Diseño e implementación de un juego online para la plataforma .NET

Autor: Francisco Arsenio Espinosa Béjar
Director: Alberto Nuñez Covarrubias

Universidad Carlos III de Madrid

Junio, 2009

Tabla de contenido

TABLA DE CONTENIDO

Índice de figuras

ÍNDICE DE FIGURAS

ÍNDICE DE FIGURAS

Índice de tablas

ÍNDICE DE TABLAS

1

Introducción

La finalidad de este proyecto es diseñar y posteriormente implementar un juego online. La Real Academia de la Lengua Española define el juego como:

"Ejercicio recreativo sometido a reglas, y en el cual se gana o se pierde"

Cuando empleamos el término **juego** este puede referir a *"ejercicios recreativos"* muy distintos: que para jugar se precisen varias personas o una sola; que se emplee el uso de materiales como cartas, balones, fichas, etc. o que simplemente se use el habla; que se juegue sobre un tablero, al aire libre o en unas instalaciones determinadas...

Queda claro que cuando hablamos de juegos en un sentido general, el objeto designado puede llegar a ser muy variado, pero en este proyecto cuando utilicemos el término juego el lector deberá tener en mente que se está haciendo referencia al género de los videojuegos. Los videojuegos son ese tipo de juego donde para jugar se emplea un soporte digital, pero la finalidad sigue siendo el *"ejercicio recreativo"*, y aunque no siempre el objetivo sea ganar o perder si es verdad que el juego está sometido a reglas.

Actualmente podemos afirmar fácilmente que la industria del entretenimiento genera muchos beneficios, y que los videojuegos son los que generan la mayor parte. Dado el alto desarrollo de la tecnología actual, y que las personas están cada vez más acostumbradas a utilizar computadoras o dispositivos móviles, el grupo de usuarios de videojuegos no se reduce como años atrás a niños y jóvenes, ya que actualmente la edad de usuarios de videojuegos varía desde los 5 a los 50 años.

Este creciente aumento del consumo de videojuegos ocurrido durante los últimos años junto con el propio gusto por ellos, ha despertado en mí el interés por averiguar y llevar a cabo los distintos procesos que integran el desarrollo de un juego.

Para el desarrollo de este proyecto me he inspirado en un juego existente en el mercado, por lo cual el proceso creativo de idear las reglas y la forma de juego son anteriores a dicho proyecto. El juego que se va a diseñar y posteriormente implementar, estará destinado para su uso en computadoras personales y consistirá en un juego que permita a sus usuarios jugar partidos de rugby cuyo enfrentamiento tiene lugar en el espacio, de ahí que haya titulado al juego **SpaceRugby**; y está inspirado en el juego de mesa **BloodBowl**, propiedad de *Games Workshop* (**?**).

Como hemos dicho al principio, se trata de un juego online, lo cual quiere decir que puede jugarse a través de internet. Los usuarios se conectarán con un servidor, que es el que posee toda la lógica de juego, y a través de él podrán jugar las partidas.

1.1 Objetivos

El objetivo de este proyecto es desarrollar un juego online utilizando la plataforma .NET.

Para poder llevar a cabo estos procesos no basta con poseer conocimientos sobre análisis, programación, tecnologías de desarrollo, ya que estos conocimientos solo abarcan una parte del desarrollo de juegos.

Podemos decir que el desarrollo de un videojuego consta por lo general de seis procesos, aunque este número obviamente varía dependiendo del juego en concreto, que son:

- Desarrollo de los elementos del entorno multimedia
- Desarrollo de la lógica de manejo de gráficos
- Desarrollo de la lógica del juego
- Desarrollo de las comunicaciones
- Integración de los distintos elementos
- Pruebas (testing)

A continuación explicaremos brevemente en que consiste cada proceso y que objetivos quieren alcanzarse.

1.1.1 Desarrollo de los elementos del entorno multimedia

En este proceso se producen los elementos que interaccionan con el usuario (tanto de forma visual como auditiva), y aunque es importante que todos los procesos se lleven a cabo de forma satisfactoria, los elementos multimedia desarrollados ocuparán un papel importante en el efecto final del juego sobre el usuario, ya que se trata de la parte con la que el usuario está en continuo contacto.

Por esta razón la introducción en el mundo del modelado 3D y el aprendizaje de herramientas que permitan el desarrollo de dichos modelos son objetivos secundarios de este proyecto.

1.1.2 Desarrollo de la lógica de manejo de gráficos

Cualquier videojuego precisa disponer de un motor gráfico, que es el elemento encargado del manejo y control de los elementos gráficos del juego en el dispositivo hardware destinado para su visualización.

