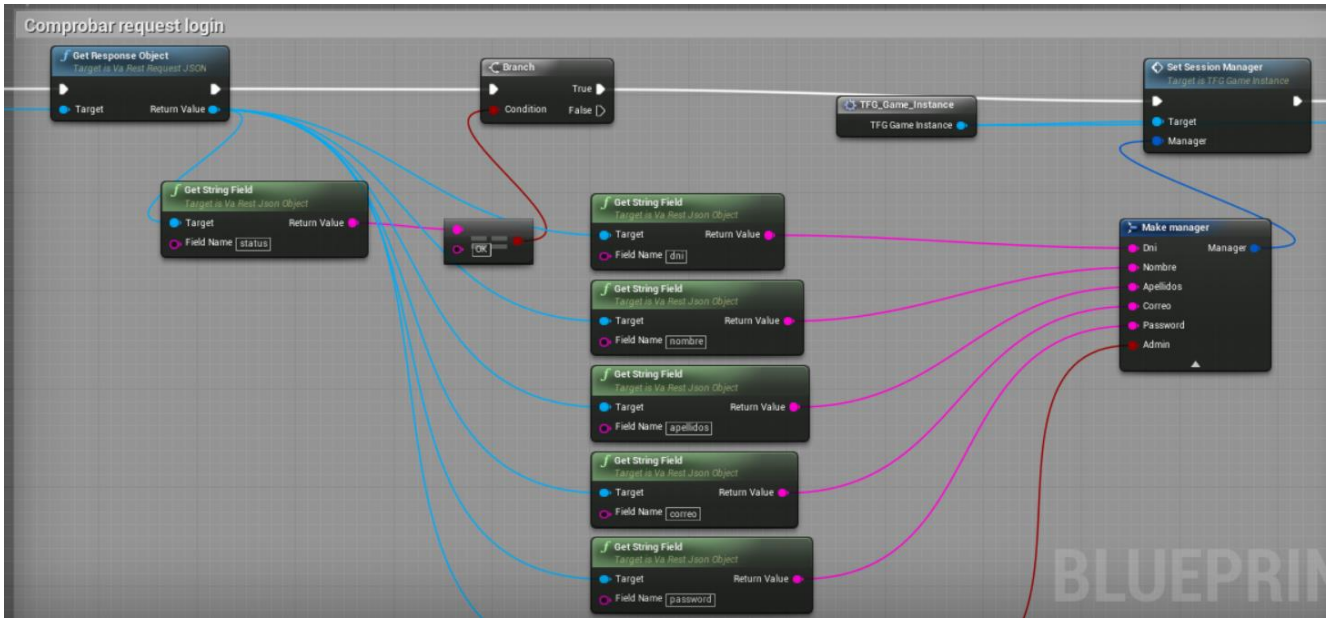


Figura 100: Validación del objeto request devuelto desde el gestor de managers

Si los datos son erróneos, el controlador comunica a la interfaz que debe mostrar un **mensaje de error** informando de los fallos o errores al usuario. También se ha implementado un botón que permite salir de la aplicación cuando se pulsa.

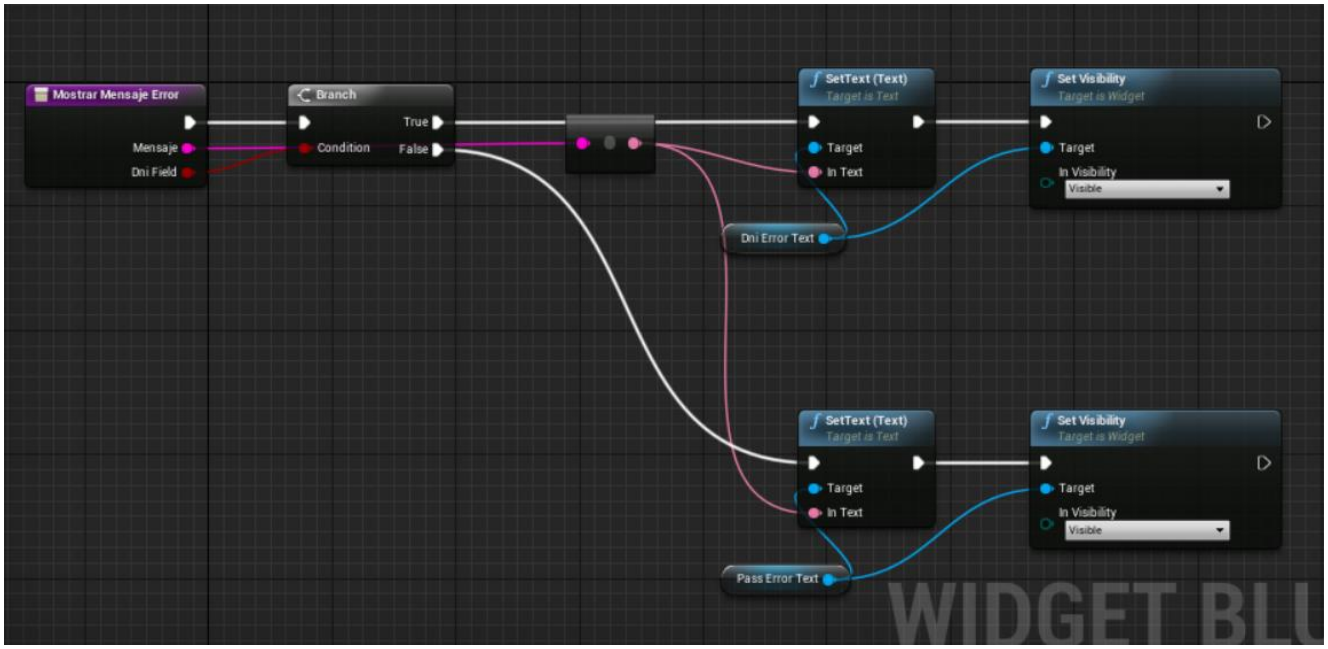


Figura 101: Función para mostrar mensajes de error en el formulario de la interfaz de login

7.3.2 Interfaz Main Menu

Esta interfaz funciona como un elemento de transición hacia otras 3 interfaces con funcionalidades, que son:

- Crear partida
- Registrar jugador
- Registrar manager

Cada una de estas es accesible mediante un componente *UButton* que lanza un evento cuando se hace click sobre este, de modo que cuando esto ocurre el evento llama al controlador *TFG_player_state* que se comunica con el *TFG_gameInstance* y actualiza la vista correspondiente del jugador según el evento lanzado, por ejemplo, si el usuario hace click sobre ***Crear partida*** entonces se lanzará un evento que hará que el controlador destruya la vista actual y muestre la interfaz gráfica de creación de partidas.

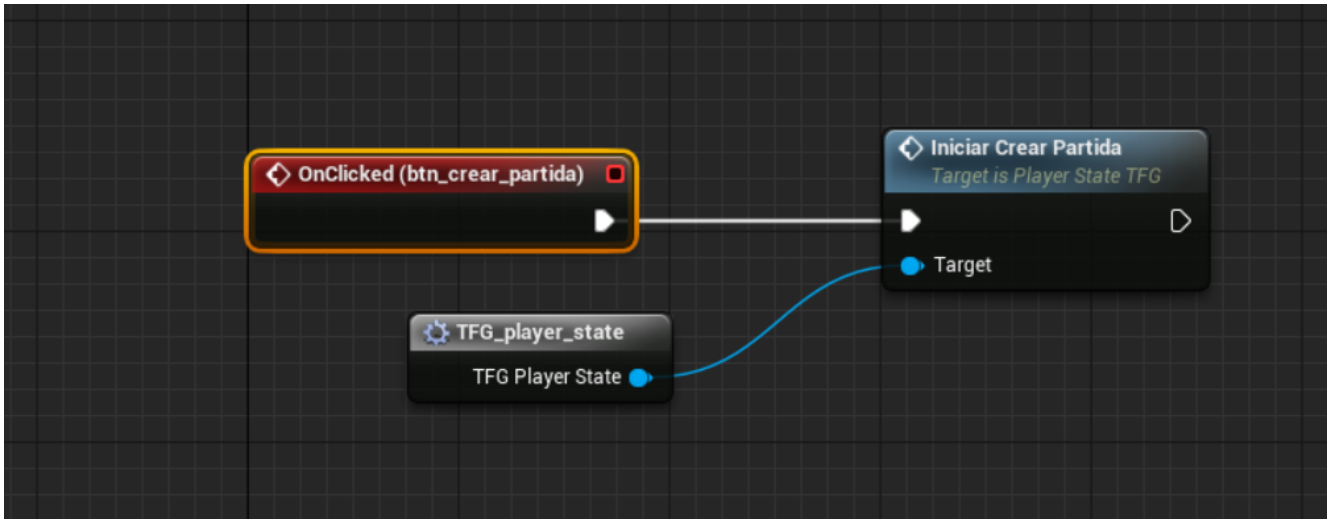


Figura 102: Evento lanzado cuando se hace click sobre el botón de 'btn_crear_partida' de la interfaz de Crear partida

7.3.3 Interfaz de Registrar Manager

Accesible desde la interfaz *MainMenu*, es la encargada de registrar a un usuario manager que no haya sido registrado previamente en la base de datos. Únicamente es accesible por usuarios manager con permisos de administración, en caso contrario la opción aparecerá como 'No seleccionable'. Esta interfaz requiere la introducción de los siguientes datos:

- **Nombre**. No puede contener un número de caracteres superior a 15 ni inferior a 3. Sólo se permiten letras y espacios.
- **Apellidos**. El máximo número de caracteres permitidos es 25 y un mínimo de 3. Sólo se permiten letras y espacios.
- **Correo**. Se comprueba que sea un correo válido, es decir, que contenga al menos un carácter '@'. Se permiten entre 3 y 25 caracteres.
- **Dni**. Solo se permiten 9 caracteres.
- **Contraseña**. Se genera una contraseña de forma automática formada por las tres primeras letras del nombre y las 3 primeras del apellido. Posteriormente un usuario manager puede cambiar su contraseña por defecto.
- **Administrador**. Si tiene o no permisos de administrador.

Cuando el usuario rellena los datos, la vista se comunica con el controlador *TFG_player_state* que comprueba que los datos son correctos, si es así se comunica con el gestor de managers ***GestorManagers*** pasándole la ruta al script PHP ***registro_manager_script*** y ejecuta una consulta de búsqueda en la base de datos, si el dni introducido no existe entonces se registra al manager en la base de datos, en caso contrario el controlador comunica a la vista el error.

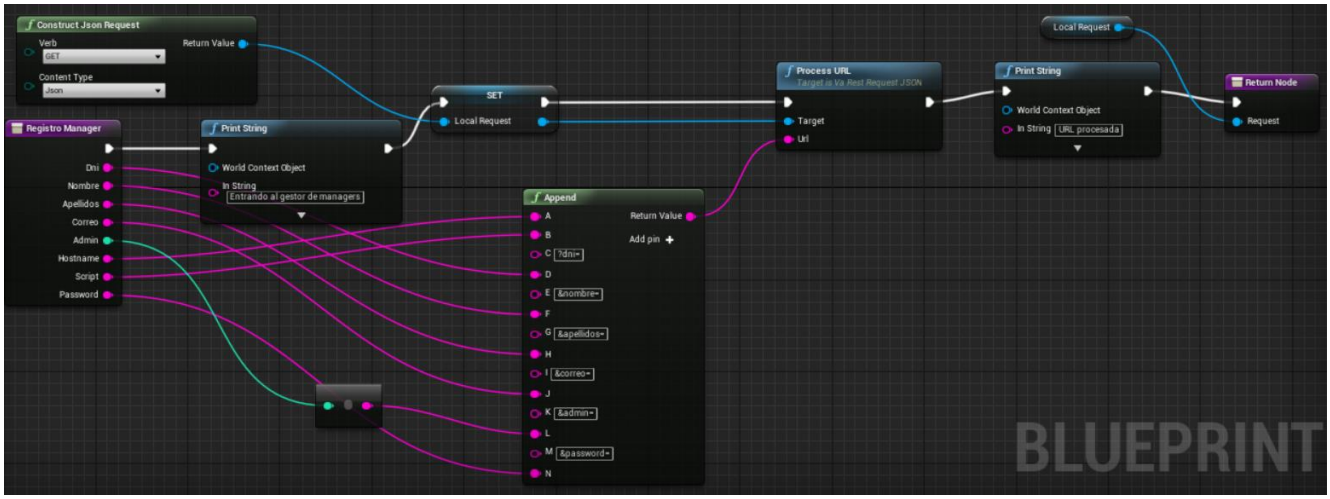


Figura 103: Función de registro de manager del gestor de managers

Además, al igual que en la mayoría de interfaces del usuario manager, existe un botón para salir de la aplicación y otro que permite volver a la interfaz anterior.

Figura 104: Evento lanzado cuando se hace click sobre el botón 'back_btn'

7.3.4 Interfaz de Registrar jugador

Es accesible desde la interfaz de *MainMenu* y desde la interfaz *CrearPartida* y permite a un usuario manager registrar a un usuario jugador indicando:

- **Nombre**. Permitiendo un máximo de 15 caracteres y un mínimo de 3. Únicamente se permiten letras.
- **Apellidos**. Acepta un máximo de 25 caracteres y un mínimo de 3, sólo se están permitidas las letras.
- **DNI**. Únicamente permite 9 caracteres.
- **Edad**. Mediante un componente *SpinBox* permite un rango de edad entre 18 y 65 años.
- **Experiencia**. Si el usuario tiene experiencia previa o no en el campo de trabajo de la aplicación, se recoge mediante un componente checkBox.

Una vez los campos han sido rellenados por el usuario, la vista pasa los datos al controlador que los verifica, si estos son correctos se comunica con la clase del modelo ***GestorJugadores*** que accede a la base de datos para registrar al jugador. Si el jugador ya ha sido registrado previamente en la base de datos, el controlador informa a la vista del error.

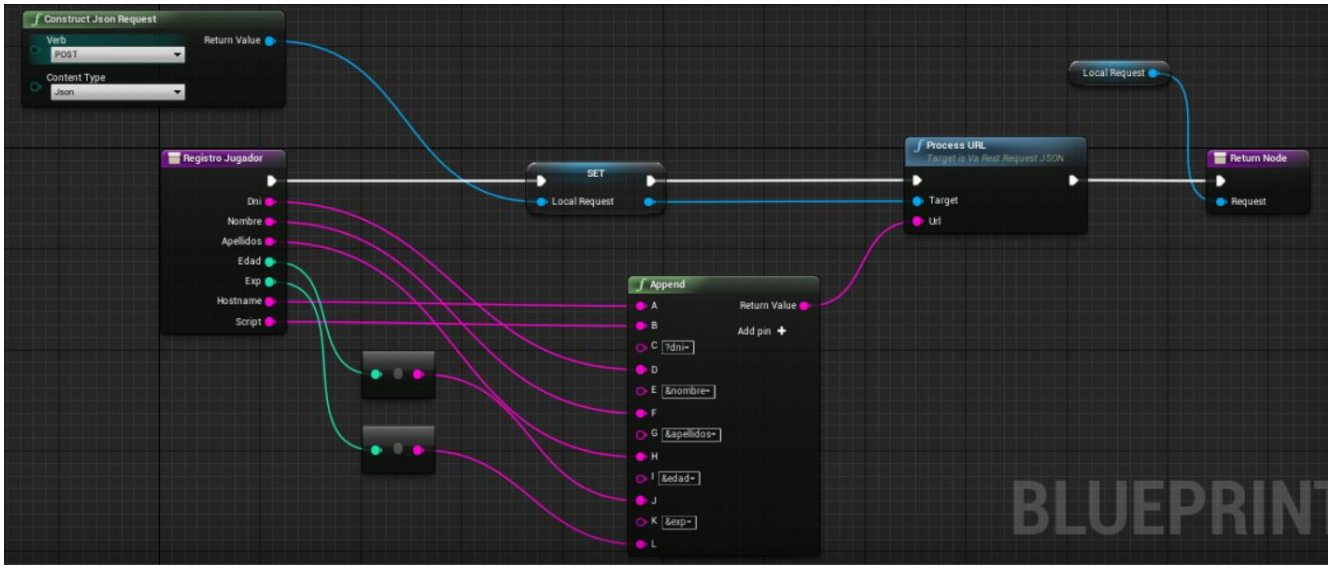
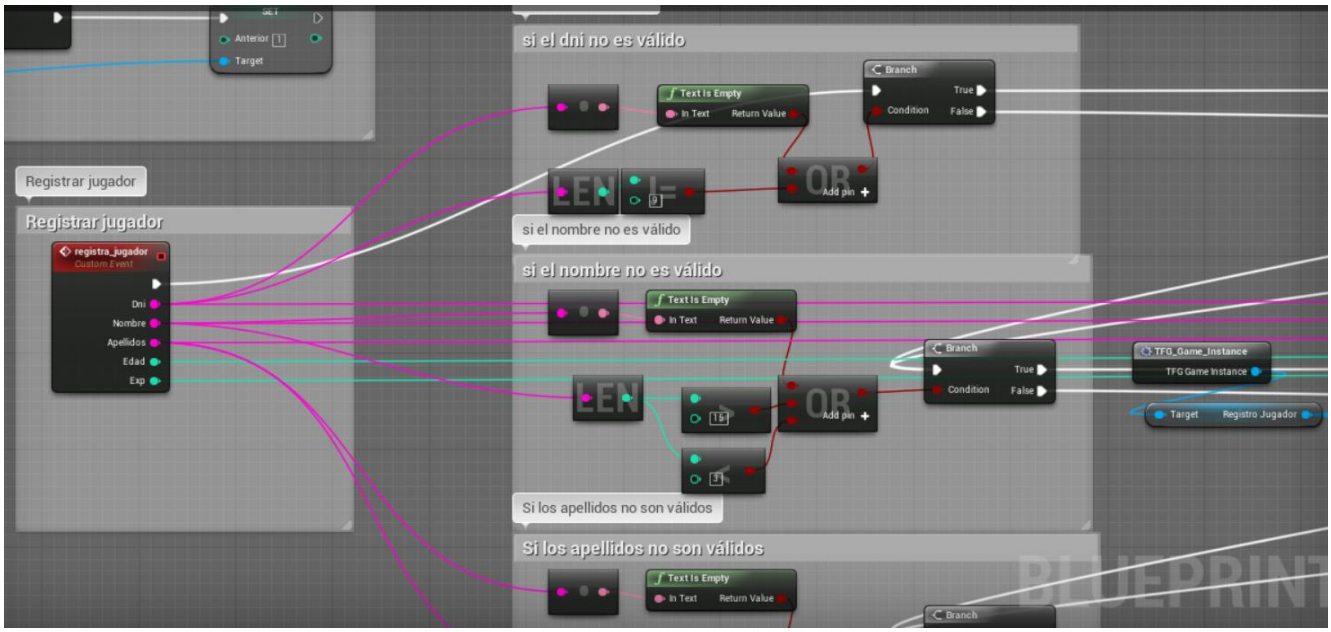


Figura 105: Función de registro de jugador del gestor de jugadores

7.3.5 Interfaz de Crear partida

A través de esta interfaz un usuario manager puede crear una partida, para ello primero debe introducir un documento de identidad perteneciente a un jugador previamente registrado. Si el documento del jugador introducido no coincide con ninguno existente en la base de datos, entonces el usuario manager puede registrar al jugador lo que le llevaría a la interfaz 'Registrar jugador'.

Una vez introducidos los datos, el usuario manager selecciona el escenario en el que quiere que se desarrolle la partida (únicamente disponible el escenario de fabricación) para el jugador con el dni introducido, la vista informa al controlador que comprueba que los datos del jugador son correctos.



Si el jugador está registrado, el controlador *TFG_gameInstance* **crea una sesión LAN** en la red a la que podrá unirse el jugador desde otro dispositivo conectado a la misma red y además se almacenan los datos del jugador con el dni introducido en una variable de tipo '**Jugador'**.

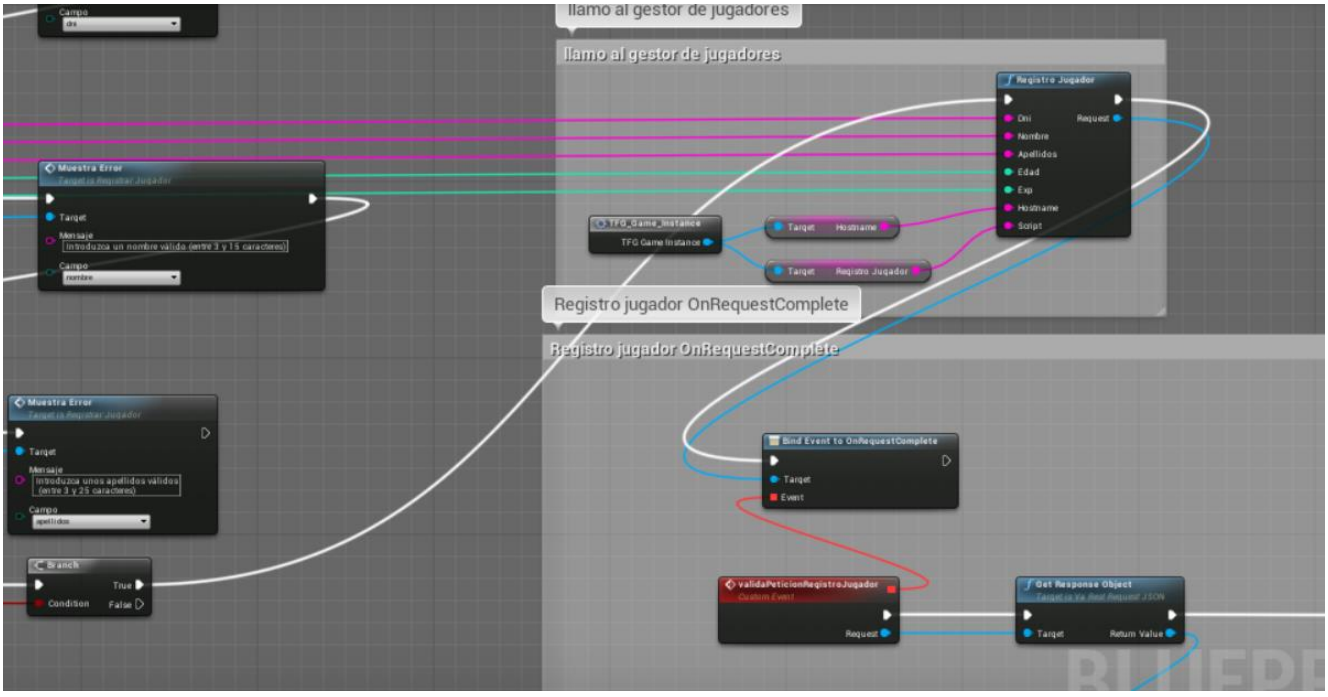


Figura 106: Llamada a la función 'Registro_Jugador' del gesto de jugadores y ligado de la de la respuesta a un evento local

El controlador *TFG_gameInstance* crea la **interfaz de espera de jugadores** hasta que un usuario se conecte a la sesión. Cuando un usuario se une a la sesión entonces el controlador **inicia el escenario de juego** seleccionado con el parámetro 'listen' que hace que el servidor escuche los eventos del jugador y no participe de forma directa en el escenario de juego.

Una vez iniciado el nivel, este ejecuta se comunica con el controlador *TFG_gameInstance* para que cree la interfaz de espera de jugadores hasta que un jugador se una a la partida.

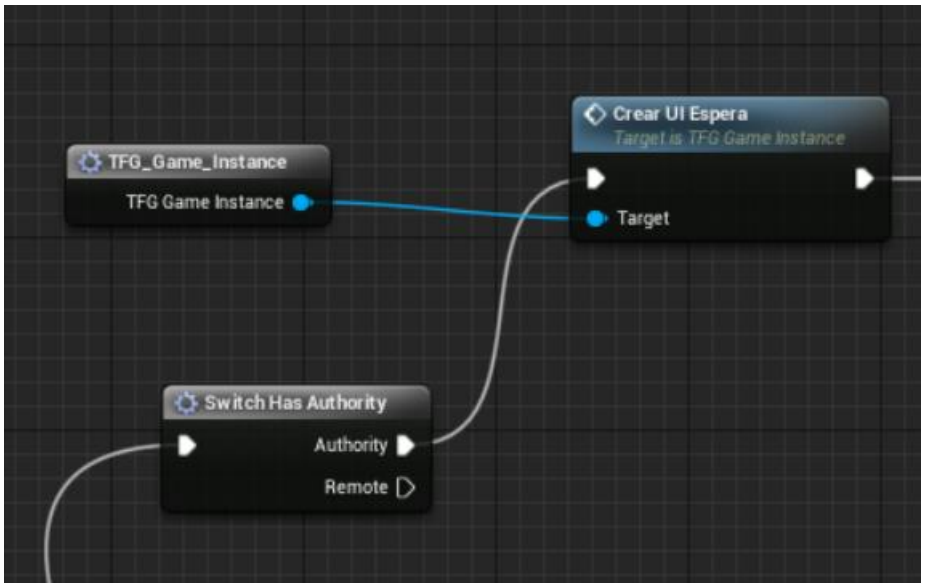


Figura 107: Llamada al evento del TFG_gameInstance 'Crear_UI_Espera' desde el blueprint del nivel EscenarioFabricacion

7.3.6 Interfaz de espera de jugadores

La interfaz de espera de jugadores sirve como transición desde que se crea una partida hasta que se une el jugador correspondiente y se inicia la aplicación. Está formada por un componente *circular throbber* y dos componentes de texto que informan del estado del juego al usuario, de modo que cuando se une el jugador, el controlador *TFG_player_state* informa a la vista de que se actualice, de modo que el usuario sepa que va a iniciarse la partida. Una vez iniciado el escenario de juego, comunica al controlador *TFG_gameInstance* que inicie la interfaz de **panel de control** del manager.

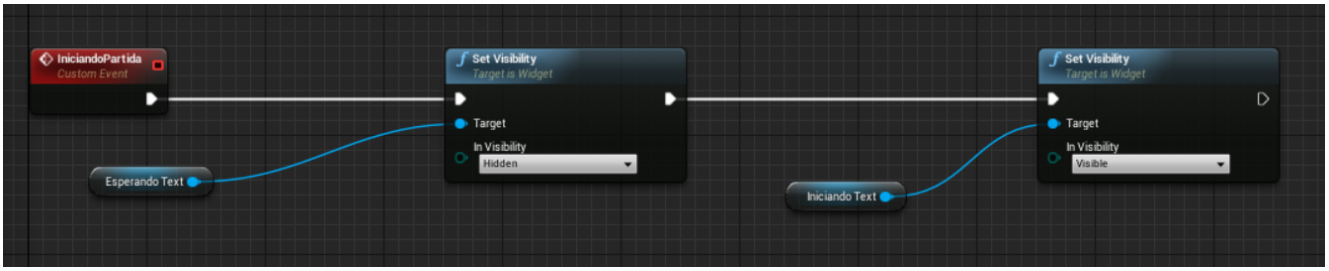


Figura 108: Modificación del entorno del nivel 'Inicio_jugador' para informar al usuario del estado del juego

7.3.7 Interfaz Panel de Control

Se trata del centro de operaciones desde el que el usuario manager gestionará la partida de un jugador en tiempo real. Desde este panel de control un usuario manager puede realizar diversas acciones que dividiremos en secciones en las que se detalla cómo se ha implementado cada una de ellas.

7.3.7.1 Minimapa

El usuario manager puede ver al jugador en el escenario a través de una pequeña ventana en el panel de control, además, puede navegar entre las diferentes vistas disponibles implementadas en el panel.

Se posiciona un componente especial '**SceneCapture2D'** que contiene una cámara capaz de capturar una imagen de un fragmento de del escenario del juego en 2 dimensiones según la localización y orientación del actor dentro del escenario.

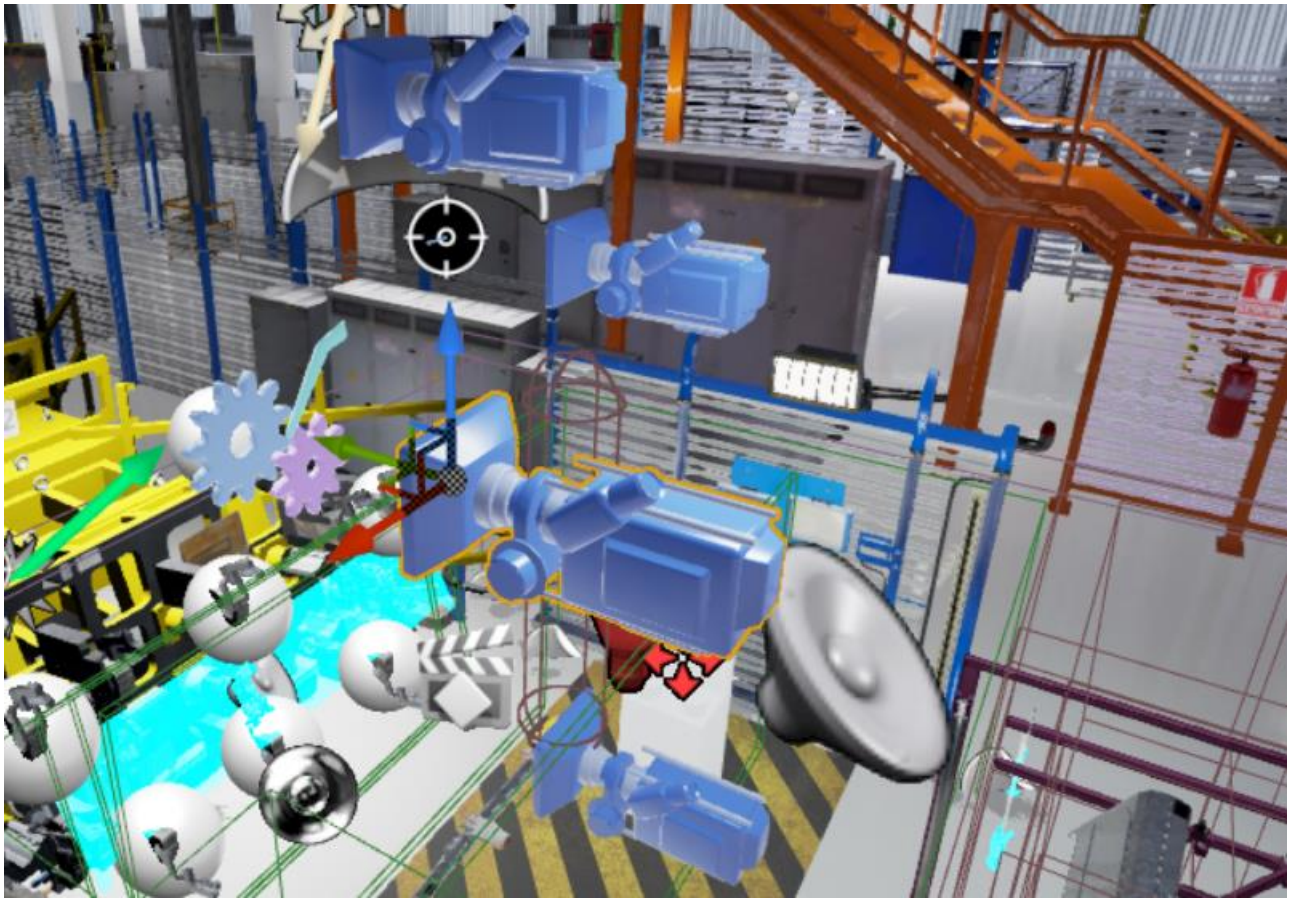


Figura 109: Componentes SceneCapture2D posicionados dentro del nivel EscenarioFabricacion

Las imágenes son capturadas constantemente por el componente 'SceneCapture2D' y se utiliza un componente '**Render target**' donde la cámara irá guardando las imágenes y podrá utilizarse como una textura normal del juego. A partir de esta textura se crea un material que se utiliza en un componente 'UImage' en la interfaz del panel de control.

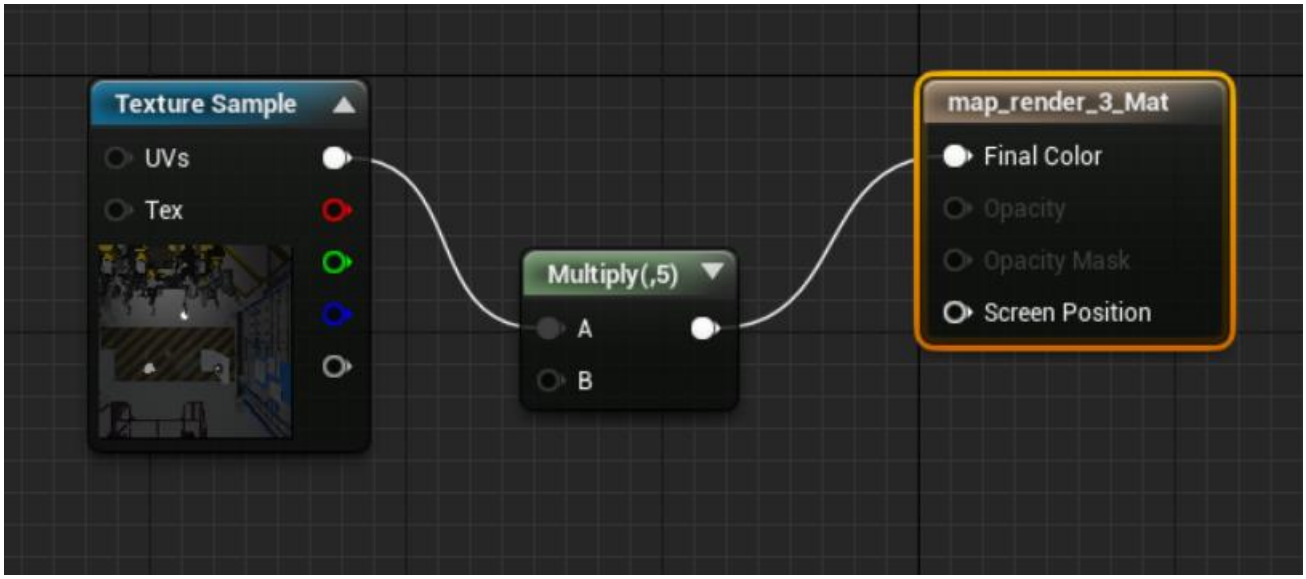


Figura 110: Material que enlaza la textura capturada desde el escenario de fabricación

Se colocan **3 cámaras en diferentes posiciones** del escenario de juego siguiendo los pasos explicados anteriormente para cada una de las cámaras. En la interfaz **tendremos 3 botones diferentes para cambiar la vista** que obtiene el usuario del escenario y el jugador, de forma que cuando se pulsa uno de estos se lanza un evento que hace que se **cambie la textura** actual del componente 'UImage' del panel de control haciendo que se visualice la zona del escenario correspondiente a la captura de la cámara que representa el botón seleccionado.

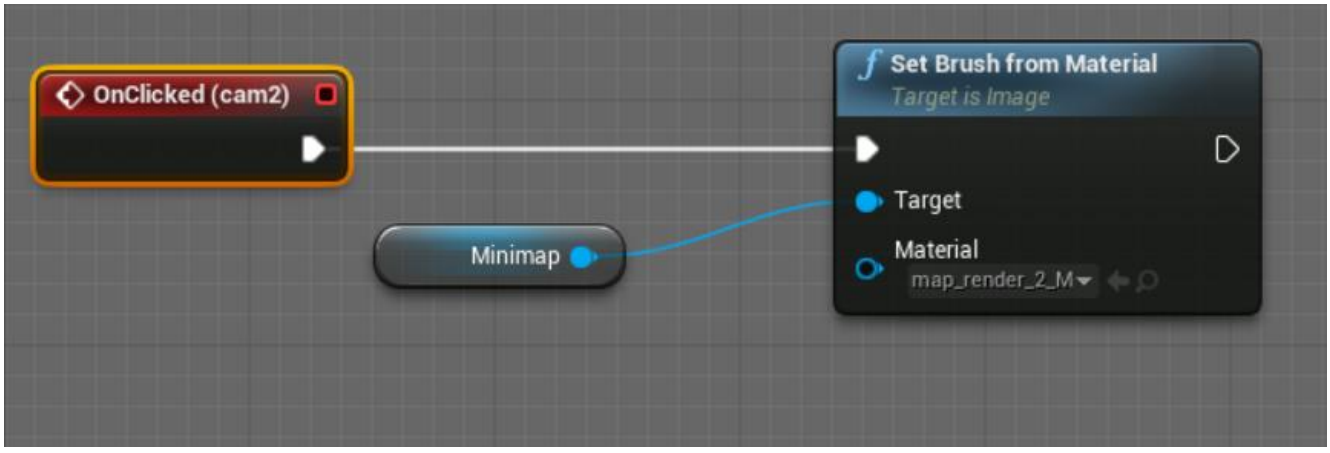


Figura 111: Evento lanzado cuando se pulsa sobre el botón 'cam2' que cambia la vista del minimapa en el panel de control

7.3.7.2 Acción: Proyección de material

Cuando el usuario manager pulsa el botón que reza '**Brazo robot'** dentro del panel de control, se lanza un evento desde la vista que informa al controlador *TFG_player_state* sobre la acción llevada a cabo por el manager y realiza 3 acciones en paralelo. En primer lugar, **llama el *event dispatcher spawn_spark***, en segundo lugar, comunica al controlador *TFG_gameInstance* que registre un evento de tipo ***EventoManager*** y le pasa como argumento el tipo de evento que se ha producido. Por último, comprueba si el jugador se ha puesto las gafas, en caso contrario aumenta el número de fallos graves en el controlador *TFG_gameInstance*.

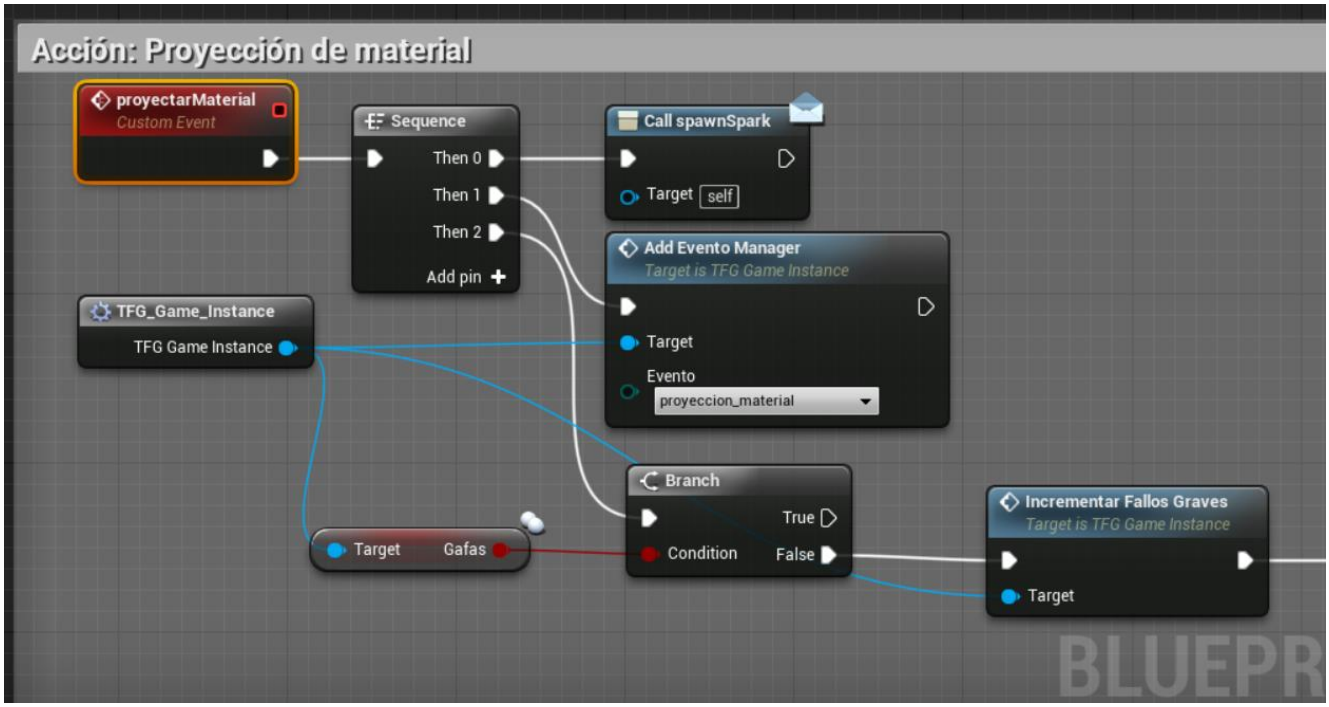


Figura 112: Código de proyección de material en el Pawn VRPawn_P

Cuando se llama al *event dispatcher*, desde el *blueprint* del nivel de juego del escenario de fabricación se liga el evento del controlador a un evento local que se ejecutará cuando ocurra el evento en el controlador y que **se replica en modo 'Multicast'** esto es importante ya que si el evento se ejecuta de cualquier otro modo no se replicará al jugador y por tanto no surtirá efecto.

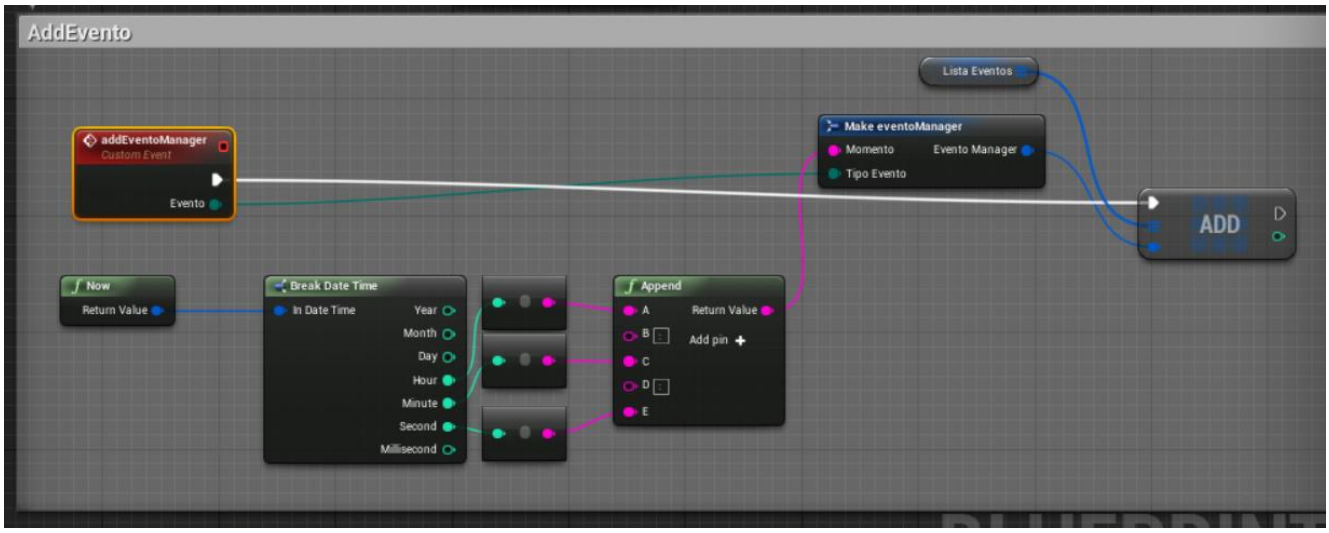


Figura 113: Evento de registro de una acción del manager producida durante una partida

Cuando esto ocurre se crean unas partículas que simulan que el robot soldador delantero ha lanzado una proyección de material.

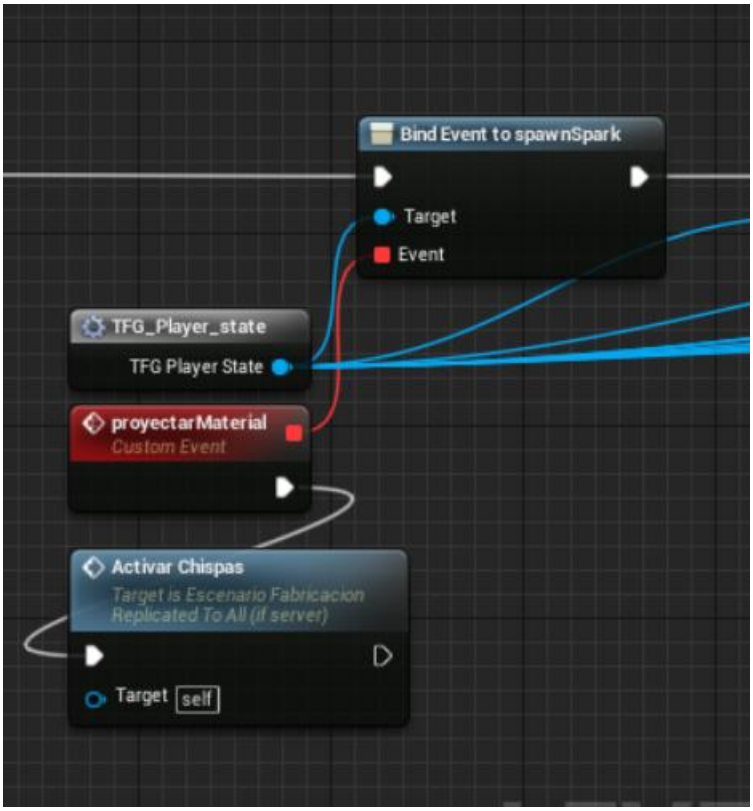


Figura 114: Ligado del event dispatcher 'spawnSpark' del TFG_player_state a un evento local

Se crea un componente emisor de partículas y se deshabilita la opción de volver a llamar a la misma acción hasta pasados al menos 2 segundos.

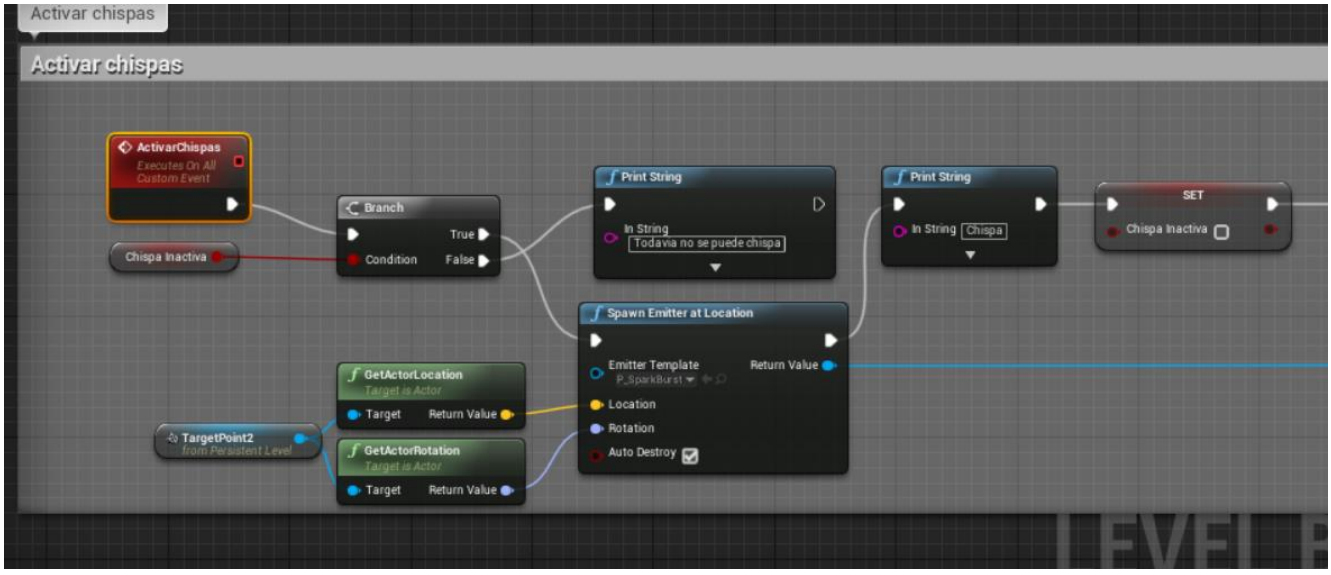


Figura 115: Evento multicast del level blueprint del nivel EscenarioFabricacion que activa las partículas sobre el escenario

7.3.7.3 Acción: Atasque en el robot soldador

Cuando el usuario manager pulsa el botón de '**Atasque'** del panel de control durante una partida en curso, se acciona un evento desde la vista que comunica al controlador *TFG_player_state* que debe llamar al *event dispatcher* '**stopMachine'.**

Figura 116: Evento lanzado desde el TFG_player_state y que atasca el robot soldado del escenario de fabricación

Este evento es ligado a un evento local desde el *level blueprint* del escenario donde transcurre la partida. Cuando ocurre el evento se crea un objeto animado en el escenario que le indica al jugador que el robot se ha atascado y se replica al cliente a través de la red.

Figura 117: Evento del blueprint del nivel que atasca el robot visualmente y cambia el estado de la máquina en el Pawn

Además, el level blueprint se comunica con el ***pawn*** del cliente e **inhabilita seguir con el proceso** de fabricación hasta que el jugador solvente la situación el fallo.

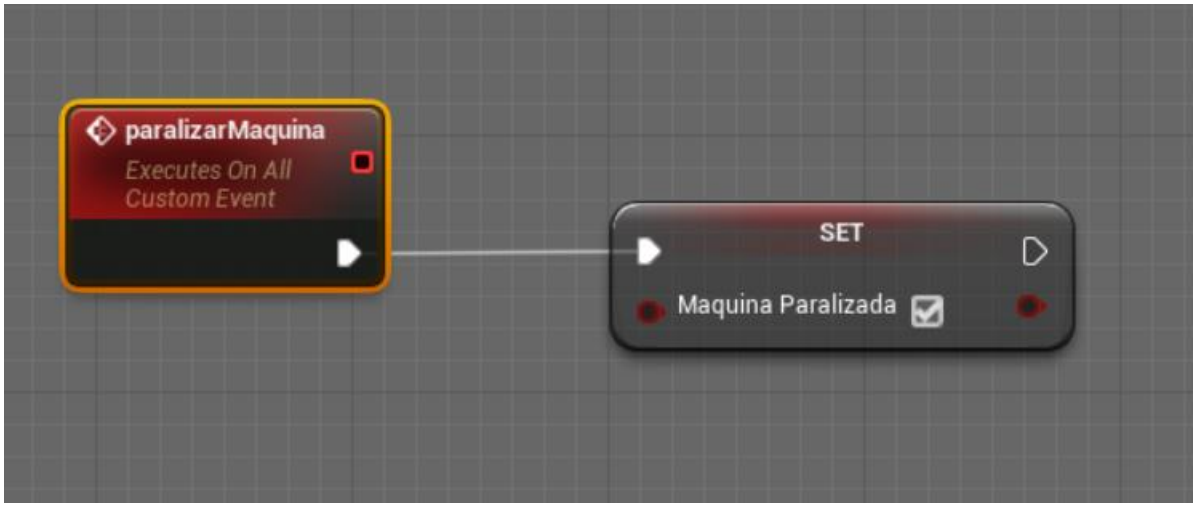


Figura 118: Evento multicast del Pawn 'VRPawn_P'

Al igual que pasaba con la acción de proyección de material, **se registra el evento** a través del controlador para que cuando la partida finalice se guarden los resultados.

7.3.7.3 Acción: Fallo de cadena

Al igual que con el resto de acciones del panel de control, la vista comunica al controlador que el usuario manager ha realizado la acción de fallo en cadena de producción, este que llama al event dispatcher ***fallo_produccion*** que es ligado a un evento local que se ejecuta en modo '**Multicast'** desde el level blueprint.

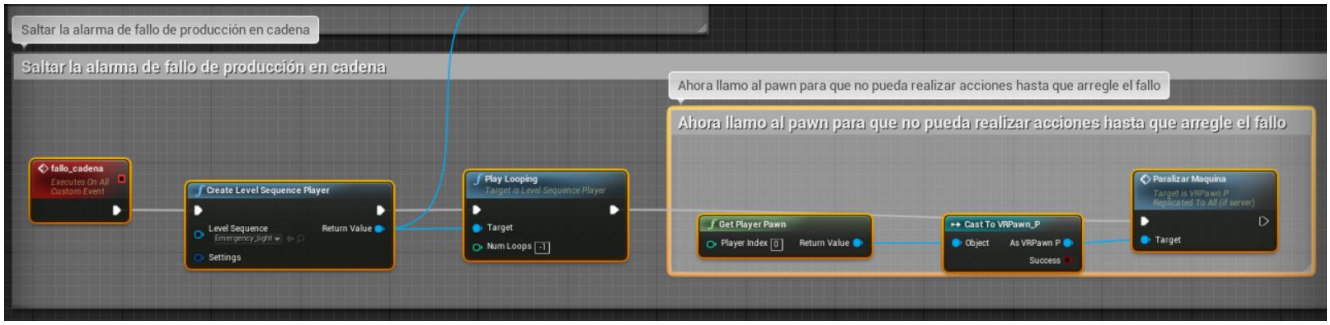


Figura 119: Evento del nivel que activa la alarma del fallo de producción y actualiza el estado de la máquina en el Pawn

Esto acciona una luz roja intermitente en el escenario mediante una secuencia de nivel que se replica al cliente a través de la red y, al igual que con la acción de atasque del robot soldador, el nivel se comunica con el pawn del jugador para inhabilitar que siga con el proceso de fabricación hasta que solvente el problema. Igual que con el resto de acciones, el controlador registra la acción del servidor en el tiempo que se produjo.

7.3.7.4 Acción: Corte por barra metálica

Al igual que con el resto de acciones del manager, cuando el usuario manager pulsa el botón de 'Corte' durante una partida, la vista informa al controlador de la ocurrencia del evento y este a su vez llama al event dispatcher '**corteDispatcher'** que es ligado desde el *level blueprint* del escenario de fabricación a un evento local que se ejecuta en modo Multicast.

Si en el momento de la acción el jugador está sujetando una barra sin protecciones entonces el controlador *TFG_player_state* informa al controlador *TFG_gameInstance* que actualiza los fallos

graves del jugador y además informa a la vista de que actualice el contador de fallos en la interfaz gráfica.

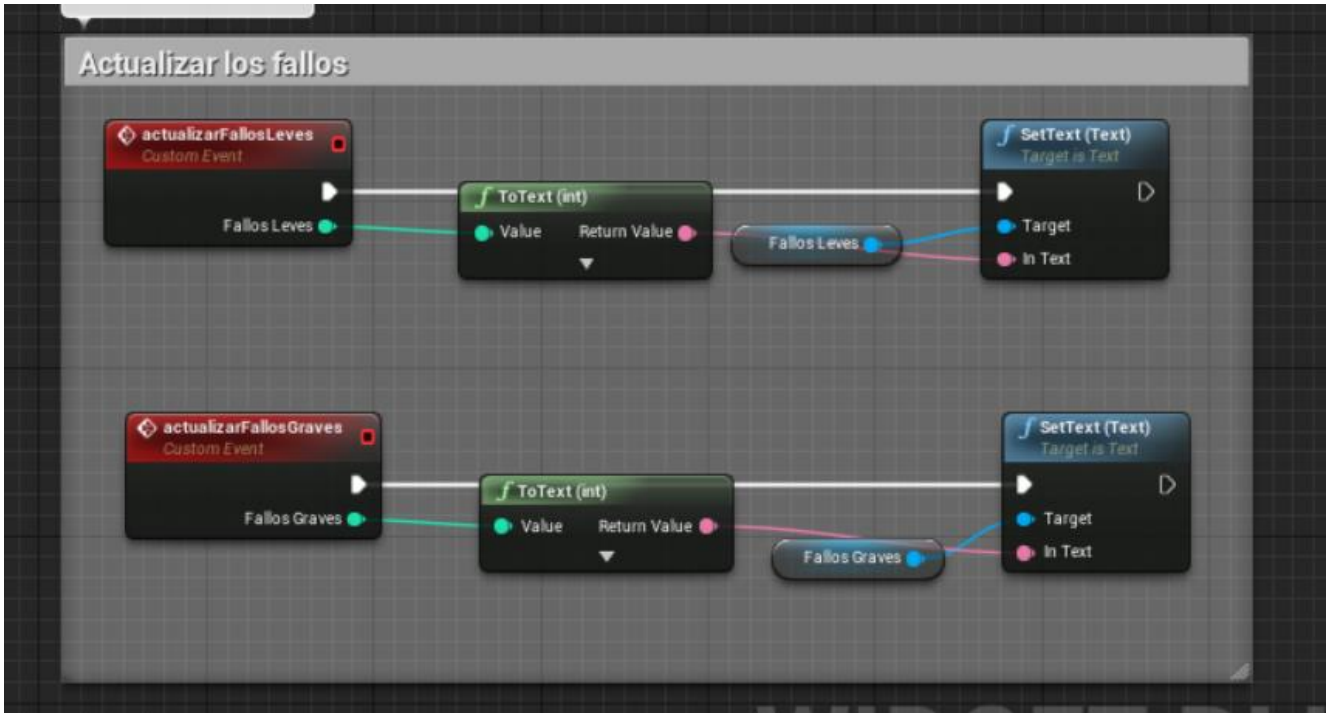


Figura 120: Eventos de actualización de los fallos del jugador en el panel de control del servidor

Además, **se registra el evento** accionado en el controlador *TFG_gameInstance*.

7.3.7.5 Acción: Corte por barra metálica

Cuando el usuario jugador pulsa uno de los botones para coger cualquiera de las piezas metálicas, este **comunica al servidor que está cogiendo una barra**, entonces el controlador *tfg_player_state* se comunica con la interfaz del panel de control y actualiza el botón de cortar jugador del panel, es entonces cuando el usuario manager puede accionar este botón, cortando al jugador.

Una vez el usuario manager acciona el botón de corte, la interfaz de panel de control se comunica con el controlador *tfg_player_state* y ejecuta el evento *cortar_jugador* que realiza 3 pasos.

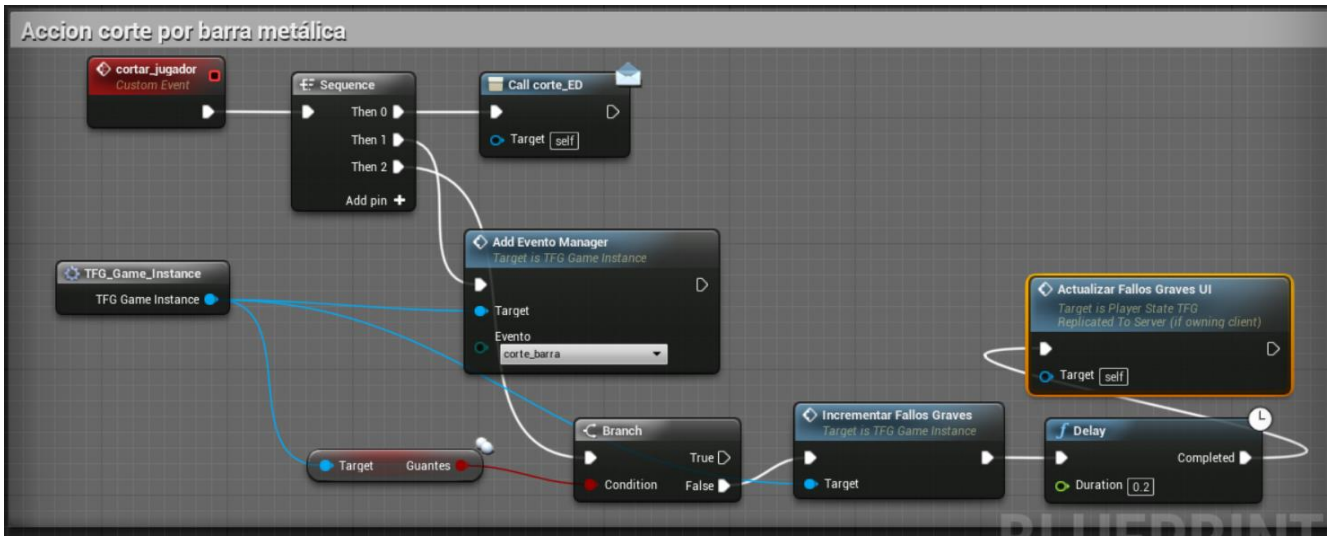


Figura 121: Secuencia de acciones realizadas por el 'tfg_player_state' cuando el manager pulsa el botón de corte

En primer lugar, llama al *event dispatcher* ‘corte_ED’ después añade el evento producido al controlador *TFG_gameInstance* y comprueba si el jugador lleva guantes, si no es así, se incrementan los fallos graves en la interfaz del panel de control y en el controlador *TFG_gameInstance*. El event dispatcher es **ligado esta vez desde el Pawn del jugador** haciendo que se actualice el material de las manos del jugador simulando un corte en la mano.

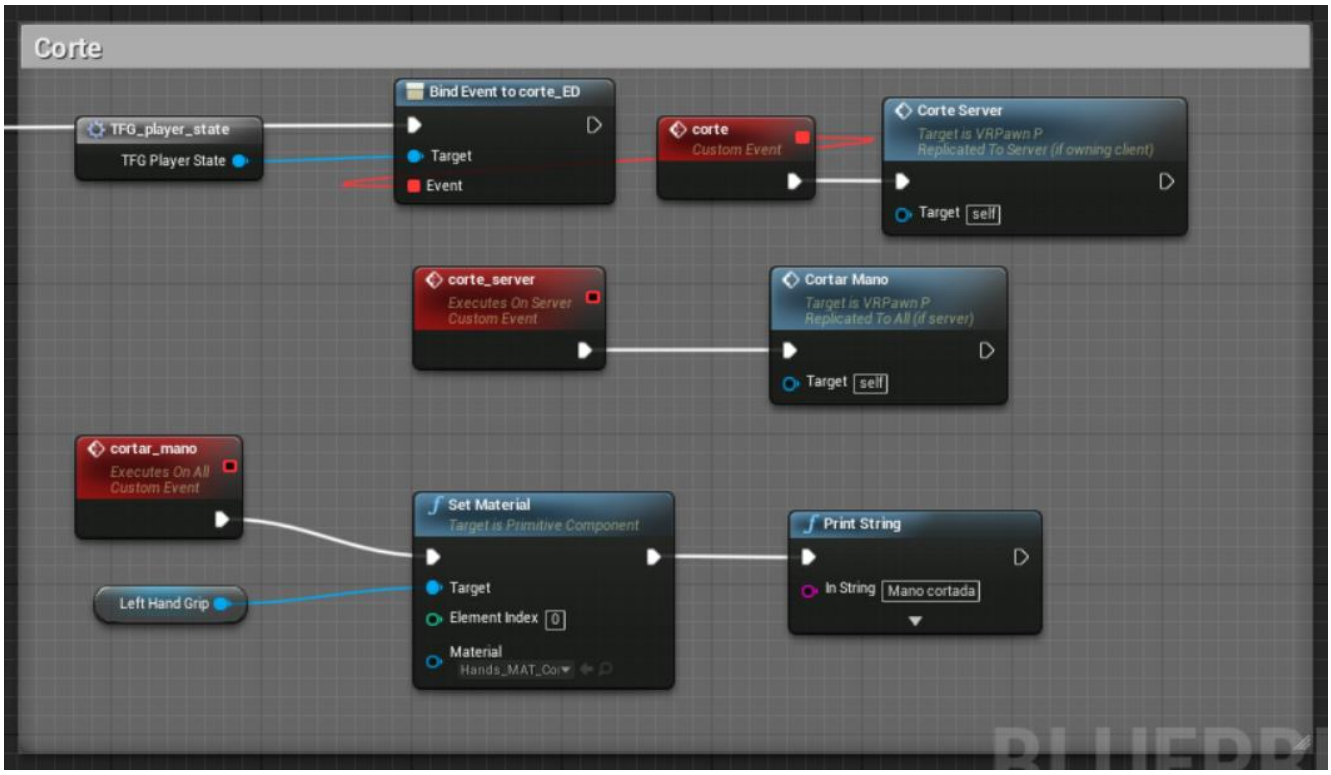


Figura 122: Ligado del event dispatcher 'corte_ED' desde el Pawn del jugador

7.3.7.6 Contador de juego

Cuando se inicia una partida, el panel de control inicia un contador que indica el **tiempo transcurrido** desde que un jugador inicia una partida hasta que se termina, bien porque el jugador terminó el juego o porque el manager decide cancelarla antes de tiempo. Este contador se puede ver en el submenú inferior del panel de control debajo de la etiqueta ‘TIEMPO’.

Un evento de tipo ‘**Event Tick’** se encarga de actualizar una variable que contiene los segundos transcurridos desde el inicio de la aplicación.

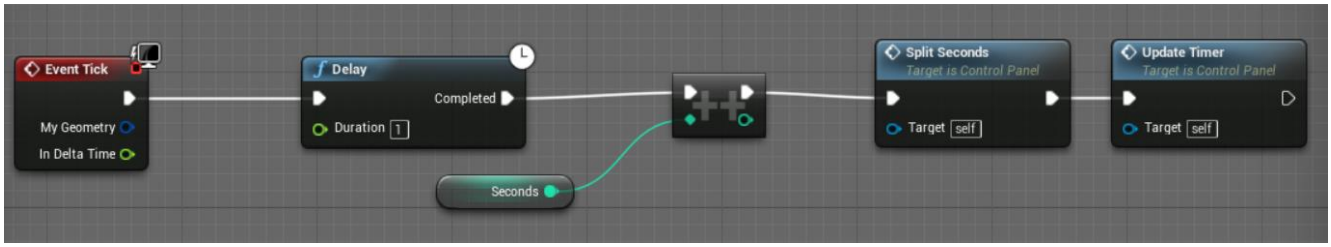


Figura 123: Evento que actualiza el contador de tiempo en el panel de control

Cuando los segundos llegan a 60 se actualiza la variable que contiene los minutos incrementándola en 1, de este modo se representa el tiempo en el panel de control.

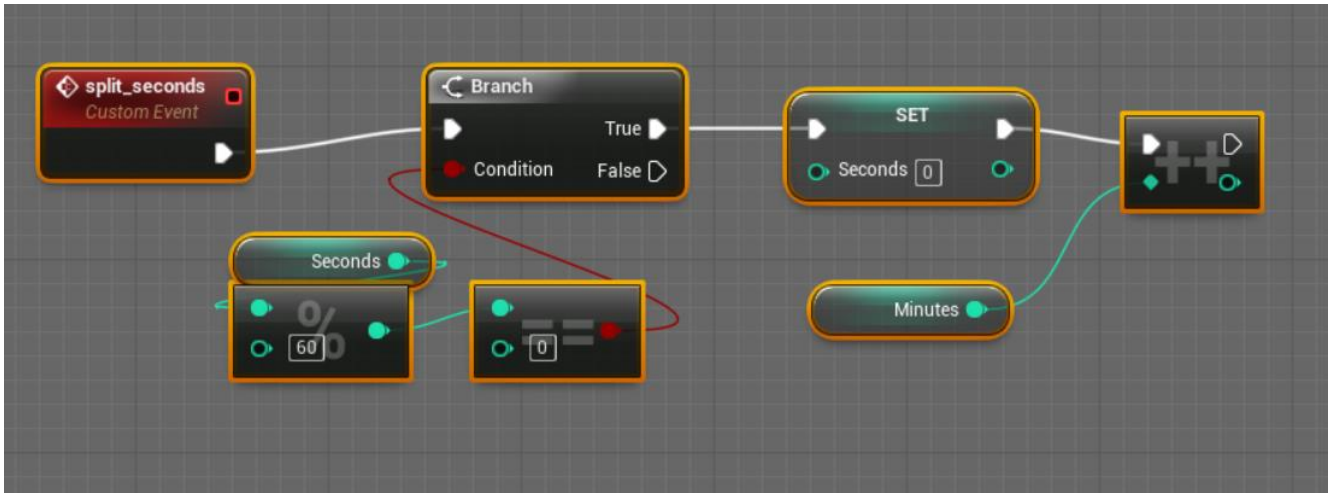


Figura 124: Evento que actualiza las variables del contador

7.3.7.7 Indicadores del estado del juego

Existen 3 tipos de indicadores en el juego:

- **Protecciones** de seguridad o EPIS: Formada por unos guantes de seguridad, un casco y unas gafas de protección.
- **Secuencia 1**: indica el número de barras colocadas correctamente de la primera secuencia de trabajo de la máquina.
- **Secuencia 2**: indica el número de barras colocadas correctamente de la segunda secuencia de trabajo de la máquina.

Cuando el jugador se pone una de las protecciones, este lo comunica al servidor a través del *Pawn* que lanza un mensaje RPC y que comunica mediante un evento que **únicamente se ejecuta en el servidor** (Run on Server) al controlador *TFG_player_state* que actualiza la acción correspondiente a cada uno de los estados del juego. La parte de comunicación del jugador se detalla en el apartado 7.3.

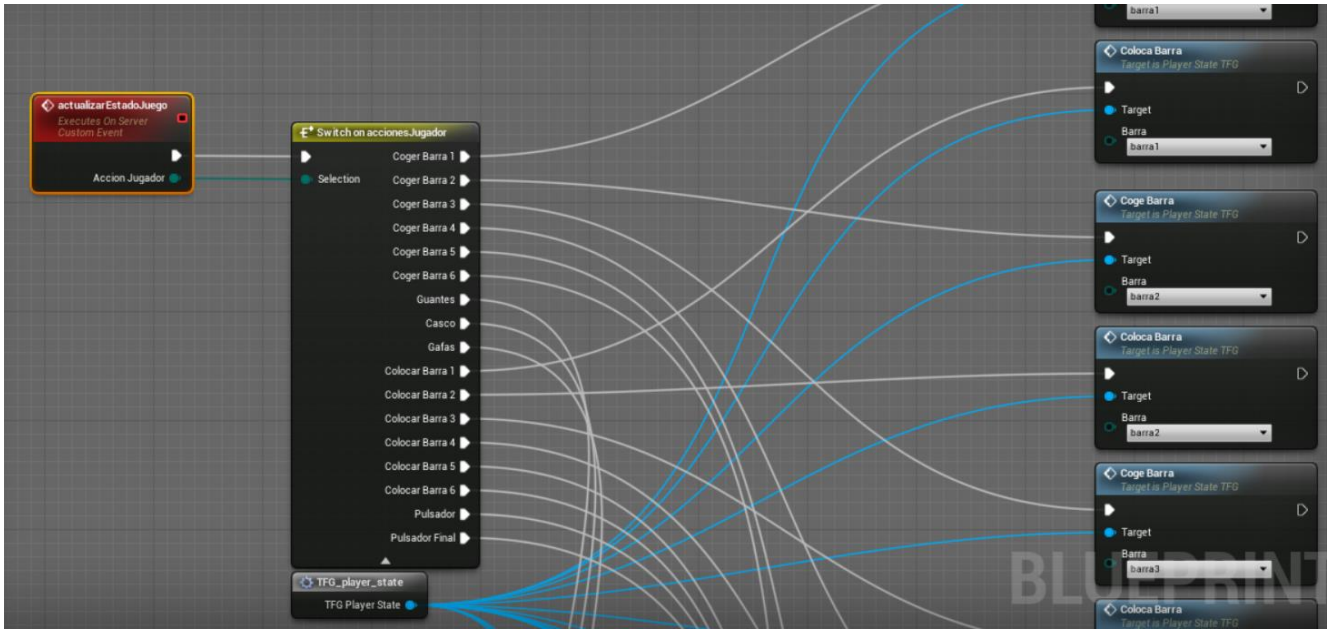


Figura 125: Evento que se ejecuta en el servidor y que ejecuta una de las acciones del jugador en función de la entrada

7.3.7.8 Consultar datos del jugador

Si el usuario manager pulsa en la pestaña donde pone ‘Jugador’ en la interfaz *controlPanel*, entonces se muestra la **información del jugador actual en la partida**. Esto ocurre porque desde la vista se lanza un evento que oculta parte de las zonas de la interfaz y muestra otras. La información del jugador es proporcionada por el controlador *TFG_player_state* al inicio de la partida.

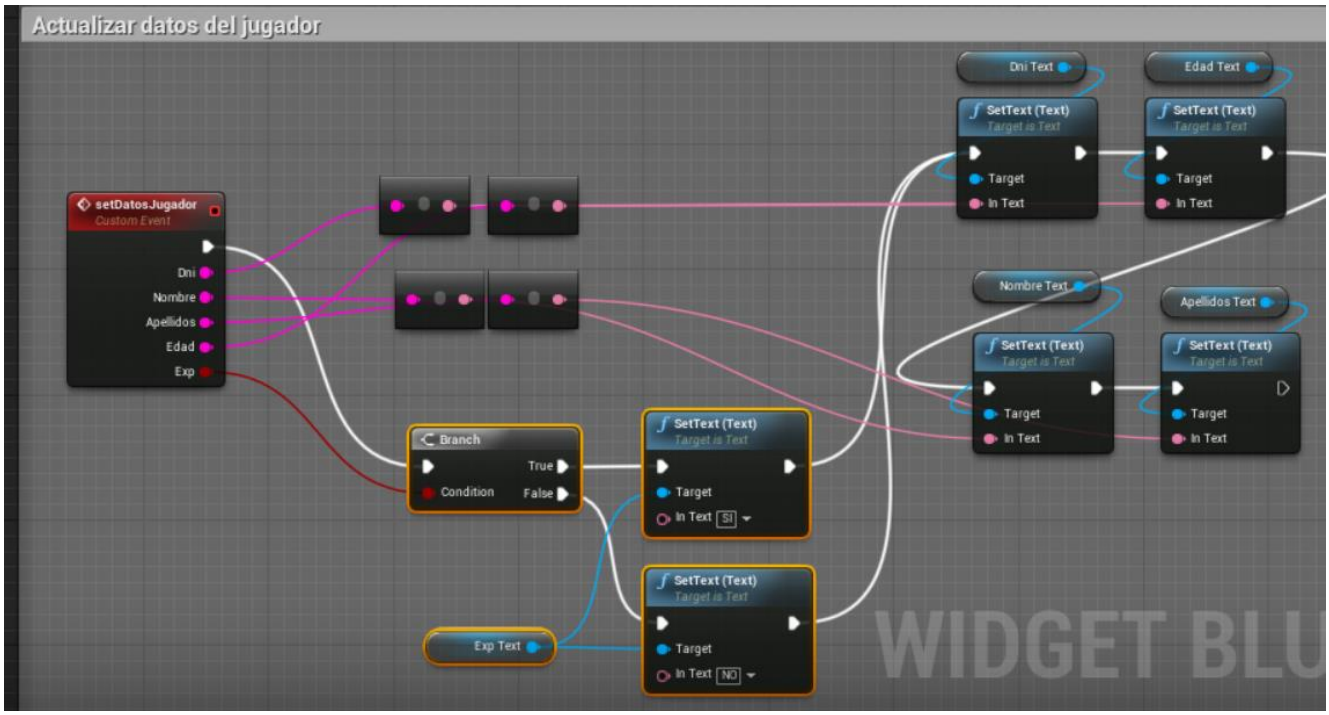


Figura 126: Evento 'setDatosJugador' que actualiza los datos del jugador en el panel de control

7.3.7.9 Guardar resultados y cancelar partida

Si el usuario manager decide cancelar la sesión de juego antes de que el usuario jugador haya terminado la partida se cancela la sesión de juego y se informa al cliente de la sesión.

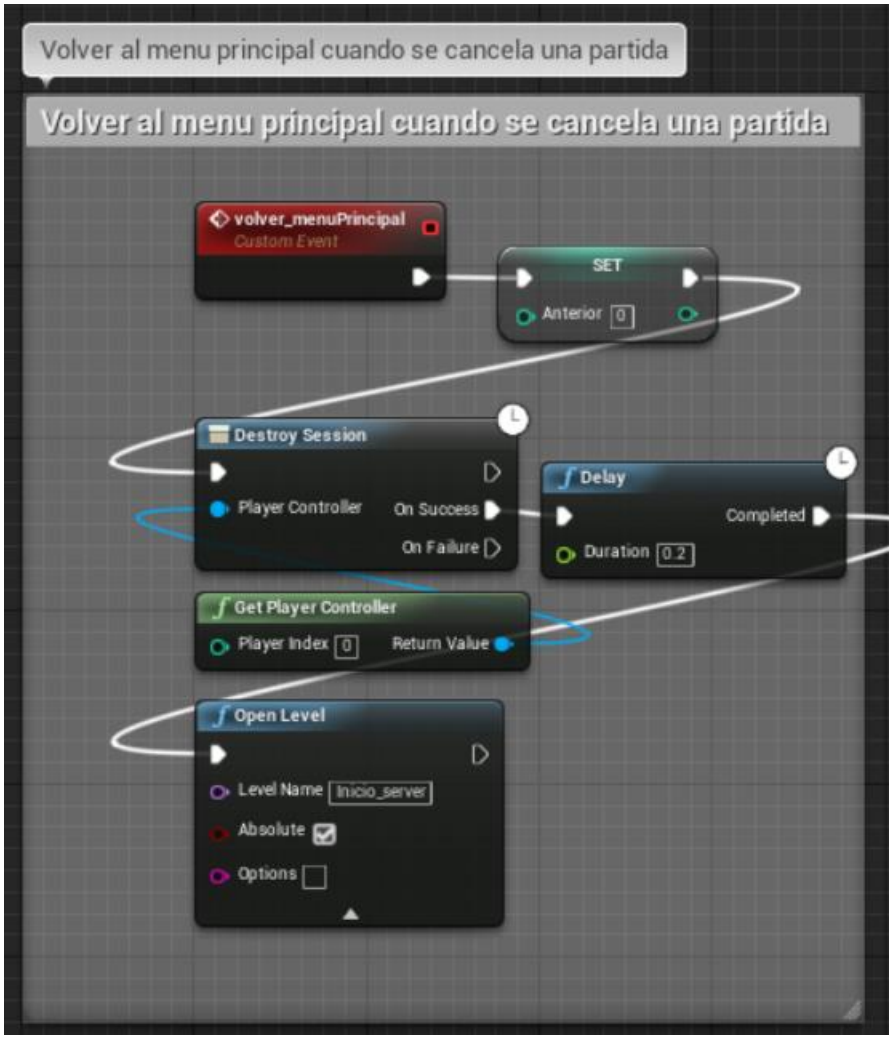


Figura 127: Se destruye la sesión y se vuelve a abrir el nivel 'Inicio_server' en el controlador TFG_gameInstance

Cuando el usuario jugador termina el proceso de fabricación, la partida termina y se manda un mensaje de finalización RPC a través del *Pawn* del jugador, el controlador *TFG_player_state* recoge la llamada y ejecuta dos acciones.

En primer lugar, llama a un *event dispatcher* que será ligado a un evento local desde el *level blueprint* del escenario de fabricación y que hará que se ejecute la animación final de la maquina principal e inhabilita al jugador de seguir realizando acciones desde el Pawn *VRPawn_P*.

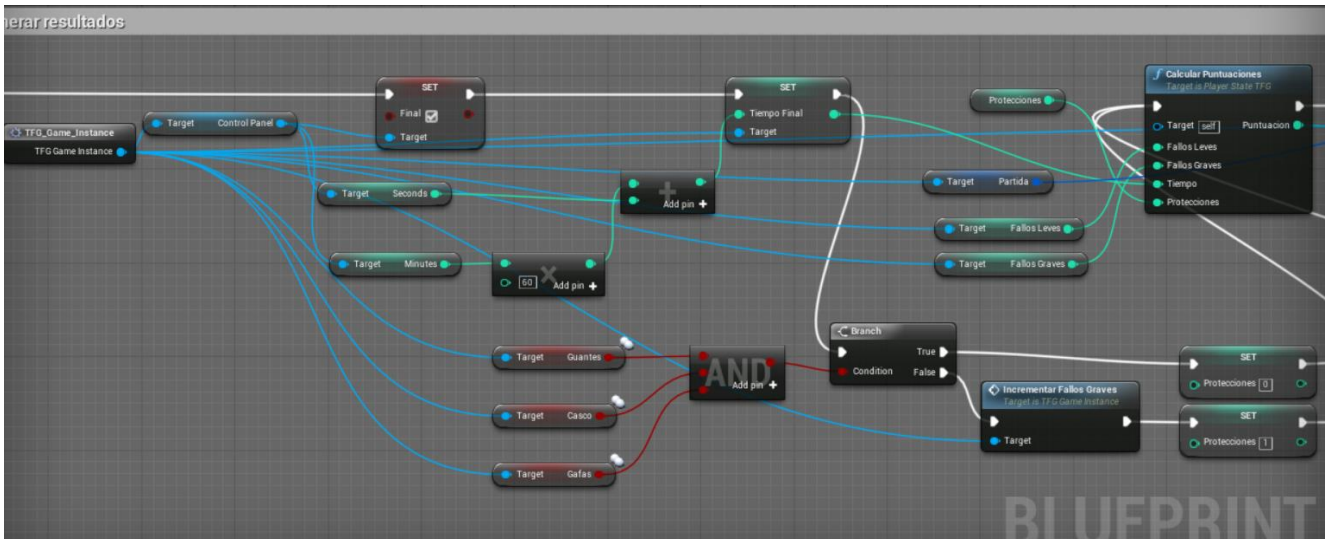


Figura 128: Código de generación de resultados en el controlador TFG_player_state

En segundo lugar, llama a la función de la interfaz panel de control y muestra un **mensaje de confirmación al usuario manager**, si este confirma la acción se generan los resultados de la partida llamando a la función '**Calcular puntuaciones'** en el *TFG_player_state* y actualiza la variable de tipo **partida** en el controlador *TFG_gameInstance*, si las puntuaciones fueran menores que 0, únicamente se pondría un 0 en la puntuación final. Llama al gestor de partidas y le pasa como argumento dicha partida, este gestor llama a un script PHP que registra la partida y los eventos del manager.

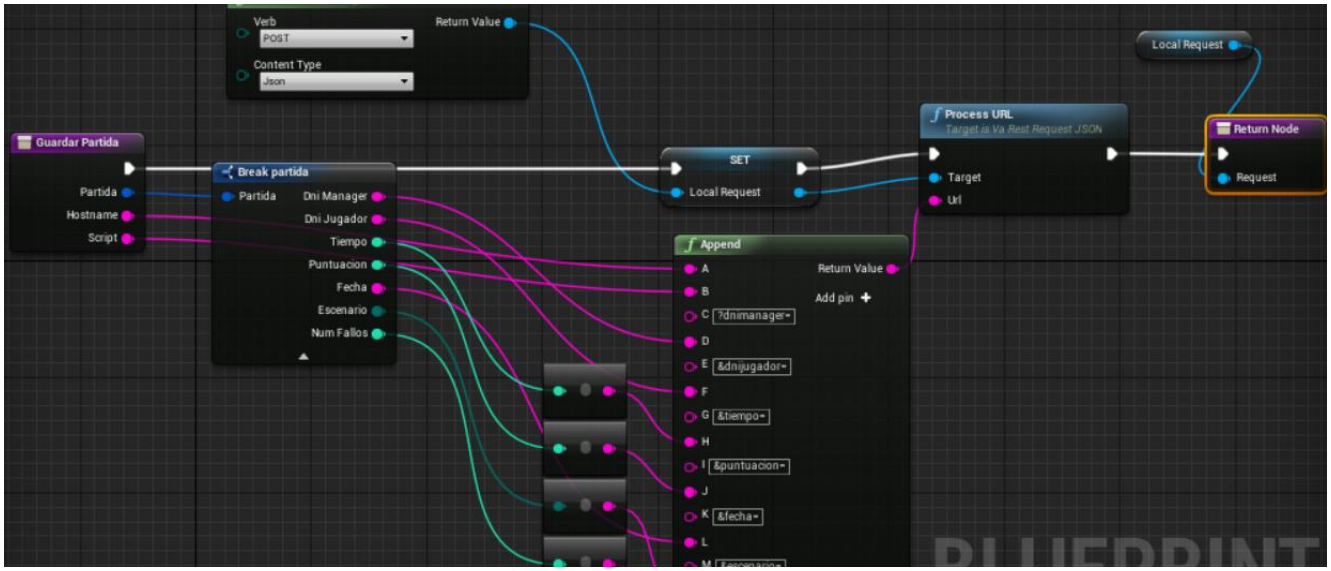


Figura 129: Función de registro de partidas en la base de datos desde el gestor de partidas

Una vez hecho esto, se vuelve cierra la sesión y se vuelve al menú principal de la aplicación, mientras que el usuario jugador vuelve a la sala de espera.

7.4 Fase 3: Funcionalidad del jugador

Esta sección se centra en la implementación desarrollada para el usuario jugador. Debido a que este software se basa en un trabajo previo comercializado, se han suprimido algunas de las funcionalidades del usuario jugador respecto al software original centrándose así la aplicación desarrollada en este trabajo en el panel de control, las sesiones multijugador y el registro de las puntuaciones y las partidas, de modo que el jugador utilizará un mando o un teclado para simular la funcionalidad y no dispondrá de los controladores propios de las gafas de realidad virtual para la realización de las acciones.

7.4.1 Inicio del jugador y búsqueda de partidas

El jugador empieza en el nivel de juego '**Inicio_jugador**' que busca partidas en la red LAN cuando este se inicia.

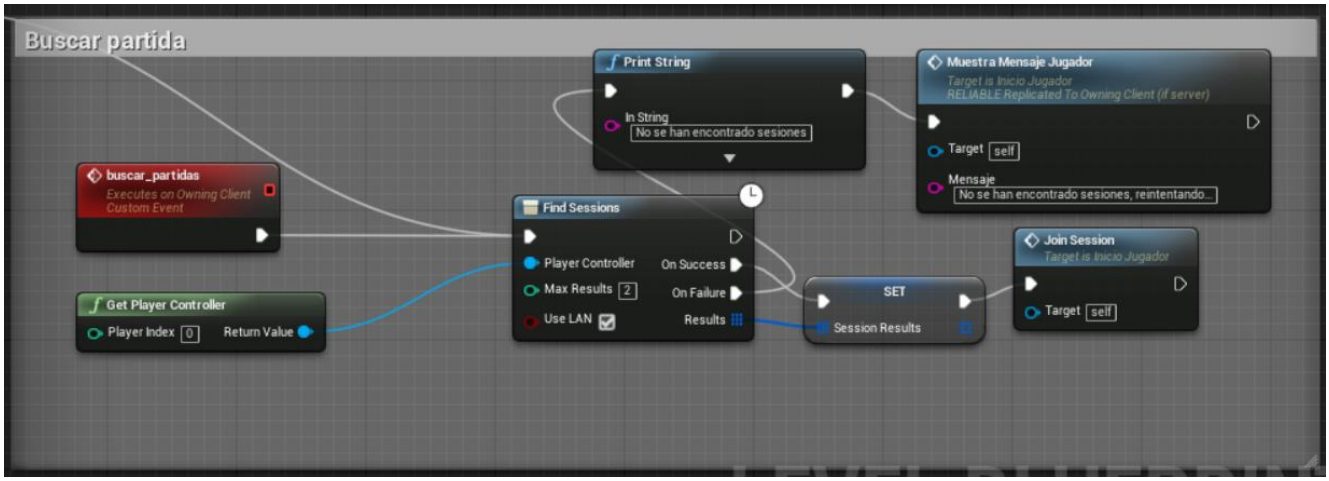


Figura 130: Evento de búsqueda de sesiones en redes LAN desde el level blueprint del nivel 'Inicio_jugador'

Si encuentra una sesión activa en la red se une a esta y carga el nivel del servidor que creó la sesión de juego. Si no se encuentran sesiones activas, entonces vuelve a buscar sesiones en red hasta que exista una.

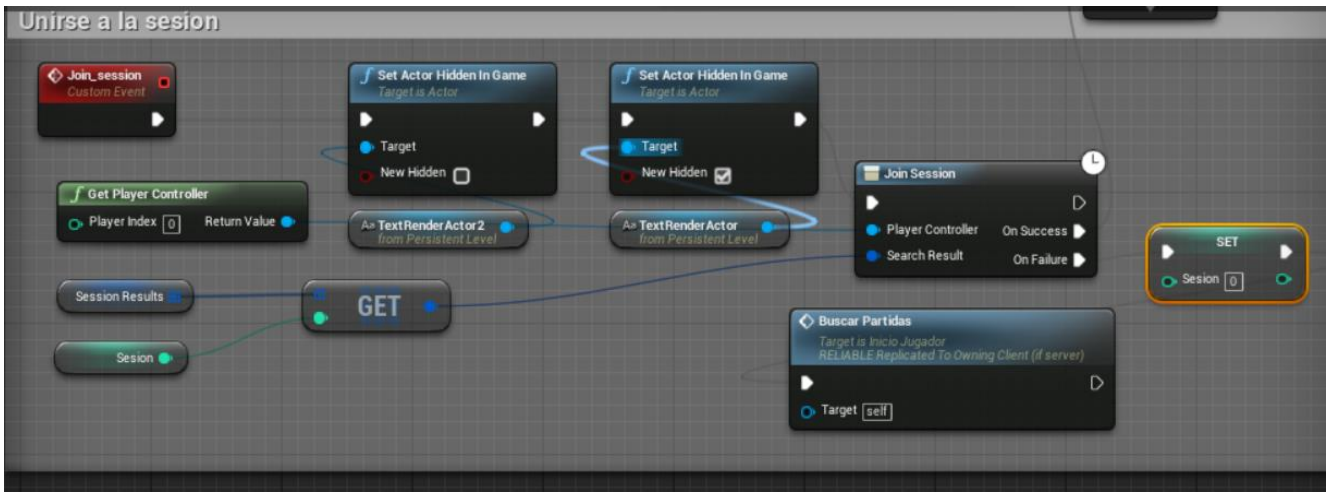


Figura 131: Evento de unión a la sesión desde el level blueprint del nivel 'Inicio_jugador'

7.4.2 Ponerse protecciones de seguridad

El usuario contará con un panel de control visible en el escenario y que le indicará la funcionalidad que puede activar mediante el uso de un mando o teclado, de esta forma cuando se pone los guantes de protección, el pawn **'VRPawn_P'** que es poseído por el *player controller* del jugador comunica esta acción al servidor mediante un evento 'Run on Server' que **sólo se ejecutará en el servidor y que actualiza la vista del Pawn** (sólo el server puede tomar este tipo de decisiones, por eso es necesario llamar al servidor mediante un evento intermedio y que después este llame a un evento 'Multicast'), también actualiza el estado del jugador en el controlador *TFG_gameInstance*.

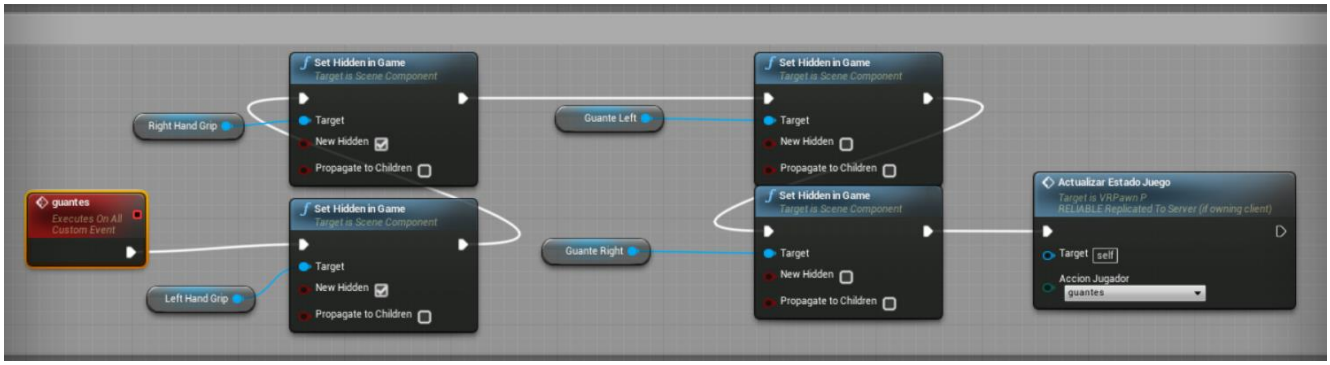


Figura 132: Evento de actualización de los guantes de protección en el Pawn 'VRPawn_P'

Además, esto hace que se actualicen los componentes del pawn mostrando visualmente la protección de seguridad colocada.

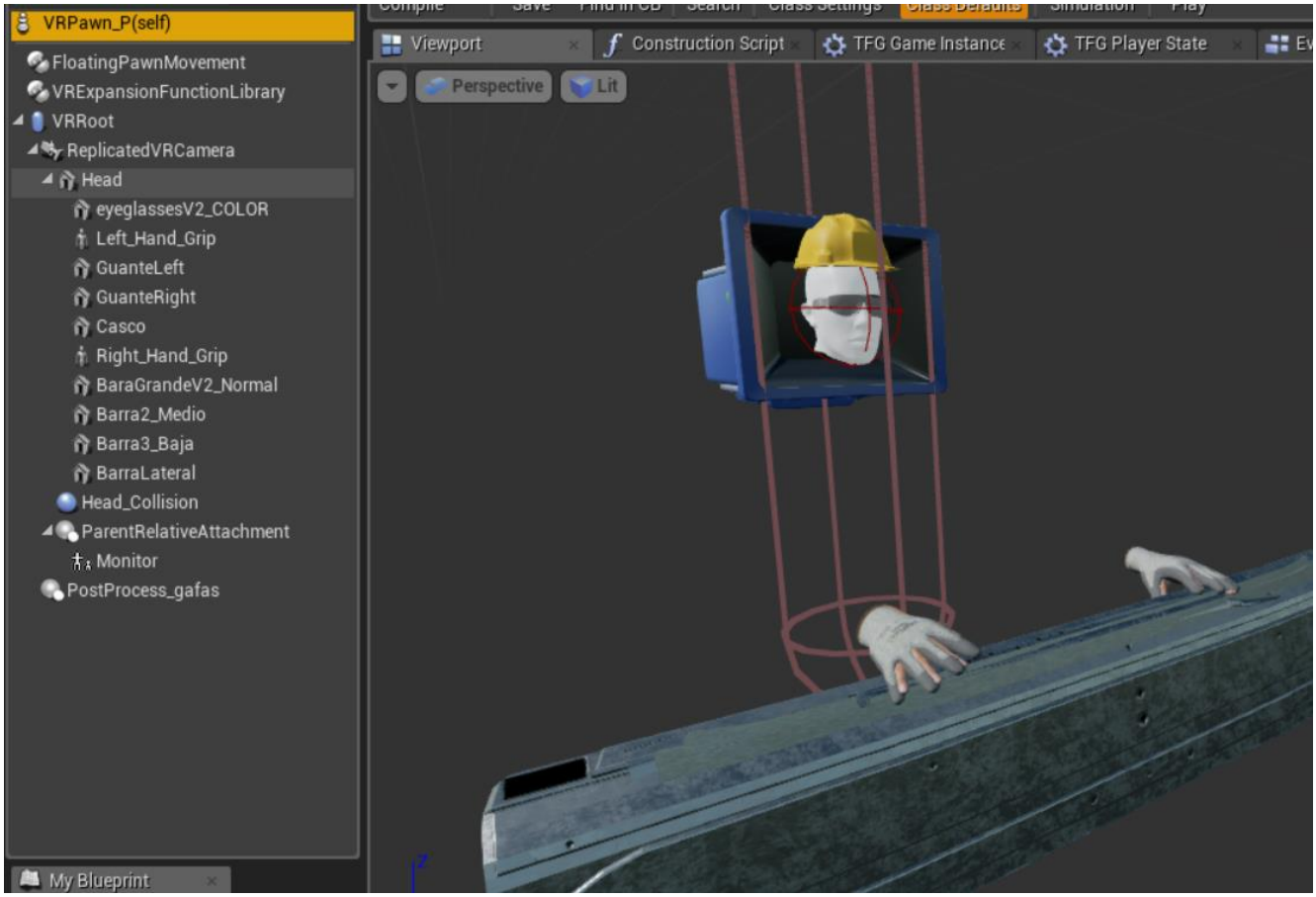


Figura 133: Componentes visuales del Pawn 'VRPawn_P'

7.4.3 Sujetar y colocar barras metálicas

El objetivo del proceso del escenario de montaje consiste en ir colocando unas piezas metálicas en un orden determinado en una máquina de soldadura de piezas, para conseguir esto, cuando el usuario pulsa el botón correspondiente a una barra, mientras lo mantenga pulsado sujetará la barra y podrá moverse con ella por el escenario, cuando deje de mantener pulsado entonces la barra se colocará en su lugar correspondiente en la máquina. El pawn tiene un componente visual de cada una de las barras pero que están ocultas, de modo que cuando se sujeta una barra se hacen visibles haciendo que parezca que el jugador mantiene agarrada una barra en concreto.

Cuando se coge una barra se informa al pawn del jugador que **comprueba si se está cogiendo la barra adecuada** en el orden de secuencia correcto y si no es la autoridad del juego (es el jugador), si es así, llama al pawn del servidor mediante un evento *Run on Server* que modifica el componente visual de la barra concreta dentro del pawn a través de un evento *Multicast*.

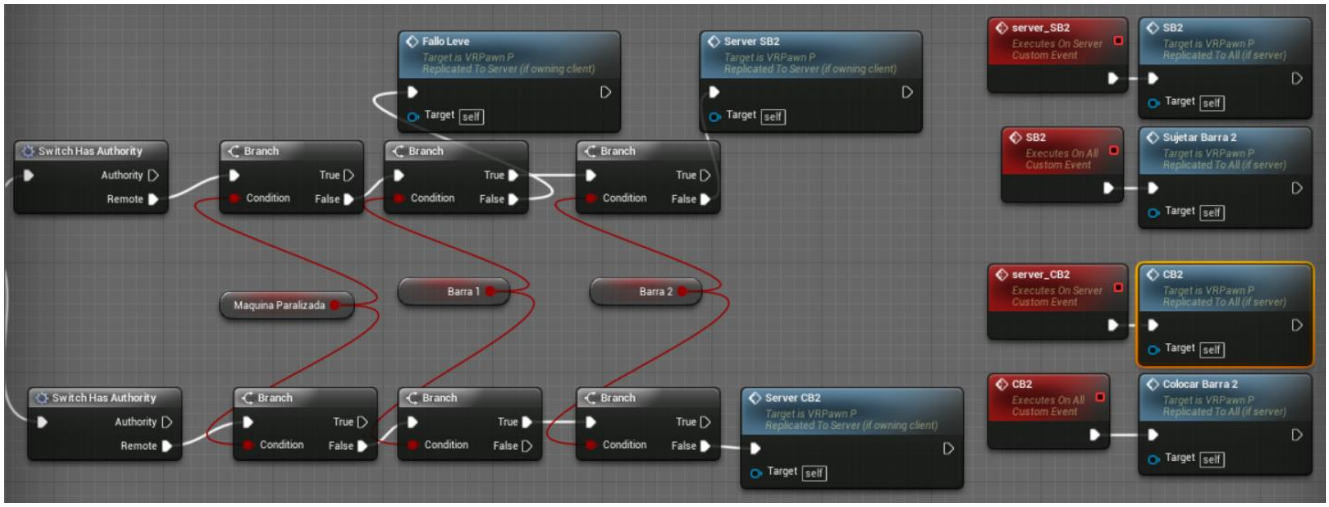


Figura 134: Comprobación de uso de piezas metálicas en el pawn 'VRPawn_P'

Además, se lanza un **evento que se ejecutará en el servidor** (Run on Server) que ejecuta un evento en el controlador *TFG_player_state* que dependiendo del tipo de barra **actualiza el panel de control del usuario manager** haciendo que se incremente el número de acciones de secuencia que puede verse debajo del minimapa del panel y, además, actualiza el estado del juego en el controlador *TFG_gameInstance*.

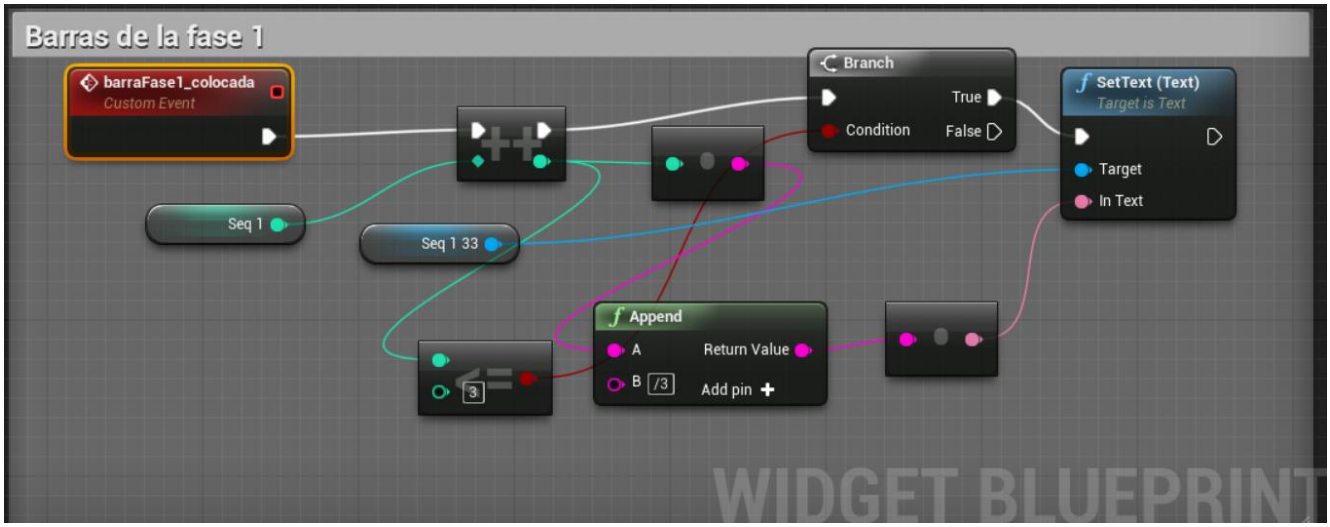


Figura 135: Actualización del estado de la secuencia en el panel de control del manager

Si el usuario intentara coger una barra en un orden incorrecto, entonces el pawn del jugador informa mediante un evento 'Run on Server' al controlador 'TFG_player_state' de que se ha cometido un **fallo leve**, de modo que este se comunica con la interfaz del panel de control en el manager actualizando los fallos leves del usuario y, además, con el controlador 'TFG_gameInstance' para actualizar el número de fallos leves del jugador.

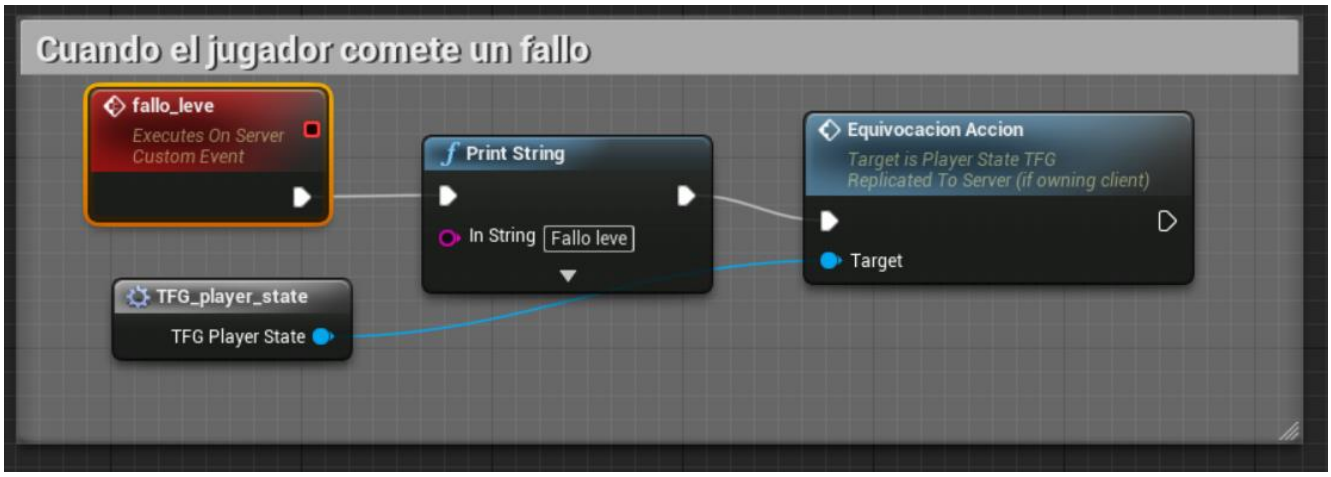


Figura 136: Llamada al evento 'EquivocacionAccion' del controlador TFG_player_state en el servidor

Si la máquina de soldadura estuviera paralizada por la acción del manager de *fallo en la cadena de producción* o por el atasque en el robot soldador, no se permite realizar ninguna otra acción (aparte de ponerse protecciones de seguridad) hasta que no se solvente el fallo por parte del jugador, esto se hace mediante la comprobación de una variable de control.

7.4.4 Cambiar de fase de proceso

Existen dos fases de proceso y en cada una de ellas deben ser colocadas en la máquina 3 barras en un orden determinado, cuando en una de estas fases se han colocado las tres barras correspondientes, entonces es necesario que el usuario realiza la acción de 'cambio de fase' que hace que la máquina gire y cambie de posición para continuar con la fase 2 del proceso.

Cuando un jugador pulsa el botón correspondiente al cambio de proceso en el orden de secuencia adecuado, el pawn del jugador **comprueba que el estado del juego se encuentra en el orden correcto**, es decir, si se encuentra en la fase 1 del proceso comprueba que se ha colocado la última barra de este y si se encuentra en la fase 2 comprueba que se haya colocado en la máquina la tercera y última barra correspondiente a este proceso, además comprueba si es el final del juego para restringir seguir realizando acciones al jugador. Cuando esto ocurre el *pawn* del jugador se comunica con el controlador *TFG_player_state* del servidor a través del evento '**actualizarEstadoJuego'** que recibe como **parámetro el tipo de acción del jugador**.

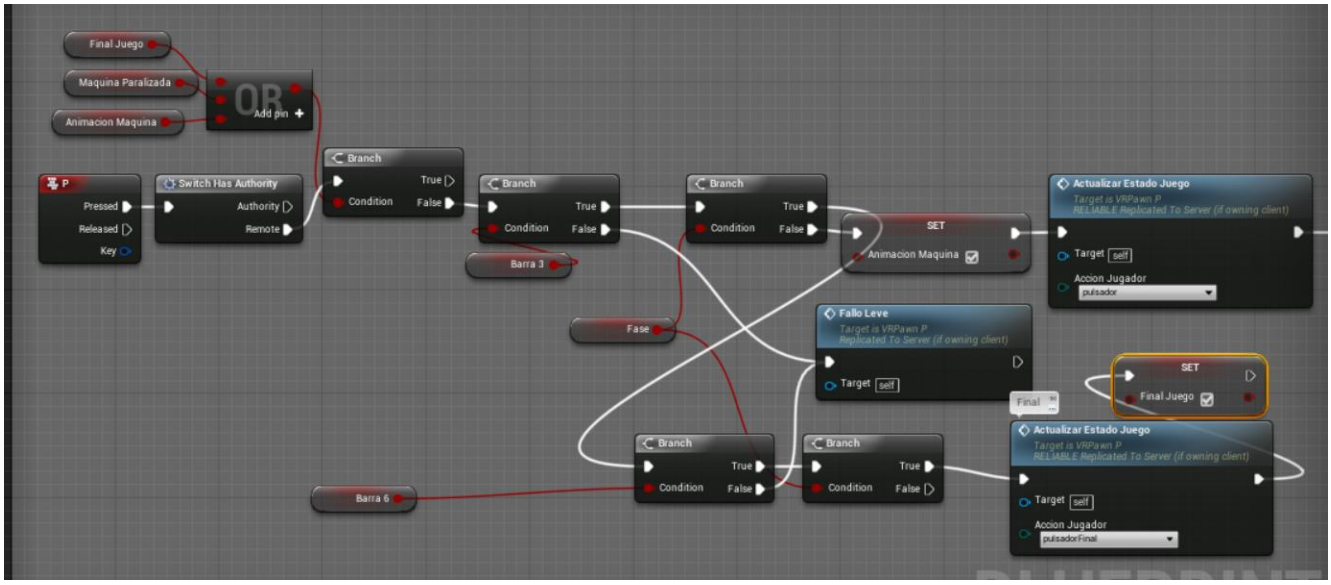


Figura 137: Código ejecutado en el Pawn cuando un jugador ejecuta el evento de cambio de fase

Este a su vez llama a un *event dispatcher* que será ligado a un evento local desde el *level blueprint* del escenario de fabricación haciendo que la máquina gire y se pase a la siguiente fase del proceso.

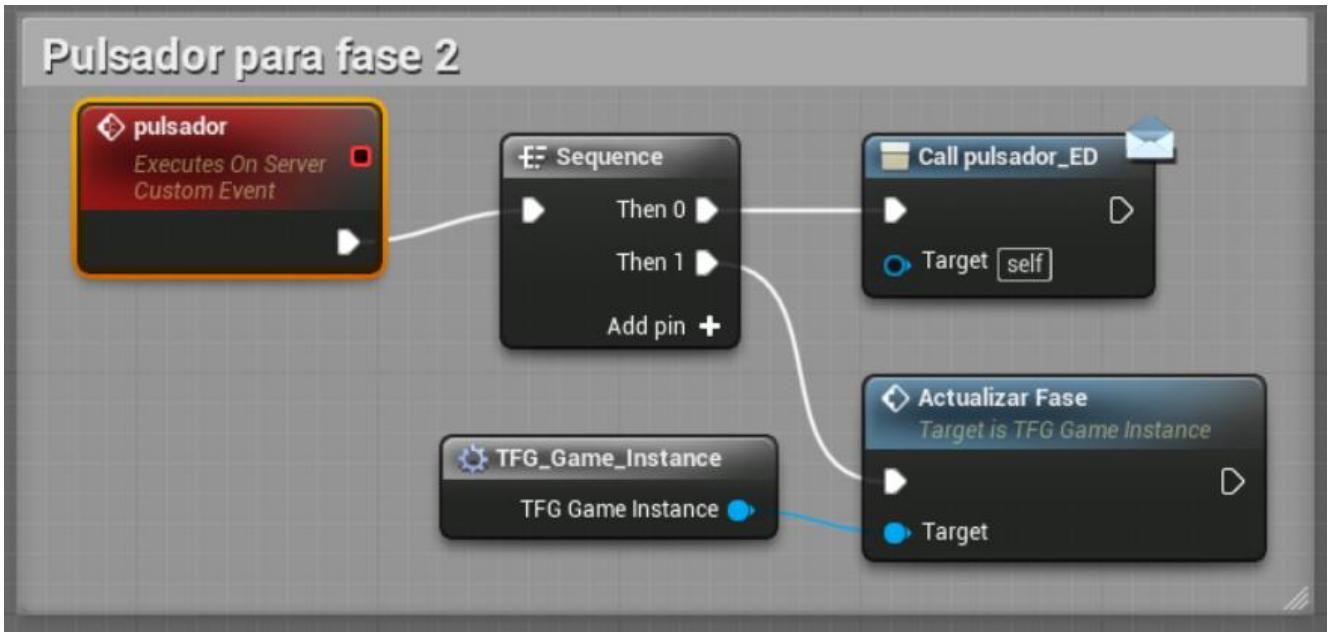


Figura 138: Actualización de la fase del escenario de fabricación en el controlador 'TFG_player_state'

Si el jugador se encuentra en la fase 2 del proceso y se pulsa el botón, se comunica de la misma forma con el controlador del servidor *TFG_player_state* y ahora llama a un event dispatcher distinto que se liga también desde el *level blueprint* del escenario de fabricación haciendo que se produzca la animación final de la máquina de soldadura y además el controlador genera los resultados de la partida y la guarda en base de datos como se explicaba en el apartado 7.2.7.8 **si el usuario manager confirma la acción** de guardar partida.

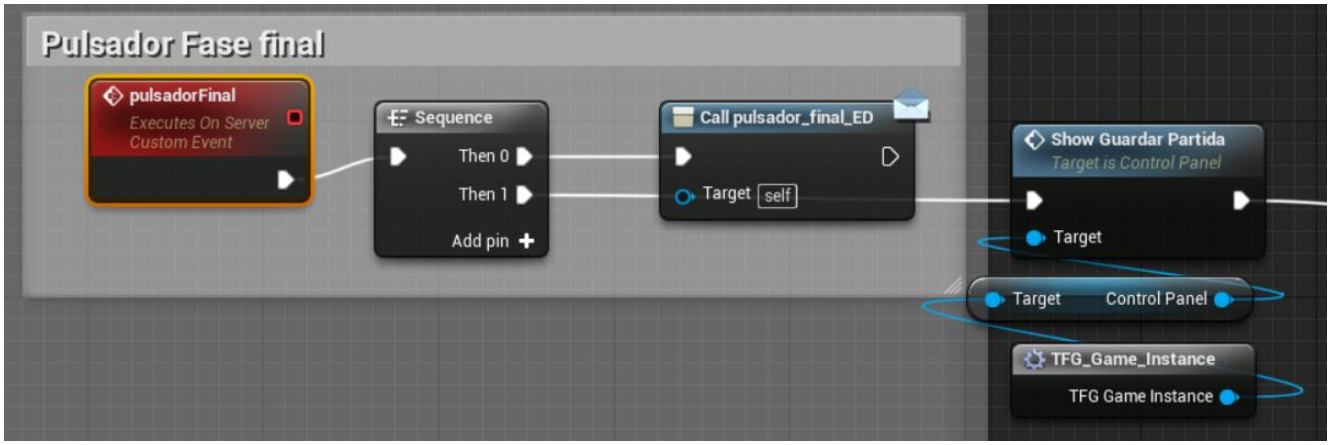


Figura 139: Llamada al event dispatcher 'pulsador_final_ED' cuando un usuario jugador finaliza la partida

7.4.5 Solventar atasque del robot soldador

Cuando un usuario manager ejecuta la acción de atascar el robot soldador, la vista se comunica con el controlador 'TFG_player_state' que llama a un event dispatcher que es ligado desde el level blueprint del escenario de fabricación mostrando el atasque del robot al jugador y además este se comunica con el pawn del jugador actualizando la **variable de control 'maquinaParalizada'** que impide la ejecución de otras acciones hasta que el jugador solvente el problema.

Para que el jugador desatasque el robot soldador y pueda seguir con el proceso de fabricación, debe pulsar el botón correcto que hace que esto ocurra. **El pawn comprueba que la máquina se encuentra atascada** (de lo contrario comunicará al controlador del servidor que contabilice un fallo leve) y si lo está, entonces llama al controlador *TFG_player_state* mediante un evento 'Run on Server' y este a su vez **actualiza el escenario del nivel** mediante un *event dispatcher* y oculta el modelo que indica el atasque del robot.

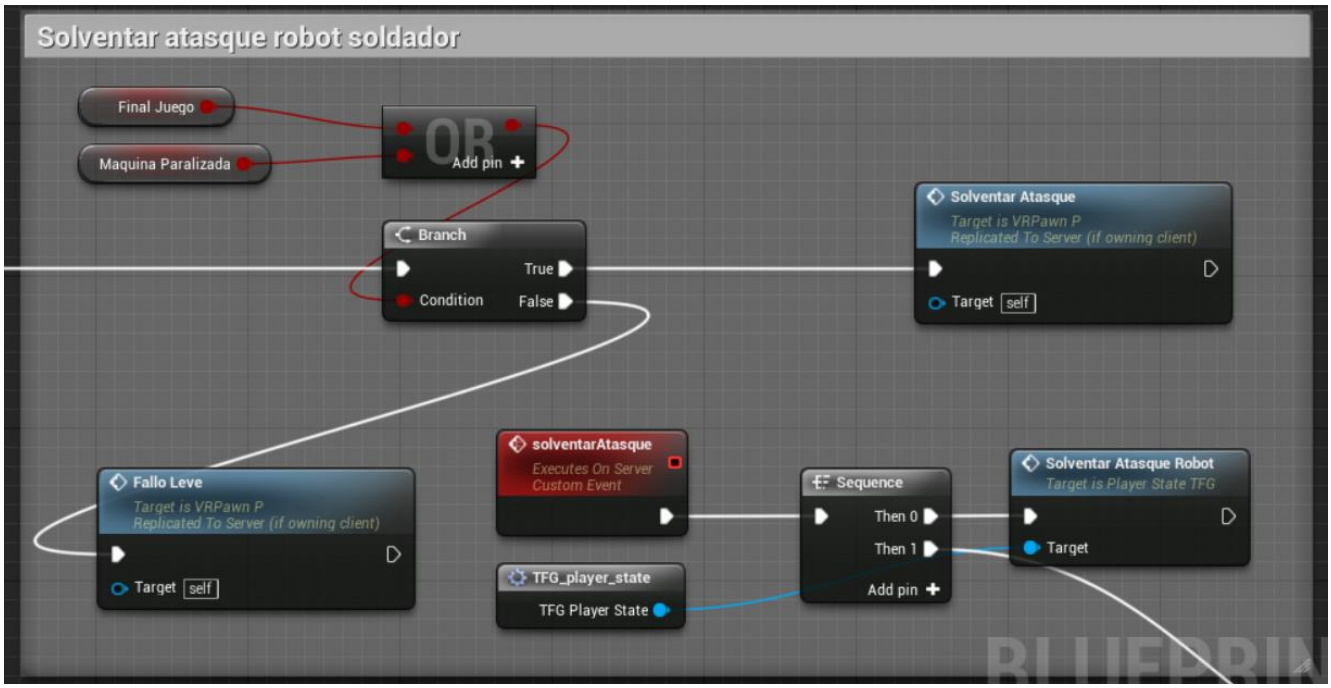


Figura 140: Código ejecutado en el Pawn del jugador cuando solventa el atasque en el robot soldador

Además, **actualiza el estado de la variable de control 'maquinaParalizada'** permitiendo de nuevo seguir con el proceso de fabricación y también lo comunica al controlador del servidor.

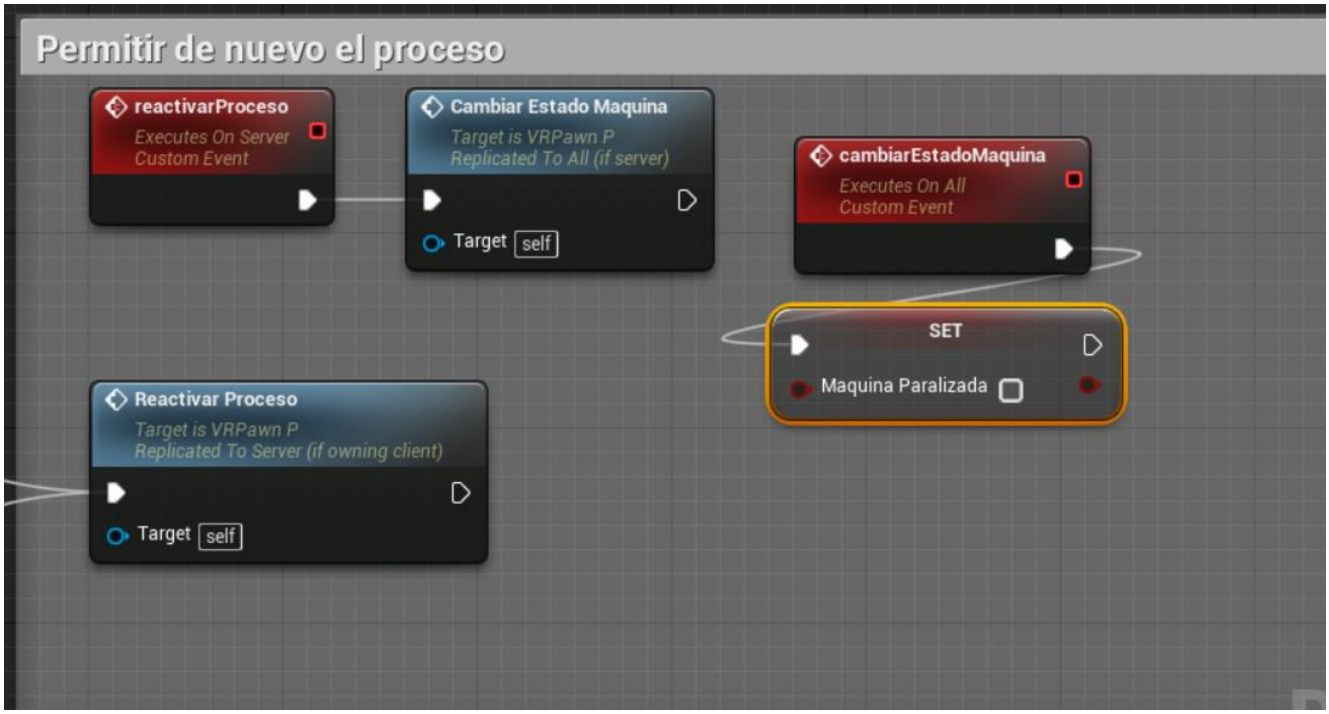


Figura 141: Evento de reactivación de la máquina cuando un usuario jugador solventa un problema con la máquina

7.4.6 Solventar fallo en la cadena de producción

Al igual que ocurre cuando un jugador intenta solventar un fallo de atasque en el robot soldador provocado por el usuario manager desde el panel de control, se crea una luz roja parpadeante en el escenario que se replica al jugador y que le indica el fallo en la cadena de producción.

Cuando el usuario pulsa el botón correspondiente que solventa el fallo en la cadena de producción, el pawn del jugador se comunica con el controlador del servidor 'TFG_player_state' que llama a un 'event dispatcher' que es ligado desde el escenario de fabricación y que deshabilita a luz roja parpadeante y **habilita de nuevo el proceso de fabricación**.

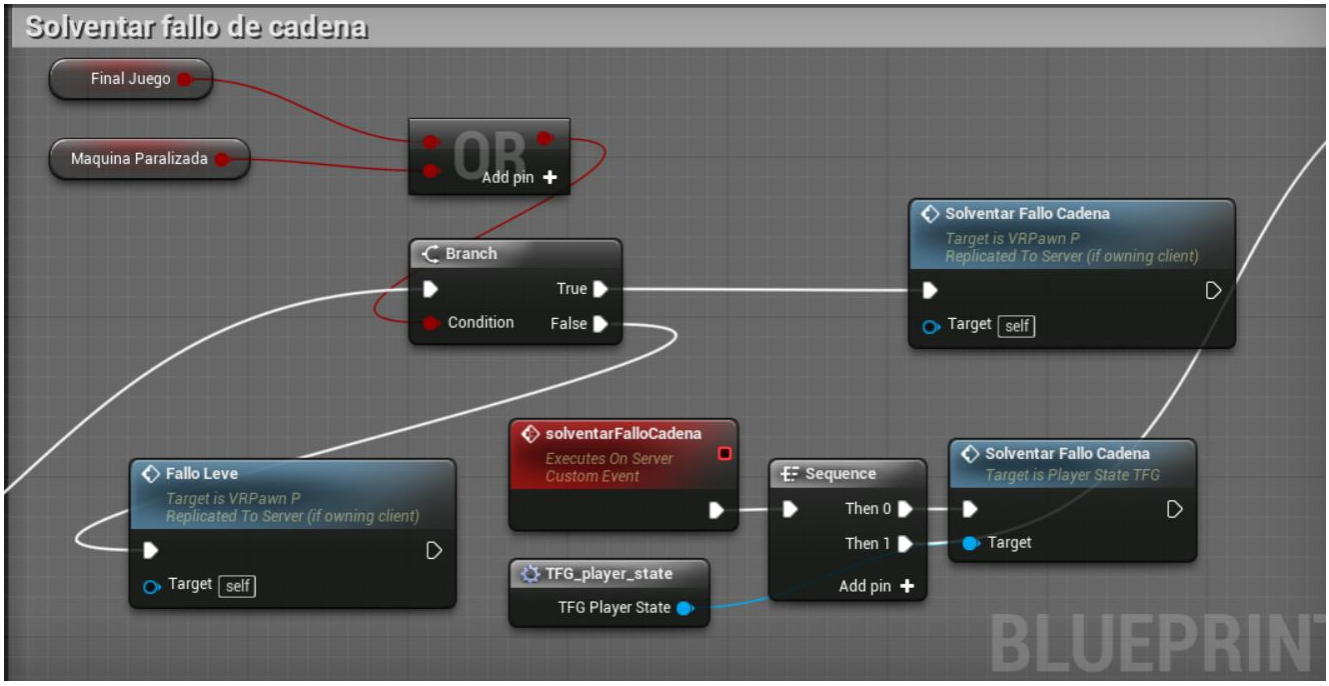


Figura 142: Código ejecutado cuando un jugador solventa un fallo en la cadena de producción

7.5 Fase 4: Creación de la base de datos y conexión desde la aplicación

Se ha elegido **MySQL** como lenguaje para la base de datos y el **plugin gratuito *VaRest*** [47] disponible desde el Marketplace del lanzador de Epic Games para las versiones 4.11 – 4.16 del motor de Unreal Engine 4. Este plugin **permite realizar conexiones y peticiones a una base de datos** utilizando únicamente blueprints dentro de un proyecto.

Se ha utilizado la aplicación gratuita ***XAMPP*** [48] que permite simular la ejecución de un servidor Apache junto con una base de datos de forma local, entre otros servicios, por lo que ha resultado muy útil a la hora de hacer pruebas con la aplicación.

En la aplicación, se crea una variable de tipo *String* denominada '**hostname'** en el controlador 'TFG_gameInstance' que contiene la **ruta de la localización de los scripts PHP** que se detallaban en el apartado anterior. Además, por cada script se existe una variable de tipo *String* que contiene el nombre del archivo PHP concreto dentro de esta ruta.

Cuando el controlador 'TFG_player_state necesita realizar una operación en la base de datos, este se **comunica con uno de los gestores del modelo** y le indica el tipo de petición a realizar como ya se ha explicado en el apartado 7.2.

El código de implementación para los **scripts PHP** son los siguientes:

- **tfg_db_connection**
  - Se encarga de crear la conexión a la base de datos proporcionando la dirección del servidor, el usuario de la base de datos y su contraseña y el nombre de la base de datos. El resto de scripts incluirán este script para conectarse a la base de datos.

```php
<?php

$servername = 'localhost';
$username = 'tomdoniphon';
$password = 'adminvrpanel';
$dbname = 'tfg_vrpanel';

// Creamos la conexión
$conn = new mysqli($servername, $username, $password,
$dbname);

// Se comprueba la conexión
if ($conn->connect_error) {
    echo(json_encode(array('status'=>"Fallo en la conexion:
" . $conn->connect_error)));
     die;
}

?>
```

Figura 143: Script PHP de conexión con la base de datos

- **login_manager**
  - Realiza una petición a la base de datos mediante un **dni y una contraseña** proporcionadas desde la aplicación a través de URL. Devuelve un objeto **JSON** que contiene el status de la operación, si es correcta devolverá ***status = 'OK'***.

```php
<?php

include_once "tfg_db_connection.php";

$dni = $_GET['dni'];
$password = $_GET['password'];

//Prepara la consulta
$stmt = $conn->prepare("SELECT * FROM managers WHERE dni = ?
AND password = ?");

//Liga las incognitas a los parametros pasados por URL
$stmt->bind_param("ss", $dni, $password);  // "s" means the
database

//Se ejecuta la consulta en la base de datos
$stmt->execute();

//Se liga el el resultado obtenido
$stmt->bind_result($row_dni, $row_pass, $nombre, $apellidos,
$correo, $admin);

if ($stmt->fetch()) {

//Se crea un objeto JSON que contiene los datos del manager
     echo json_encode(array('status'=>"OK", 'dni'=> $row_dni,
'password'=> $row_pass, 'nombre'=> $nombre, 'apellidos'=>
$apellidos, 'correo'=> $correo, 'admin'=> $admin));

     }

else
{echo  json_encode(array('status'=>'La informacion
proporcionada es incorrecta.'));
}


?>
```

Figura 144: Script PHP de identificación de usuarios manager en la base de datos

- **register_player**
  - **Registra un jugador** con los datos proporcionados desde la aplicación. Devuelve un objeto JSON con el status de la operación.

```
<?php

include_once "tfg_db_connection.php";

//Recupera los datos de la URL

$dni = $_GET['dni'];
$nombre= $_GET['nombre'];
$apellidos= $_GET['apellidos'];
$edad= $_GET['edad'];
$experiencia= $_GET['exp'];

//Prepara la consulta
if ($stmt = $conn->prepare("SELECT * FROM jugadores WHERE dni
= ? ")) //check if the account name is already taken
{
      $stmt->bind_param("s", $dni);

//Se ejecuta la consulta
      $stmt->execute();

      if ($stmt->fetch())  echo
json_encode(array('status'=>'No existe el DNI introducido'));


      else {

//Se registra el jugador en la base de datos

            $stmt = $conn->prepare("INSERT INTO jugadores VALUES
(?, ?, ?, ?, ?)");

            $stmt->bind_param("sssii", $dni, $nombre,
$apellidos, $edad, $experiencia);

            $stmt->execute();

            $stmt->close();

//Se devuelve un objeto JSON
            echo json_encode(array('status'=>'OK' ));

            }
}
?>
```

Figura 145: Script PHP de registro de jugadores en la base de datos

- **registrar_manager**
  - **Registra un usuario manager** con los datos proporcionados desde la aplicación. Devuelve un JSON con el status de la operación.

```
<?php

include_once "tfg_db_connection.php";

//Se recuperan los datos de la URL

$dni = $_GET['dni'];
$nombre= $_GET['nombre'];
$apellidos= $_GET['apellidos'];
$correo= $_GET['correo'];
$admin= $_GET['admin'];
$password= $_GET['password'];


if ($stmt = $conn->prepare("SELECT * FROM managers WHERE dni
= ? ")) //check if the account name is already taken
{
     $stmt->bind_param("s", $dni);  // "s" means the database
expects a string

//Se ejecuta la peticion
     $stmt->execute();

     if ($stmt->fetch())  echo
json_encode(array('status'=>'This dni is unavailable'));

     else {

//Se registra al manager en la abse de datos
          $stmt = $conn->prepare("INSERT INTO managers VALUES
(?, ?, ?, ?, ?, ?)");

          $stmt->bind_param("sssssi", $dni, $password,
$nombre, $apellidos, $correo, $admin);

          $stmt->execute();

          $stmt->close();

//Se crea un objeto JSON con el status
          echo json_encode(array('status'=>'OK' ));

          }
}
?>
```

Figura 146: Script PHP para el registro de managers en la base de datos

- **buscar_jugador**.
    - **Busca un jugador registrado en la base de datos** mediante un dni proporcionado por URL desde la aplicación. Devuelve un **objeto JSON** que será decodificado en la aplicación y creará una entidad de **tipo Jugador**.

```
<?php

include_once "tfg_db_connection.php";

//Se recuperan los datos de la URL
$dni = $_GET['dni'];

//Se prepara la consulta
$stmt = $conn->prepare("SELECT * FROM jugadores WHERE dni
= ?");

$stmt->bind_param("s", $dni);

//Se ejecuta la consulta
$stmt->execute();

$stmt->bind_result($row_dni, $row_nombre, $row_apellidos,
$row_edad, $row_exp);

if ($stmt->fetch()) {

//Se crea un objeto JSON con los datos del jugador
      echo json_encode(array('status'=>"OK", 'dni'=> $row_dni,
'nombre'=> $row_nombre, 'apellidos'=> $row_apellidos, 'edad'=>
$row_edad, 'exp'=> $row_exp));

      }

else
{echo  json_encode(array('status'=>'Los dni intorducido no
existe en la base de datos.
}


?>
```

Figura 147: Script PHP para la búsqueda de jugadores en la base de datos

- **guardar_partida**
  - Guarda una partida en la base de datos en función de los datos proporcionados desde la aplicación. Devuelve un objeto JSON con el status de la operación.

```
<?php

include_once "tfg_db_connection.php";

//Se recuperan los datos de la URL

$dnimanager = $_GET['dnimanager'];
$dnijugador= $_GET['dnijugador'];
$tiempo= $_GET['tiempo'];
$puntuacion= $_GET['puntuacion'];
$fecha= $_GET['fecha'];
$escenario= $_GET['escenario'];
$fallos= $_GET['fallos'];

//Se prepara la consulta
$stmt = $conn->prepare("INSERT INTO partidas VALUES
(NULL, ?, ?, ?, ?, ?, ?, ?)");

           $stmt->bind_param("ssiissi", $dnimanager,
$dnijugador, $tiempo, $puntuacion, $fecha, $escenario,
$fallos);

//Se ejecuta la consulta en la abse de datos
           $stmt->execute();

           $stmt->close();

//Como el id de la partida se genera mediante auto increment,
se devuelve el id generado la última partida guardada

$stmt = $conn->prepare("SELECT id FROM partidas ORDER BY id
DESC LIMIT 1");

$stmt->execute();

$stmt->bind_result($row_id);

if ($stmt->fetch()) {

     echo json_encode(array('status'=>"OK", 'id_partida'=>
$row_id));

}

?>
```

Figura 148: Script PHP de registro de partidas en la base de datos

- **registrar_evento**
  - Este script php se encarga de **registrar todos los eventos** del usuario manager ocurridos durante una partida y asociarlos a la partida concreta.

```
<?php

include_once "tfg_db_connection.php";

//Recupera los datos de la URL

$id_partida = $_GET['id_partida'];
$momento= $_GET['momento'];
$tipo= $_GET['tipo'];

//Prepara la consulta

            $stmt = $conn->prepare("INSERT INTO eventomanager
VALUES (NULL, ?, ?, ?)");

            $stmt->bind_param("iss", $id_partida, $momento,
$tipo);

//Se ejecuta la consulta en la base de datos

            $stmt->execute();

            $stmt->close();

            echo json_encode(array('status'=>'OK'));

?>
```

Figura 149: Script PHP de registrod e eventos del manager en la base de datos

7.6 Fase 5: Despliegue

Dado que se trata de una aplicación multijugador que tendrá funcionalidades diferenciadas para tanto para cliente como para servidor, se empaquetan **dos aplicaciones de escritorio** destinadas a funcionar en plataformas Windows de 64 bits.

Además, en esta etapa se lleva a cabo un proceso de **optimización del código y revisión del rendimiento** mediante herramientas de *profiling* [49] de CPU y GPU propias del motor de Unreal Engine .

7.6.1 Despliegue de la aplicación servidor

En los ajustes del proyecto se selecciona como mapa inicial al ejecutar la aplicación el nivel ‘Inicio_Server’ que ya comentamos en el apartado 7.1.1, de esta forma hacemos que la aplicación servidor inicie en un punto diferente a la aplicación cliente, consiguiendo diferenciar las dos aplicaciones de forma sencilla mediante la preparación previa de la lógica de la aplicación mediante niveles.

7.6.2 Despliegue de la aplicación cliente

Al igual que con la aplicación servidor, en los ajustes del proyecto seleccionamos como nivel de inicio el mapa '*Inicio_Jugador'*, de esta forma al iniciar la aplicación del jugador aparecerá por defecto en la sala de búsqueda de sesiones, mientras que la aplicación del servidor iniciará con las interfaces previas al panel de control.

8. PRUEBAS

En este capítulo se recogen las pruebas realizadas en el sistema para verificar que se cumplen correctamente todas las funcionalidades y objetivos del sistema. Haremos una división por casos de uso, es decir, se realiza una batería de pruebas por cada caso de uso identificado del sistema.

Las pruebas son de **caja negra** con el objetivo de comprobar que el sistema se comporta de la forma esperada en cada escenario de pruebas.

8.1Pruebas de funcionalidad

A continuación, se muestran las baterías de pruebas de cada caso de uso del sistema.

8.1.1 CU-01: Registrar manager

Prueba	Resultado esperado	Valoración
Rellenar correctamente los datos del formulario de registro	El sistema registra a un usuario manager en la base de datos	Validado
Registrar un usuario manager con un DNI ya existente	El sistema no registra al manager e informa al usuario	Validado
Registrar un usuario manager con algún campo vacío.	El sistema no registra al manager e informa del error al usuario	Validado
Registrar un usuario manager con algún campo erróneo	El sistema no registra al manager e informa del error al usuario	Validado

Tabla 30: Batería de pruebas para el caso de uso 'CU01 - Registrar manager'

8.1.2 CU-02: Login manager

Prueba	Resultado esperado	Valoración
Rellenar correctamente los datos del formulario de inicio de sesión	El usuario inicia la sesión en el sistema.	Validado
Identificarse con un DNI o contraseña inválidas	El sistema no puede identificar al usuario	Validado
Identificarse con algún campo vacío	El sistema no puede identificar al usuario	Validado
Identificarse con un dni que no pertenece a ningún usuario	El sistema no encuentra al usuario e informa del fallo	Validado
Comprobar que los datos devueltos por el script PHP son correctos	El sistema almacena una variable de tipo ‘Manager’ con los datos correctos	Validado

Tabla 31: Batería de pruebas del caso de uso 'CU02 - Login manager'

8.1.3 CU-03: Crear partida

Prueba	Resultado esperado	Valoración
Se introduce el dni de un jugador existente en la base de datos y se inicia una partida	El sistema crea una sesión de juego en red LAN	Validado
Se introduce un dni no válido	El sistema detecta el error, informa al usuario y la partida no se inicia	Validado
Se introduce un dni inexistente	El sistema no puede identificar al usuario y la partida no se inicia	Validado
Identificarse con un dni que no pertenece a ningún usuario	El sistema detecta el error, informa al usuario y la partida no se inicia	Validado
Se intenta iniciar una partida sin seleccionar un escenario de juego	El sistema informa del error y no se crea la sesión de juego	Validado

Tabla 32: Batería de pruebas del caso de uso 'CU03 – Crear partida'

8.1.4 CU-04: Registrar jugador

Prueba	Resultado esperado	Valoración
Rellenar correctamente los datos del formulario de registro de jugador	El sistema registra a un usuario jugador	Validado
Rellenar de forma incorrecta algún campo del formulario	El sistema no registra al usuario e informa al usuario	Validado
Dejar algún campo vacío del formulario	El sistema no registra al usuario e informa al usuario	Validado
Registrar a un jugador con dni ya existente	El sistema no registra al usuario e informa al usuario	Validado

Tabla 33: Batería de pruebas del caso de uso 'CU04 - Registrar jugador'

8.1.5 CU-05: Proyección de material

Prueba	Resultado esperado	Valoración
Se realiza la acción de proyectar material cuando el usuario no lleva gafas de protección en una partida	El sistema cambia la vista del usuario haciendo que pierda visibilidad, el servidor registra el evento y actualiza la partida	Validado
Se realiza la acción de proyectar material cuando el usuario lleva gafas de protección en una partida	El sistema informa al jugador de que ha evitado la proyección del material gracias a las gafas de protección y se ha registrado el evento	Validado
Se intenta realizar muchas veces seguidas la proyección de material	El sistema detecta el intento y no permite ejecutar más de 1 acción cada 2 segundos	Validado

Tabla 34: Batería de pruebas del caso de uso 'CU05 - Proyección de material'

8.1.6 CU-06: Atasque en el robot soldador

Prueba	Resultado esperado	Valoración
Se realiza la acción de atasque en el robot soldador en una partida	Se muestra el atasque en el escenario de juego y se replica al usuario jugador, el servidor registra el evento y actualiza la partida	Validado
Se realiza más de dos veces seguidas la acción de atasque en el robot soldador en una partida	El sistema únicamente muestra la primera acción e ignora el resto hasta que el jugador desactive el atasque	Validado

Tabla 35: Batería de pruebas del caso de uso 'CU06 - Atasque en el robot soldador'

8.1.7 CU-07: Fallo en cadena de producción

Prueba	Resultado esperado	Valoración
Se realiza la acción de fallo en la cadena de producción durante una partida	Se muestra en el escenario una luz roja parpadenate indicativa del fallo en cadena y se replica al usuario jugador, el servidor registra el evento y actualiza la partida	Validado
Se realiza más de una vez la acción de fallo en la cadena de producción durante una partida sin que el jugador resuelva el problema	El sistema únicamente muestra la primera acción e ignora el resto hasta que el jugador desactive el fallo de producción	Validado

Tabla 36: Batería de pruebas del caso de uso 'CU07 - Fallo en la cadena de producción'

8.1.8 CU-08: Corte por barra metálica

Prueba	Resultado esperado	Valoración
Se realiza la acción de corte por barra cuando el jugador sostiene una barra metálica sin protecciones	El sistema informa al jugador del corte provocado por la barra, el servidor registra el evento y actualiza la partida	Validado
Se realiza la acción de corte por barra cuando el jugador sostiene una barra metálica con protecciones	El sistema informa al jugador de que se ha protegido del corte provocado por la barra gracias a las protecciones	Validado
Se realiza más de una vez la acción de corte por barra durante una partida	El sistema únicamente muestra la primera acción e ignora el resto	Validado

Tabla 37: Batería de pruebas del caso de uso 'CU08 - Corte por barra metálica'

8.1.9 CU-09: Cambiar vista del minimapa

Prueba	Resultado esperado	Valoración
Se selecciona otra vista diferente a la actual	El sistema cambia la vista del minimapa a la cámara seleccionada	Validado
Se selecciona la misma cámara de visión actual en el minimapa	El sistema no realiza cambios y mantiene la misma zona de imagen del minimapa	Validado

Tabla 38: Batería de pruebas del caso de uso 'CU09 - Cambiar vista del minimapa'

8.1.10 CU-10: Consultar información del jugador

Prueba	Resultado esperado	Valoración
Se realiza la acción de ver la información del jugador durante una partida	El sistema muestra la información del jugador de forma correcta	Validado

Tabla 39: Batería de pruebas del caso de uso 'CU10 - Consultar información del jugador'

8.1.11 CU-11: Guardar resultados

Prueba	Resultado esperado	Valoración
Cuando finaliza una partida se confirma la acción de guardar los resultados en la base de datos	El sistema guarda la partida en la base de datos y todos los eventos del manager realizados durante esta	Validado
Cuando finaliza una partida se cancela el registro de la partida	El sistema no registra la partida en la base de datos y el juego finaliza	Validado

Tabla 40: Batería de pruebas del caso de uso 'CU11 - Guardar resultados'

8.1.12 CU-12: Terminar intento

Prueba	Resultado esperado	Valoración
Se termina el intento antes de la finalización de una partida	El sistema informa al jugador de lo ocurrido y se cancela la sesión de juego.	Validado

Tabla 41: Batería de pruebas del caso de uso 'CU12 - Terminar intento'

8.1.13 CU-13: Unirse a partida

Prueba	Resultado esperado	Valoración
El jugador intenta unirse a una partida cuando el servidor aún no ha creado partidas	El sistema no detecta partidas en curso y reanuda la búsqueda segundos después	Validado
El jugador intenta unirse a una sesión existente en la red	El sistema carga el mapa del servidor y une al jugador en la sesión del servidor	Validado

Tabla 42: Batería de pruebas del caso de uso 'CU13 - Unirse a partida'

8.1.14 CU-14: Colocar barra metálica

Prueba	Resultado esperado	Valoración
El jugador pulsa los botones apropiados para colocar una barra en la máquina	El sistema informa al servidor que coloca una barra en el escenario y lo replica al jugador	Validado
El jugador pulsa el botón de colocar barra cuando no procede	El sistema informa sobre el error al jugador y se actualizan los fallos del jugador	Validado
Se intenta colocar una barra cuando la máquina soldadora ha sido paralizada por una acción del manager	El sistema impide seguir con el proceso hasta que se solvente la acción.	Validado

Tabla 43: Batería de pruebas del caso de uso 'CU14 – Colocar barra metálica'

8.1.15 CU-15: Ponerse gafas de protección

Prueba	Resultado esperado	Valoración
El jugador pulsa los botones apropiados para colocarse las gafas de protección	El sistema informa al servidor que el usuario se ha colocado las gafas de protección e informa al jugador	Validado
El jugador pulsa el botón de colocarse las gafas de protección cuando ya las tiene puestas	El sistema ignora la acción e informa al jugador	Validado

Tabla 44: Batería de pruebas del caso de uso 'CU15 - Ponerse gafas de protección'

8.1.16 CU-16: Ponerse guantes de protección

Prueba	Resultado esperado	Valoración
El jugador pulsa los botones apropiados para colocarse los guantes de protección	El sistema informa al servidor que el usuario se ha colocado los guantes de protección e informa al jugador	Validado
El jugador pulsa el botón de colocarse los guantes de protección cuando ya las tiene puestas	El sistema ignora la acción e informa al jugador	Validado

Tabla 45: Batería de pruebas del caso de uso 'CU16 - Ponerse guantes de protección'

8.1.17 CU-17: Ponerse casco de protección

Prueba	Resultado esperado	Valoración
El jugador pulsa los botones apropiados para colocarse el casco de protección	El sistema informa al servidor que el usuario se ha colocado el casco de protección e informa al jugador	Validado
El jugador pulsa el botón de colocarse el casco de protección cuando ya las tiene puestas	El sistema ignora la acción e informa al jugador	Validado

Tabla 46: Batería de pruebas del caso de uso 'CU17 - Ponerse casco de protección'

8.1.18 CU-18: Desatascar robot soldador

Prueba	Resultado esperado	Valoración
El jugador desatasca el robot soldador cuando este está atascado	El sistema informa al servidor que desatasca el robot soldador	Validado
El jugador desatasca el robot soldado cuando no está atascado	El sistema informa del fallo al jugador y al servidor	Validado

Tabla 47: Batería de pruebas del caso de uso 'CU18 - Desatascar robot soldador'

8.1.19 CU-19: Solventar fallo de cadena

Prueba	Resultado esperado	Valoración
El jugador solventa el fallo de producción cuando este ha sido activado por el manager	El sistema informa al servidor que desactiva el parón de producción	Validado
El jugador solventa el fallo de producción cuando este no ha sido activado por el manager	El sistema informa del fallo al jugador y al servidor	Validado

Tabla 48: Batería de pruebas del caso de uso 'CU19 - Solventar fallo de cadena'

8.1.20 CU-20: Cambio de fase

Prueba	Resultado esperado	Valoración
El jugador pulsa el botón de cambio de fase cuando no procede	El sistema contabiliza la acción como un fallo leve e informa al jugador	Validado
El jugador pulsa el botón de cambio de fase cuando procede	El sistema ejecuta la animación correspondiente de la máquina soldadora y actualiza el estado del juego	Validado

Tabla 49: Batería de pruebas del caso de uso 'CU20 - Cambio de fase'

9. CONCLUSIONES Y TRABAJO FUTURO

9.1 Objetivos de la aplicación

El desarrollo de la aplicación finalizó de forma correcta cumpliendo con los objetivos de la aplicación, estos eran los de obtener una versión funcional de un panel de control que monitorizara a un usuario en tiempo real dentro de un entorno virtual y partiendo de un software previamente desarrollado.

Además de eso, también se ha realizado la implementación y conexión de la aplicación con una base de datos que sirve para almacenar todos los datos relevantes de la aplicación, haciendo que pueda utilizarse en un entorno real para después realizar diversos análisis de datos sobre los distintos usuarios que simulan el desarrollo de una actividad profesional en este entorno virtual.

El desarrollo de esta aplicación abre muchas posibilidades en diversos ámbitos y sectores profesionales en los que se requiera la monitorización de usuarios en diferentes tipos de entornos en los que no es factible realizarlo en la vida real por diferentes razones, como por ejemplo para la concienciación sobre riesgos laborales o la formación en entornos peligroso, o incluso para la monitorización de tareas de rehabilitación para pacientes en hospitales y centros de salud y que además estos puedan realizar los ejercicios jugando o incluso viajando a otros mundos y entornos gracias a la realidad virtual.

9.2 Trabajo futuro

En el futuro podrían incluirse nuevas funcionalidades como:

- Más escenarios de formación.
- Añadir mejoras gráficas a las interfaces.
- Ampliar el alcance del panel de control desarrollado haciendo que puedan consultarse acciones de la base de datos directamente desde las interfaces del manager.
- Permitir la modificación de la localización de los archivos de script y de hostname para que sean configurables.
- Buscar e identificar sesiones en red, no necesariamente redes LAN.
- Incluir análisis estadísticos basándose en los resultados obtenidos desde la aplicación.

Bibliografía

Libros

[1] Alireza Tavakkoli. *Game Development and Simulation with Unreal Technology,* 7 oct 2015

[2] Mitch McCaffrey. *Unreal Engine VR Cookbook: Developing Virtual Reality with UE4*, 2 feb 2017

[3] Dan Pilone*. UML 2.0 Pocket Reference*, 24 mar 2006

[4] David Lightbown. Designing the User Experience of Game Development Tools, David Lightbown, 6 mar 2015

Referencias

[5] Características y producto 'leap motion', disponible desde: https://www.leapmotion.com/ (última consulta: 08/07/2017)

[6] Tomasz Mazuryk and Michael Gervautz, *Virtual Reality: History, Applications, Technology and Future*

https://www.cg.tuwien.ac.at/research/publications/1996/mazuryk-1996-VRH/TR-186-2-96-06Paper.pdf (última consulta: 08/07/2017)

[7] *How Does Virtual Reality Work?*

http://www.marxentlabs.com/how-does-virtual-reality-work-part-i-of-2/, AR Blog,

(última consulta: 08/07/2017)

[8] Wareable, *How does VR actually work?*

https://www.wareable.com/vr/how-does-vr-work-explained *(última consulta: 08/07/2017)*

[9] PopularScience, How It Works: The Oculus Rift. *(última consulta: 08/07/2017)*

http://www.popsci.com/oculus-rift-how-it-works#page-2

[10] XinReality, Constellation. *(última consulta: 08/07/2017)*

https://xinreality.com/wiki/Constellation

[11] Gadget blog, Here's EXACTLY How The HTC Vive Works. *(última consulta: 08/07/2017)*

https://www.gadgetdaily.xyz/heres-exactly-how-the-htc-vive-works/

[12] SteamVR Quick Start, Unreal Engine 4 – SteamVR Development. *(última consulta: 08/07/2017)*

https://docs.unrealengine.com/latest/INT/Platforms/SteamVR/QuickStart/

[13] Unity, *Manual de Unity*

https://docs.unity3d.com/es/current/Manual/ *(última consulta: 08/07/2017)*

[14] Ejemplos de proyectos hechos con Unity, Documentación Unity. (última consulta: 08/07/2017)

https://madewith.unity.com/

[15] Documentación sobre elementos del framework del motor, Unreal Engine 4 Documentation (última consulta: 08/07/2017)

https://www.unrealengine.com/features

[16] Documentación sobre blueprints, Unreal Engine 4 Documentation (última consulta: 08/07/2017)

https://docs.unrealengine.com/latest/INT/Engine/Blueprints/

[17] Página de registro para la descarga del lanzador de Games https://accounts.unrealengine.com/register?lang=en_US&redirectUrl=https%3A%2F%2Fwww.unrealengine.com%2Fdownload%3Fdismiss%3Dhttps%253A%252F%252Fwww.unrealengine.com%252F&client_id=43e2dea89b054198a703f6199bee6d5b&noHostRedirect=true

[18] Requisitos de hardware y software del lanzador de Epic Games (última consulta: 08/07/2017)

http://help.epicgames.com/customer/en/portal/articles/2105922-epic-games-launcher-system-requirem

[19] Microsoft Visual Studio oficial site (última consulta: 08/07/2015)

https://www.visualstudio.com/es/?rr=https%3A%2F%2Fwww.bing.com%2F

[20] Funcionamiento del level editor, Unreal Engine 4 Documentation (última consulta: 08/07/2017)

https://docs.unrealengine.com/latest/INT/Engine/UI/LevelEditor/index.html

[21] Engine from 4.0 to 4.16 release notes, Unreal Engine 4 Documentation (última consulta: 08/07/2017)

https://docs.unrealengine.com/latest/INT/Support/Builds/index.html

[22] Level Editor Toolbar, Unreal Engine 4 Documentation (última consulta: 08/07/2017)

https://docs.unrealengine.com/latest/INT/Engine/UI/LevelEditor/Toolbar/index.html

[23] Working of levels, Unreal Engine 4 Documentation (última consulta: 08/07/2017)

https://docs.unrealengine.com/latest/INT/Engine/Levels/

[24] Actors and Geometry, Unreal Engine 4 Documentation (última consulta: 08/07/2017)

https://docs.unrealengine.com/latest/INT/Engine/Actors/index.html

[25] Level blueprints, Unreal Engine 4 Documentation (última consulta: 08/07/2017)

https://docs.unrealengine.com/latest/INT/Engine/Blueprints/UserGuide/Types/LevelBlueprint/

[26] Events, Unreal Engine 4 Documentation (última consulta: 08/07/2017)

https://docs.unrealengine.com/latest/INT/Engine/Blueprints/UserGuide/Events/index.html

[27] Inputs, Unreal Engine 4 Documentation (última consulta: 08/07/2017)

https://docs.unrealengine.com/latest/INT/Gameplay/Input/

[28] Build process in UE4, Unreal Engine 4 Documentation (última consulta: 08/07/2017)

https://docs.unrealengine.com/latest/INT/Engine/QuickStart/6/

[29] Plugins, Unreal Engine 4 Documentation (última consulta: 08/07/2017)

https://docs.unrealengine.com/latest/INT/Programming/Plugins/index.html

[30] Flujo de juego en UE4, Unreal Engine 4 Gameplay Framework (última consulta: 08/07/2017)

https://docs.unrealengine.com/latest/INT/Gameplay/Framework/GameFlow/index.html

[31] Game Mode and Game State, Unreal Engine 4 Gameplay Framework (última consulta: 08/07/2017)

https://docs.unrealengine.com/latest/INT/Gameplay/Framework/GameMode/index.html

[32] Elementos básicos del framework, Unreal Engine 4 Documentation (última consulta: 08/07/2017)

https://docs.unrealengine.com/latest/INT/Gameplay/Framework/index.html

[33] Widget blueprints, Unreal Engine 4 UMG UI Designer User Guide (última consulta: 08/07/2017)

https://docs.unrealengine.com/latest/INT/Engine/UMG/UserGuide/WidgetBlueprints/

[34] Event dispatchers, Unreal Engine 4 Blueprints Visual Scripting (última consulta: 08/07/2017)

https://docs.unrealengine.com/latest/INT/Engine/Blueprints/UserGuide/EventDispatcher/

[35] Nodos de control de flujo, Unreal Engine 4 Basic Scripting (última consulta: 08/07/2017)

https://docs.unrealengine.com/latest/INT/Engine/Blueprints/UserGuide/FlowControl/

[36] Inicio al multijugador en UE4, Networking and Multiplayer (última consulta: 08/07/2017)

https://docs.unrealengine.com/latest/INT/Gameplay/Networking/Overview/index.html

[37] RPCs, Unreal Engine 4 – Actor Replication (última consulta: 08/07/2017)

https://docs.unrealengine.com/latest/INT/Gameplay/Networking/Actors/RPCs/

[38] Actor Replication, Unreal Engine 4 - Networking and Multiplayer (última consulta: 08/07/2017)

https://docs.unrealengine.com/latest/INT/Gameplay/Networking/Actors/index.html

[39] BBC. Virtual Reality: The hyper, the problems and the promise. (última consulta: 08/07/2017)

http://www.bbc.com/future/story/20160729-virtual-reality-the-hype-the-problems-and-the-promise

[40] Website oficial de HTC VIVE (última consulta: 08/07/2017)

https://www.vive.com/us/ready/

[41] HowStuffWorks, *How Virtual Reality Works* (última consulta: 08/07/2017)

http://electronics.howstuffworks.com/gadgets/other-gadgets/virtual-reality.htm

[42] Laguna, Miguel Ángel. "Tema 2: Modelado de requisitos", "Tema3: Casos de uso, "Tema 5: Modelado del Dominio" (Tema 5), Asignatura de Modelado de Sistemas Software, Curso 2014-2015, Universidad de Valladolid.

[43] De la Fuente, Pablo. "Tema 2: Planificación y Control de Proyecto Software", "Tema 3: Modelo de proceso de desarrollo", Planificación y Gestión de Proyectos de Software curso 2015-2016, Universidad de Valladolid.

[44] Pavón Mestras, Juan. "Estructura de las Aplicaciones Orientadas a Objetos El patrón Modelo-Vista-Controlador (MVC)", Universidad Complutense de Madrid, 2008 - 2009.

[45] Blueprint networking tutorials, Unreal Engine 4 Official blog (última consulta: 08/07/2017)

https://www.unrealengine.com/en-US/blog/blueprint-networking-tutorials

[46] Unreal Engine API Reference (última consulta: 08/07/2017)

https://docs.unrealengine.com/latest/INT/API/

[47] Vladimir Alyamkin, VaRest Plugin (última consulta: 08/07/2017)
https://www.unrealengine.com/marketplace/varest-plugin

[48] Apache XAMPP (última consulta: 08/07/2017)

https://www.apachefriends.org/es/index.html

[49] Performance and Profiling, Unreal Engine – Engine Features (última consulta: 08/07/2017)

https://docs.unrealengine.com/latest/INT/Engine/Performance/index.html

Anexos

Manual de usuario

M.1 Descripción de la aplicación

Este software está formado por dos ejecutables denominados ‘VRPanel-Manager’ y ‘VRPanel-Player’ que actúan com aplicaciones separadas y que están ideadas para ejecutarse cada una de ellas en un dispositivo diferente, encontrándose ambos dispositivos conectados a una misma red.

M.1.1 VRPanel-Manager

Esta aplicación es capaz de controlar la partida de un jugador que utilice la aplicación en otro dispositivo que ejecute la aplicación ‘VRPanel-Player’ y que se encuentre conectado a la misma red. Permite además llevar un registro de todos los usuarios del sistema y las partidas llevadas a cabo a través de una base de datos local especificada en el apartado XX de este manual. Esta aplicación está pensada para ejecutarse como una aplicación tradicional de escritorio, por lo que se requieren los controladores habituales de un dispositivo PC.

El usuario que utilice esta aplicación deberá estar registrado previamente en la base de datos local para acceder a las funcionalidades que provee la aplicación, pudiendo registrar posteriormente a un usuario jugador (utilizará ejecutable ‘VRPanel-Player’ en otro dispositivo conectado vía red) o a un usuario manager si se tienen permisos de administración. También pueden crearse sesiones de juego en la red introduciendo para ello un dni válido de un usuario jugador registrado en la base de datos.

Cuando se crea una sesión de juego en la que otro usuario abre el ejecutable ‘VRPanel-Player’ y además este se encuentra conectado a la misma red, se abrirá un panel de control que permitirá monitorizar la actividad del jugador y registrar los datos relevantes de la sesión de juego.

M.1.2 VRPanel-Player

Esta aplicación permite a un usuario simular un proceso de fabricación en un escenario de montaje de carrocería de vehículos a través del uso de unas gafas de Realidad Virtual HTC VIVE y de un controlador adicional como un teclado o un mando configurable y compatible con Windows 10.

El dispositivo en que se ejecute la aplicación deberá estar conectado a una red en la que se encuentre otro dispositivo en el que esté funcionando el ejecutable ‘VRPanel-Manager’ y este haya creado una sesión de juego, sin esto el usuario de este ejecutable únicamente podrá estar en una ‘sala de espera’ hasta que se encuentre una sesión de juego activa.

El jugador deberá realizar un proceso de fabricación en el que deberá colocar una serie de objetos y realizar una serie de acciones en un orden y secuencia determinadas para obtener la mayor puntuación posible dentro del juego.

M.1.3 Información adicional

La estructura de la base de datos necesaria para que la aplicación funcione correctamente va incluida junto con el manual de instalación de este trabajo.

Si usted experimenta algún tipo de mareo o malestar durante la utilización de las gafas de realidad virtual es muy recomendable que deje de utilizarlas de inmediato.

M.2 Requisitos

A continuación, se especifican los requisitos de software necesarios y los requisitos de hardware mínimos y recomendados para ejecutar los ejecutables.

M.2.1 Requisitos de software

- Sistema operativo Windows 10.
- DirectX 12.
- Steam VR.
- HTC Vive software.
- Steam Controller software (opcional).
- Base de datos local.

M.2.2 Requisitos mínimos de hardware

- Procesador: Intel Core i5-6600.
- GPU: Nvidia GeForce GTX 970 o equivalente.
- Memoria RAM: 8GB DDR4
- Disco duro: Al menos 1,5 GB de espacio libre para cada aplicación (manager y jugador).
- Capacidad de conexión a la red.
- Gafas HTC VIVE (ejecutable ‘VRPanel-Player’)
- Controlador de PC (teclado o mando)

M.2.3 Requisitos recomendados de hardware

- Procesador: Intel Core i7-6700k.
- GPU: Nvidia GeForce GTX 1070 o equivalente.
- Memoria RAM: 16GB DDR4
- Disco duro: Al menos 1,5 GB de espacio libre para cada aplicación (manager y jugador).
- Capacidad de conexión a la red.
- Gafas HTC VIVE (ejecutable ‘VRPanel-Player’)
- Controlador de PC (teclado o mando)

M.3 Detalles técnicos

Se necesitan 2 equipos que cumplan los requisitos mínimos especificados más arriba y, además se necesita una conexión de red en la que ambos dispositivos estén conectados. Únicamente se permite una partida de juego activa simultáneamente por red.

El dispositivo que utilice la aplicación en el que se desarrolla el proceso de fabricación con las gafas de realidad virtual y se deberá disponer con un espacio mínimo de 1,5 x 2 metros para un uso cómodo de la aplicación.

Puede utilizarse un mando de PC configurable como *Steam controller* para configurar los accesos del jugador que realiza el proceso de fabricación.

Es muy recomendable desactivar el antivirus si este bloquea o elimina las aplicaciones de modo automático ya que se puede perder el ejecutable de la aplicación.

M.4 Funcionalidades del manager – VRPanel-Manager

A continuación, se explica cada una de las pantallas e interfaces con las que cuenta el ejecutable de la aplicación 'VRPanel-Manager'.

Panel de login

Se trata de la pantalla inicial de la aplicación y requiere la identificación por parte del usuario. La aplicación buscará en una base de datos local un usuario manager con las credenciales introducidas (los detalles de la base de datos se encuentran en el manual de instalación):

- DNI
- Password

Si los datos introducidos no son correctos o no se encuentra a un usuario con esos datos en la base de datos, entonces se mostrará un mensaje de error indicando el mensaje específico del error.

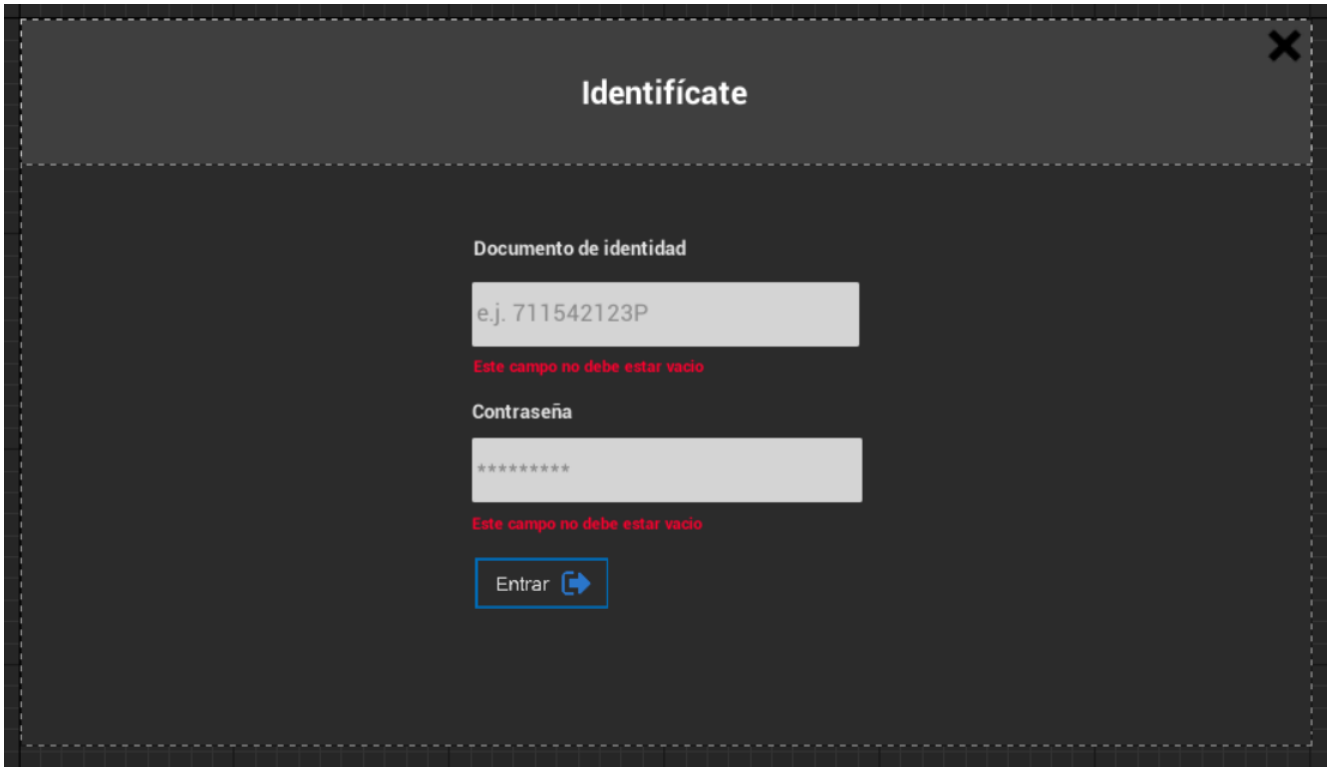


Figura 150: Panel de login

M.4.1 Interfaz de menú principal

Una vez nos hemos identificado en la interfaz anterior, desde esta pantalla podremos acceder a 3 nuevas interfaces, la segunda de ellas nos permite registrar a un usuario jugador, la tercera de ellas nos permite registrar a un usuario manager mientras que la opción primera nos permite crear una partida. También podemos salir de la aplicación pulsando el botón de 'SALIR'

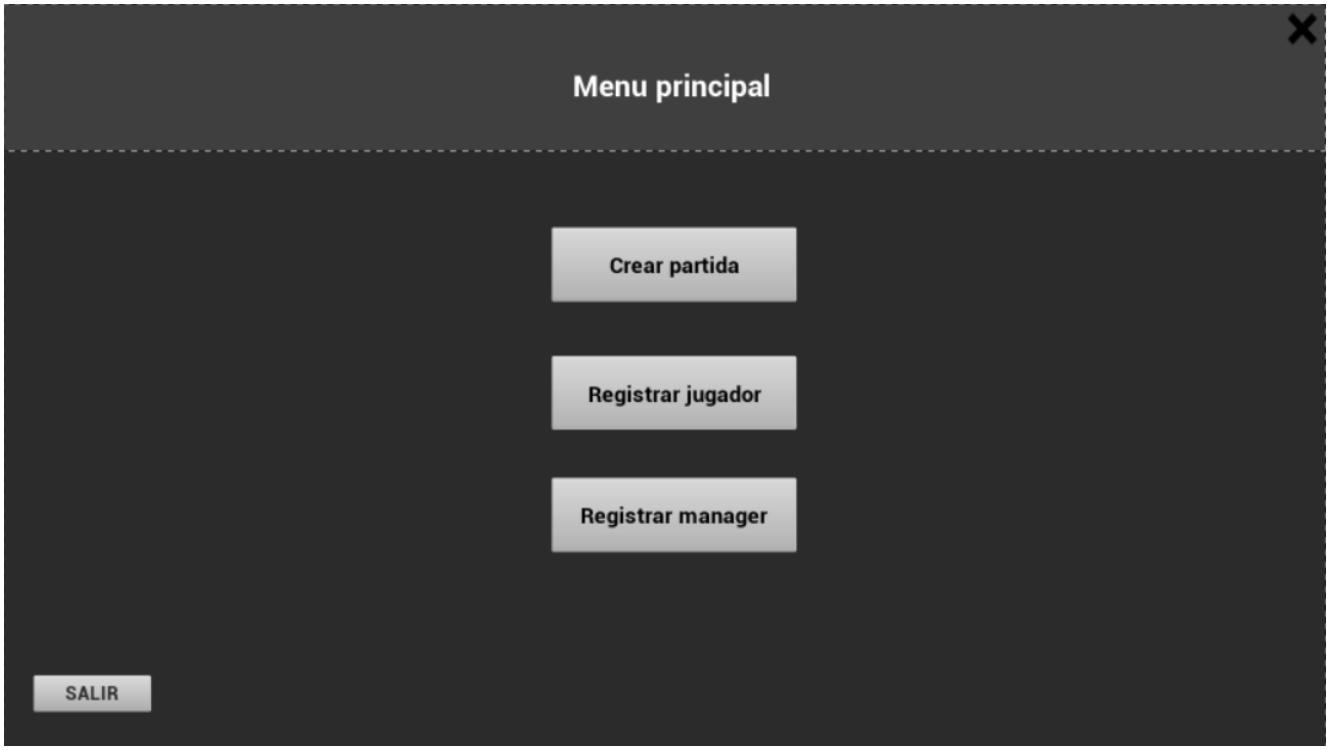


Figura 151: Menú principal

M.4.2 Interfaz de registro de managers

Desde esta interfaz podremos registrar a un usuario manager en la base de datos, para ello deberemos introducir los siguientes datos:

- **Dni** – Únicamente 9 caracteres permitidos
- **Nombre** – Entre 3 y 15 caracteres
- **Apellidos** – Entre 3 y 25 caracteres
- **Correo** – Entre 3 y 25 caracteres
- **Admin** – Si tiene permisos de administración

Una vez rellenados los campos del formulario de registro, si algunos de los datos no son válidos o ya existe un usuario manager con el dni introducido se nos mostrará el error en la pantalla, de lo contrario se registrará el manager en la base de datos generándose una contraseña formada por los tres primeros caracteres del nombre y los tres primeros de los apellidos introducidos.

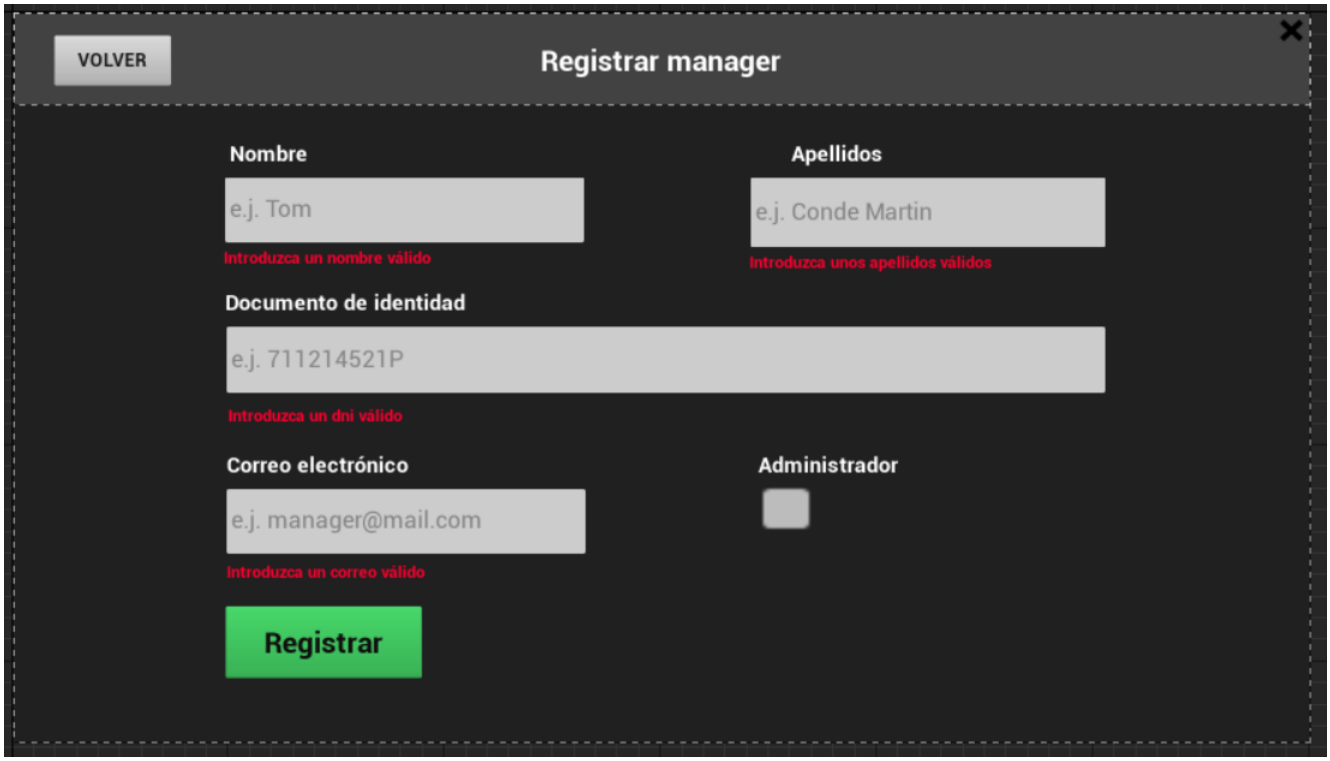


Figura 152: Interfaz de registro de managers

Cuando se ha registrado un usuario manager con éxito, se volverá a la pantalla del menú principal con un mensaje indicándonos que el registro se ha completado con éxito. Este mensaje desaparece a los 2 segundos de haberse activado.

Figura 153: Cuadro de información cuando se registra a un usuario (manager o jugador)

M.4.3 Interfaz de registro de jugadores

Desde esta interfaz podremos registrar a un usuario jugador (que utilizará la aplicación VRPanel-Player y las gafas de realidad virtual) aunque no tengamos permisos de administración. Para completar el registro debemos introducir los siguientes datos:

- **Dni** – Únicamente 9 caracteres permitidos.
- **Nombre** – Entre 3 y 15 caracteres.
- **Apellidos** – Entre 3 y 25 caracteres.
- **Edad** – Desde 18 hasta 65 años.
- **Experiencia previa** – Indicar si el jugador tiene experiencia previa.

Al igual que ocurría con el registro de managers, si alguno de los campos introducidos no es válido o ya existe un jugador con el dni introducido, entonces se nos informa del error a través de esta pantalla. Si el registro se ha producido de forma satisfactoria, entonces volveremos a la pantalla del menú principal donde se nos muestra un mensaje de éxito informándonos del registro.

Si se pulsara el botón de 'VOLVER' volvería a mostrarse la interfaz del menú principal.

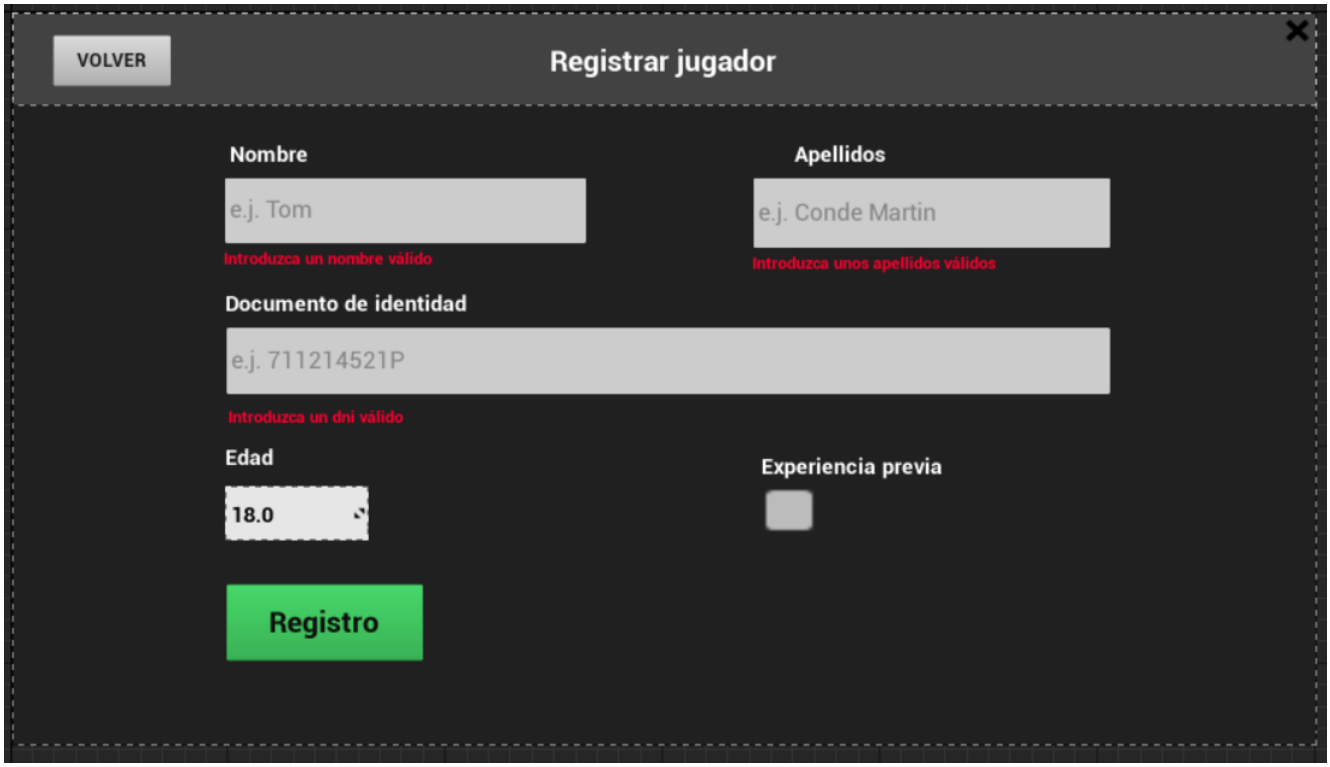


Figura 154: Interfaz de registro de jugadores

M.4.4 Interfaz de creación de partidas

Esta interfaz es accesible desde el menú principal y permite crear una sesión de juego activa. Para hacerlo deberemos estar conectados a una red e introducir un dni válido perteneciente a un jugador registrado en la base de datos. Una vez hecho esto debemos seleccionar el único escenario de juego disponible, el escenario de fabricación situado en el lado izquierdo. Si pasamos el cursor sobre el escenario de la parte derecha se nos mostrará un mensaje informándonos de que se encuentra en construcción y no podremos acceder a este.

Una vez hayamos seleccionado el escenario y hayamos introducido un dni válido, entonces se iniciará la carga del nivel de fabricación y una vez cargado se iniciará la interfaz de espera de jugadores.

Figura 155: Interfaz de creación de partidas

M.4.5 Interfaz de espera de jugadores

Esta pantalla aparecerá cuando hayamos creado una partida para un usuario jugador válido y se mostrará hasta que:

- Un jugador ejecute la aplicación ‘VRPanel-Player’ en la misma red a la que está conectado el dispositivo desde el que se crea la partida.
- Se pulsa el botón de ‘CANCELAR’ situado en la parte inferior izquierda de la interfaz.

Figura 156: Interfaz de espera de jugadores

M.4.6 Panel de control

Una vez creada una partida y habiéndose unido un jugador desde otro dispositivo y conectado a la misma red que el dispositivo que crea la partida, se nos mostrará el panel de control del manager. Desde esta interfaz podremos monitorizar las acciones del jugador en el escenario, dividiendose las funcionalidades que se describen en los siguientes apartados.

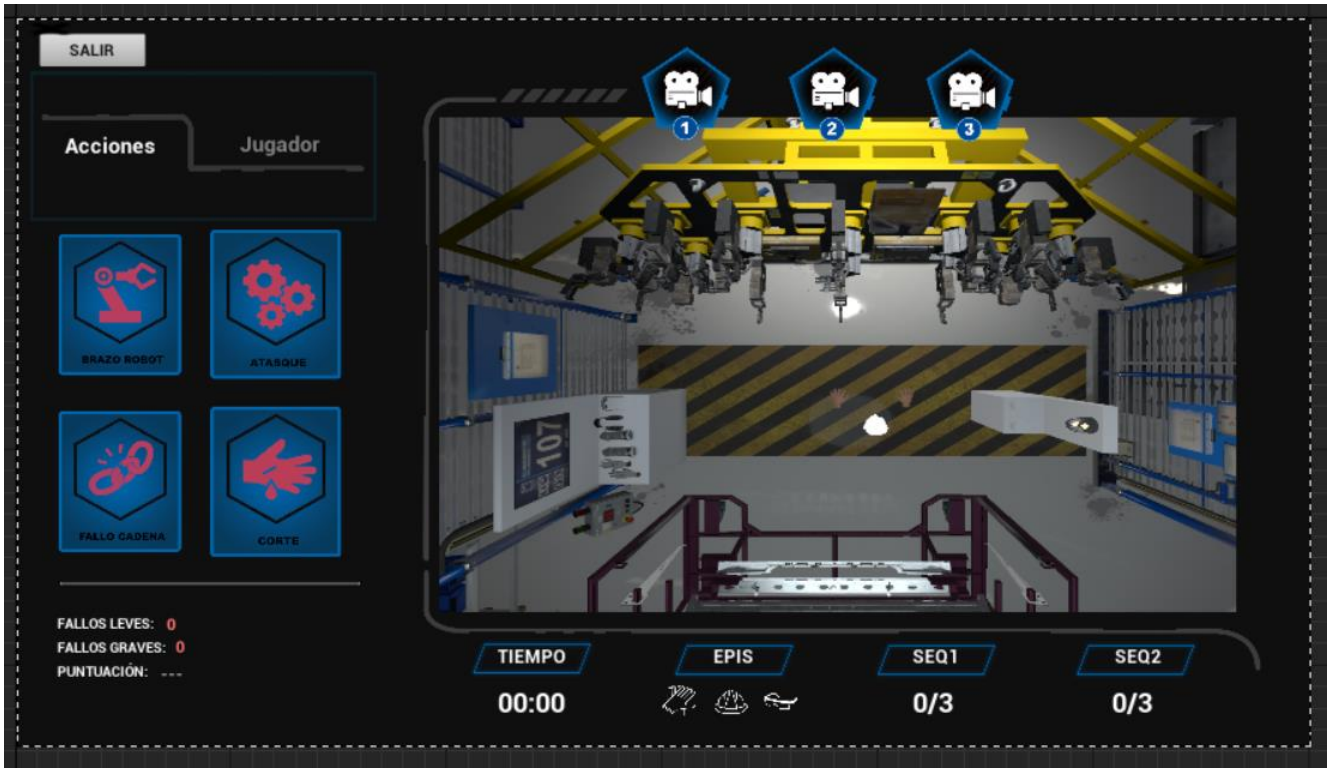


Figura 157: Panel de control del manager

M.4.6.1 Minimapa

Desde el minimapa situado en la parte central del panel podremos visualizar todo lo que está ocurriendo en el escenario con el jugador, de esta forma podemos saber en todo momento lo que está pasando en tiempo real.

M.4.6.2 Datos del jugador

Si se hace click sobre el botón que pone ‘Jugador’ se mostrarán los datos del usuario que está jugando en la partida.

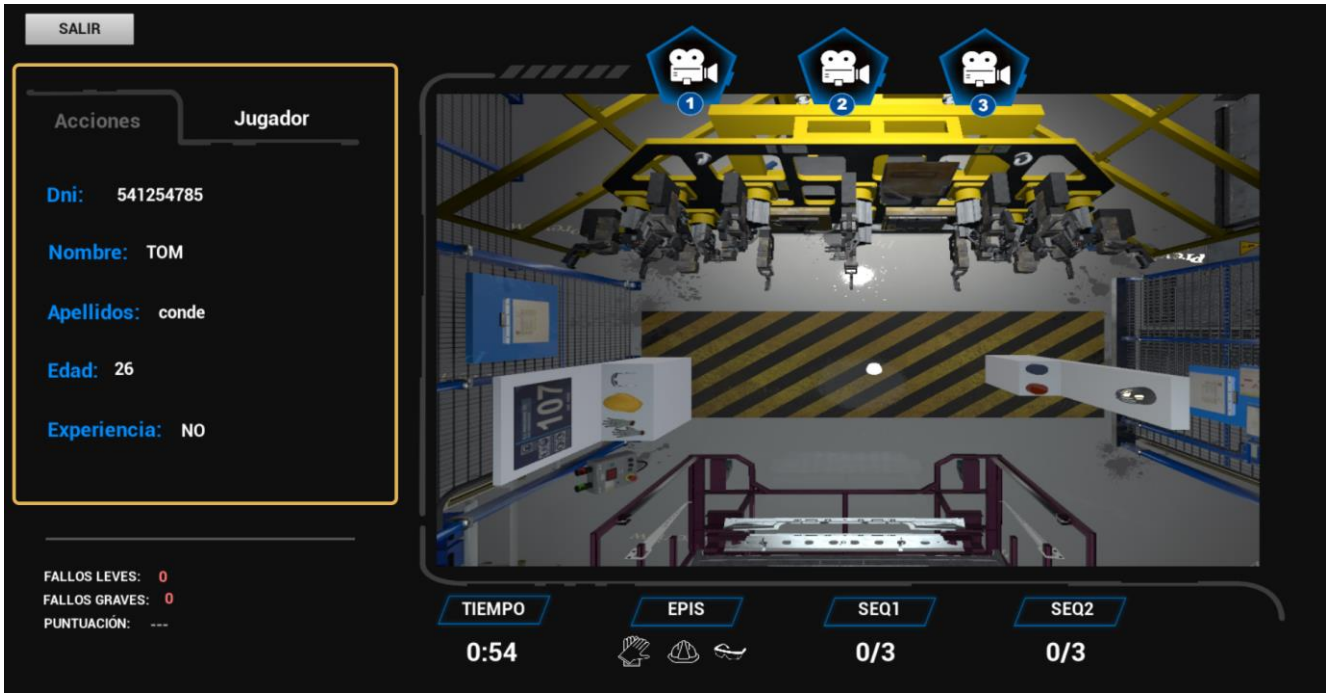


Figura 158: Señalado: datos del jugador actual en el panel de control

M.4.6.3 Secuencia del proceso de fabricación

En la parte de abajo podemos ver las distintas partes del proceso de fabricación que ha completado el usuario, de izquierda a derecha:

- Tiempo transcurrido en la partida.
- Protecciones de seguridad (EPIS) colocadas por el jugador (en verde si han sido colocadas).
- Piezas colocadas de la secuencia 1 de la fase de fabricación
- Piezas colocadas de la secuencia 2 de la fase de fabricación

Figura 159: Señalado: Indicadores de tiempo, protecciones y secuencia del panel de control

M.4.6.4 Acciones del manager

Se pueden realizar 4 acciones cuyos activadores se sitúan en la parte izquierda del panel de control, estos son:

- **‘Brazo robot’**: Acciona una proyección de material sobre el jugador, si este no lleva las gafas de protección, se contabilizará como un fallo grave.
- **‘Atasque’**: Acciona un atasque en la máquina soldadora, de modo que el jugador no puede continuar con el proceso de fabricación hasta que solvente el problema.
- **‘Fallo cadena’**: Acciona una simulación de un fallo en la cadena de producción que impide que el usuario siga con el proceso de fabricación hasta que se solvente el fallo por parte del jugador.
- **‘Corte’**: Acciona un corte en el controlador del jugador cuando sujeta una pieza metálica sin protecciones manuales. Si se acciona y se cumplen estas condiciones se contabilizará como un fallo grave.

M.4.6.5 Fallos y puntuaciones

Debajo del panel de acciones del manager sirve para informar de los fallos graves y leves que el usuario lleva cometidos durante el transcurso de la partida. Además, cuando el usuario finaliza el proceso de fabricación también se muestran las puntuaciones finales del jugador.

M.4.6.6 Guardado de partidas

Cuando el usuario completa el proceso de fabricación se da la opción de guardar la partida desde el panel de control. Si se confirma la acción, se guardará en la base de datos la partida junto con las acciones del manager que han sido lanzadas durante la partida. Si se cancela se cerrará la sesión y se volverá al menú principal.

Figura 160: Cuadro de confirmación de guardado de partidas

M.5 Funcionalidades del jugador – VRPanel-Player

A continuación, se describen los objetivos del jugador en el escenario de fabricación y los controles disponibles para llevar a cabo las acciones dentro del escenario de fabricación en una partida activa.

M.5.1 Proceso de fabricación

Cuando se inicia la aplicación el usuario aparece en un entorno de espera hasta que encuentra una sesión activa en la red, y esto ocurre cuando se crea una partida desde un dispositivo conectado a la misma red y que esté ejecutando la aplicación 'VRPanel-Manager'.

Figura 161: Nivel de inicio del ejecutable 'VRPanel-Player'

Una vez se ha unido a una sesión de juego, se carga el escenario de fabricación, en el que el usuario deberá seguir el siguiente proceso en el orden adecuado para completar la partida consiguiendo la mejor puntuación posible:

Protecciones

1. Guantes
2. Gafas
3. Casco

Secuencia 1

1. Colocar pieza metálica 1
2. Colocar pieza metálica 2
3. Colocar pieza metálica 3
4. Pulsador de cambio de fase

Secuencia 2

1. Colocar pieza metálica 4
2. Colocar pieza metálica 5

3. Colocar pieza metálica 6
4. Pulsador de cambio de fase

Solución de problemas

1. Solventar atasque del robot soldador
2. Solventar fallo de la producción en cadena

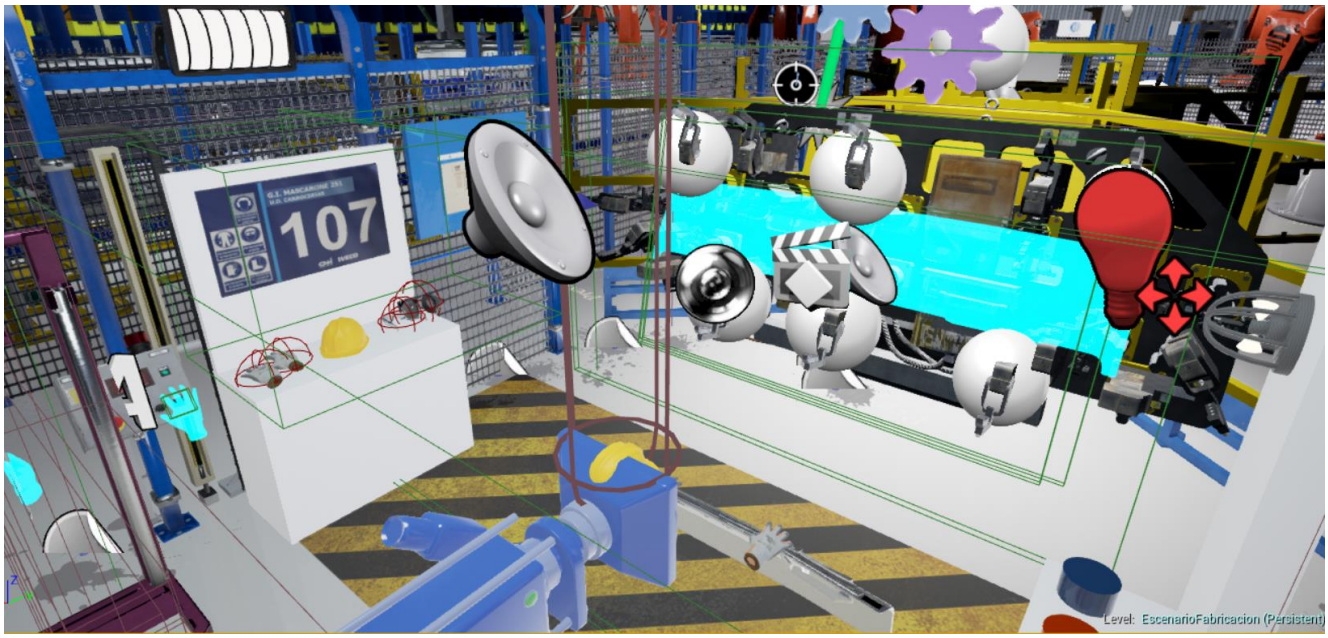


Figura 162: Escenario de fabricación

Para conseguir una puntuación mayor, lo mejor es ponerse las protecciones de seguridad cuanto antes para evitar posibles proyecciones de material o cortes por parte del usuario manager a través del panel de control, también cuando se acciona una fase de la secuencia de producción que no corresponde se comete un fallo leve, lo cual resta en las puntuaciones. Si se finaliza la partida sin haberse puesto las protecciones se contabilizará como un fallo grave y además se sumará un fallo importante que restará aún más puntos de la puntuación final.

M.5.2 Controles

A continuación, se muestran los controles utilizados para realizar las acciones del usuario jugador:

Protecciones

4. Guantes --- Tecla ‘G’
5. Gafas --- Tecla ‘K’
6. Casco --- Tecla ‘C’

Secuencia 1

5. Colocar pieza metálica 1 --- Tecla ‘1’
6. Colocar pieza metálica 2 --- Tecla ‘2’
7. Colocar pieza metálica 3 --- Tecla ‘3’
8. Pulsador de cambio de fase --- Tecla ‘P’

Secuencia 2

5. Colocar pieza metálica 4 --- Tecla ‘4’
6. Colocar pieza metálica 5 --- Tecla ‘5’
7. Colocar pieza metálica 6 --- Tecla ‘6’
8. Pulsador de cambio de fase --- Tecla ‘P’

Solución de problemas

3. Solventar atasque del robot soldador --- Tecla ‘Z’
4. Solventar fallo de la producción en cadena --- Tecla ‘A’

M.6 Recomendaciones

- Es recomendable configurar un mando compatible con Windows 10 para que la experiencia jugable sea lo más cómoda posible.
- Es recomendable seguir la configuración de HTC Vive y Steam y hacer la configuración de la habitación desde el setup de Steam VR, más información aquí.
- Si el juego la aplicación no consigue encontrar partidas en un largo periodo de tiempo es probable que tenga que reiniciar la aplicación.
- Es importante no iniciar el ejecutable ‘VRPanel-Manager’ si se tiene *SteamVR* abierto, es necesario cerrarlo antes de iniciar la aplicación puede que no se vea correctamente.
- Antes de iniciar el ejecutable ‘VRPanel-Player’ es importante iniciar *SteamVR* y comprobar que las gafas de realidad virtual funcionan correctamente.

Guía de instalación

G1. Distribución de contenidos

A continuación, se comenta brevemente el contenido de la documentación presentada:

- Memoria del trabajo
- Manual de usuario
- Guía de instalación
- Código fuente (comprimido)
    - Proyecto de Unreal Engine 4
    - Base de datos mysql
    - Scripts PHP
- Ejecutables (comprimido)
    - VRPanel-Manager
    - VRPanel-Player

Se distribuye el contenido en **2 DVD de 4,7GB**, teniendo en el CD 1 la memoria, el manual de usuario y la guía de instalación, mientras que en el CD 2 tenemos el código fuente y otra copia de la memoria, el manual de usuario y la guía de instalación en formato PDF.

G2. Configuración de la base de datos

Lo primero que será necesario es tener una base de datos SQL de nombre '**tfg_vrpanel'** que tenga la siguiente estructura:

```
SET SQL_MODE = "NO_AUTO_VALUE_ON_ZERO";
SET time_zone = "+00:00";


/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;
/*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;
/*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;
/*!40101 SET NAMES utf8mb4 */;

--
-- Base de datos: `tfg_vrpanel`
--

-- --------------------------------------------------------

--
-- Estructura de tabla para la tabla `eventomanager`
--

CREATE TABLE `eventomanager` (
  `id` int(11) NOT NULL,
  `id_partida` int(11) NOT NULL,
  `momento` varchar(20) NOT NULL,
  `tipoEventoManager` varchar(25) CHARACTER SET utf8 COLLATE
utf8_spanish_ci NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

-- --------------------------------------------------------

--
-- Estructura de tabla para la tabla `jugadores`
```

```
--

CREATE TABLE `jugadores` (
  `dni` varchar(9) CHARACTER SET utf8 COLLATE utf8_spanish_ci NOT NULL,
  `nombre` varchar(12) CHARACTER SET utf8 COLLATE utf8_spanish_ci NOT
NULL,
  `apellidos` varchar(20) CHARACTER SET utf8 COLLATE utf8_spanish_ci NOT
NULL,
  `edad` smallint(6) NOT NULL,
  `experiencia` tinyint(1) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

--
-- Volcado de datos para la tabla `jugadores`
--

INSERT INTO `jugadores` (`dni`, `nombre`, `apellidos`, `edad`,
`experiencia`) VALUES
('98765432U', 'player', 'uva', 31, 0);

-- --------------------------------------------------------

--
-- Estructura de tabla para la tabla `managers`
--

CREATE TABLE `managers` (
  `dni` varchar(12) CHARACTER SET utf8 COLLATE utf8_spanish_ci NOT NULL,
  `password` varchar(12) CHARACTER SET utf8 COLLATE utf8_spanish_ci NOT
NULL,
  `nombre` varchar(15) CHARACTER SET utf8 COLLATE utf8_spanish_ci NOT
NULL,
  `apellidos` varchar(25) CHARACTER SET utf8 COLLATE utf8_spanish_ci NOT
NULL,
  `correo` varchar(25) CHARACTER SET utf8 COLLATE utf8_spanish_ci NOT
NULL,
  `admin` tinyint(1) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

--
-- Volcado de datos para la tabla `managers`
--

INSERT INTO `managers` (`dni`, `password`, `nombre`, `apellidos`,
`correo`, `admin`) VALUES
('12345678U', '123456', 'admin', 'uva', 'admin@mail.com', 1);

-- --------------------------------------------------------

--
-- Estructura de tabla para la tabla `partidas`
--

CREATE TABLE `partidas` (
  `id` int(11) NOT NULL,
  `dni_manager` varchar(9) CHARACTER SET utf8 COLLATE utf8_spanish_ci NOT
NULL,
```

```
  `dni_jugador` varchar(9) CHARACTER SET utf8 COLLATE utf8_spanish_ci NOT
NULL,
  `tiempo` int(11) NOT NULL,
  `puntuacion` int(11) NOT NULL,
  `fecha` varchar(30) CHARACTER SET utf8 COLLATE utf8_spanish_ci NOT
NULL,
  `escenario` varchar(20) CHARACTER SET utf8 COLLATE utf8_spanish_ci NOT
NULL,
  `numFallos` int(11) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

--
-- Ã□ndices para tablas volcadas
--

--
-- Indices de la tabla `eventomanager`
--
ALTER TABLE `eventomanager`
  ADD PRIMARY KEY (`id`),
  ADD KEY `eventomanager_ibfk_1` (`id_partida`);

--
-- Indices de la tabla `jugadores`
--
ALTER TABLE `jugadores`
  ADD PRIMARY KEY (`dni`);

--
-- Indices de la tabla `managers`
--
ALTER TABLE `managers`
  ADD PRIMARY KEY (`dni`);

--
-- Indices de la tabla `partidas`
--
ALTER TABLE `partidas`
  ADD PRIMARY KEY (`id`),
  ADD KEY `dni_manager` (`dni_manager`),
  ADD KEY `dni_jugador` (`dni_jugador`);

--
-- AUTO_INCREMENT de las tablas volcadas
--

--
-- AUTO_INCREMENT de la tabla `eventomanager`
--
ALTER TABLE `eventomanager`
  MODIFY `id` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=28;
--
-- AUTO_INCREMENT de la tabla `partidas`
--
ALTER TABLE `partidas`
  MODIFY `id` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=77;
--
-- Restricciones para tablas volcadas
```

```
--

--
-- Filtros para la tabla `eventomanager`
--
ALTER TABLE `eventomanager`
  ADD CONSTRAINT `eventomanager_ibfk_1` FOREIGN KEY (`id_partida`)
REFERENCES `partidas` (`id`) ON DELETE CASCADE;

--
-- Filtros para la tabla `partidas`
--
ALTER TABLE `partidas`
  ADD CONSTRAINT `partidas_ibfk_1` FOREIGN KEY (`dni_manager`) REFERENCES
`managers` (`dni`),
  ADD CONSTRAINT `partidas_ibfk_2` FOREIGN KEY (`dni_jugador`) REFERENCES
`jugadores` (`dni`);

/*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;
/*!40101 SET CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS */;
/*!40101 SET COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION */;
```

Para poder probar la aplicación y ejecutar el código del proyecto es recomendable descargar la aplicación XAMPP disponible desde su página oficial ya que provee un servicio de base de datos y un servidor apache de forma local en el dispositivo.

Se deben colocar los scripts PHP adjuntos con este documento en la siguiente ruta:

- Si se utiliza XAMPP:
  - C:/xampp/htdocs/VRpanel/
- Si se utiliza otro programa, la aplicación buscará los scripts en el directorio:
  - http://localhost/VRPanel/

Una vez hecho esto, es necesario modificar las primeras líneas del script PHP '**tfg_db_connection.php'** y cambiar los datos por los de un usuario existente en la base de datos creada:

```php
<?php

$servername = 'localhost';
$username = 'tomdoniphon';
$password = 'adminvrpanel';
$dbname = 'tfg_vrpanel';

// Creamos la conexión
$conn = new mysqli($servername, $username, $password,
$dbname);

// Se comprueba la conexión
if ($conn->connect_error) {
    echo(json_encode(array('status'=>"Fallo en la conexion:
" . $conn->connect_error)));
     die;
}

?>
```

Figura 163: Script PHP de conexión con la base de datos

Una vez seguidos estos pasos, ya podemos disponer de nuestra base de datos y comenzar a ejecutar la aplicación. Los datos son guardados desde el ejecutable que utiliza el manager, es decir, desde el ejecutable '**VRPanel-Manager'**, por lo tanto, será necesario tener una base de datos activa con las características descritas a la hora de utilizar la aplicación.

G3. Configuración de los programas para el código fuente

El código fuente está contenido dentro de un proyecto de Unreal Engine 4 realizado con la **versión del motor 4.13.2**. Para poder acceder al código fuente son necesarios los siguientes programas:

- Epic Games Launcher
- Motor Unreal Engine 4.13.2
- Visual Studio 2015

Bastará con copiar la carpeta 'Código_fuente' a nuestro dispositivo y hacer doble click sobre el ejecutable '***uproject'*** contenido dentro de esta carpeta. Si esto no resultara debido a una mala instalación del motor, podemos acceder al motor desde el lanzador de Epic Games y desde este seleccionar el archivo del proyecto que queremos iniciar.

G4. Inicio de la aplicación

Se adjuntan dos ejecutables junto con este documento:

- **VRPanel-Manager**. El dispositivo que ejecute esta aplicación deberá cumplir los requisitos mínimos especificados en el manual de usuario y, además, será el dispositivo donde se almacenen los scripts PHP necesarios para conectar la base de datos con la aplicación.
- **VRPanel-Player.** El dispositivo que ejecute esta aplicación deberá disponer, además del software específico que se detalla en el manual de usuario, unas gafas de realidad virtual HTC VIVE y deberá contar con un espacio de juego de al menos 1.5 x 2 metros.