Por ello la investigación sobre las distintas alternativas para el manejo de gráficos que se ofrecen en la actualidad, identificando sus características globales y propias, junto con la elección de una alternativa para su uso en nuestro videojuego son objetivos secundarios de este proyecto.

1.1.3 Desarrollo de la lógica del juego

Todos los juegos tienen unas normas o reglas que definen la manera de jugar, aquello que se puede hacer y cuando se puede hacer, y de igual manera lo que no se puede hacer. Estas reglas deben ser incorporadas al videojuego de manera que el usuario deba cumplirlas en todo momento durante la partida.

Denominamos lógica del juego a la parte del videojuego que contiene los elementos que hacen que se cumplan las reglas; el desarrollo de este elemento es también un objetivo secundario del proyecto.

1.1.4 Desarrollo de las comunicaciones

En nuestro juego se precisan dos jugadores para jugar una partida; estos jugadores se conectan previamente a un servidor, por lo tanto las acciones y sucesos de la partida deben transmitirse del servidor a los jugadores y viceversa.

Son por lo tanto son objetivos secundarios de este proyecto investigar los distintos modelos y formas de comunicación que pueden utilizarse, así como elegir uno de dichos modelos para su aplicación en las comunicaciones del juego.

1.1.5 Integración de los distintos elementos

Como en todo proyecto, cada bloque se desarrolla en la medida de lo posible de forma independiente y luego se procede a la integración de dichos bloques en la aplicación. El problema que existe es que las partes a integrar en un juego son muy distintas: tenemos modelos 3D, código de manejo de gráficos, sonido, la entrada de usuario (teclado, ratón), comunicaciones, etc.

Dado que no existe ningún procedimiento estándar para la integración de elementos multimedia al desarrollo de una aplicación, así pues otro objetivo secundario del proyecto será examinar la complejidad que se precisa para integrar los distintos elementos que componen el juego.

1.1.6 Pruebas (testing)

Otra parte muy importante en el desarrollo de los videojuegos es la parte de testing o pruebas, y lo normal es que existan equipos de testers que prueban el juego en distintas fases, aportando su opinión sobre mejoras en usabilidad y manejabilidad del juego, así como pruebas de que el juego cumple con la lógica establecida y está exento de errores (por ello que las pruebas precisen ser exhaustivas).

En nuestro caso esta parte no será objeto de estudio en el proyecto, por lo que se limitará a hacer las pruebas necesarias para que el juego funcione correctamente, pero no se llevarán a cabo pruebas de usabilidad por medio de grupos de usuario, ni pruebas de carga máxima del servidor, etc.

1.2 Estructura del documento

En el capítulo 2 se explican los tipos de modelos 3D más importantes, que son las librerías de manejo de gráficos y los motores gráficos. También se detallan las técnicas más comunes en la comunicación de procesos y los lenguajes de programación más utilizados en el desarrollo de juegos.

El capítulo 3 contiene la descripción el análisis del proyecto, donde se detallan los casos de uso, las tecnologías que se utilizarán en la implementación del juego y el análisis de las comunicaciones.

En el capítulo 4 se detalla la forma en que se han implementado las reglas y lógica del partido. También se detallan las interfaces de usuario y las clases que manejan la información del partido y los jugadores. Este capítulo detalla también la forma en que se resuelve la persistencia de los datos del juego, así como el diseño de las comunicaciones y el diseño de los elementos 3D.

En el capítulo 5 se realiza una descripción del proceso de instalación de los programas necesarios para la ejecución del juego. También se describe el proceso de instalación del código ejecutable del juego, así como de los ficheros con los modelos 3D y los ficheros de configuración del entorno.

En el capítulo 6 se describe Manual de usuario.

El capítulo 7 contiene el presupuesto detallado del proyecto.

En el capítulo 8 se detallan las Conclusiones del proyecto y trabajo futuro a realizar.

2

Estado del arte

A principios de los 70 aparecieron las primeras máquinas recreativas con juegos como Computer Space, Pong, Breakout (precursor del arkanoid), Space Invaders y por último en 1980 apareció PAC-MAN. Todos ellos son juegos sencillos en 2D y con gráficos muy básicos, pero aún así cargados de entretenimiento.

Los 80 es la época de las consolas, en 1982 aparece la Atari 5200, al año siguiente SEGA había lanzado su SG1000 sin mucho éxito. En 1984 apareció la consola NES que irrumpió con mucha fuerza en el mercado; posteriormente SEGA lanzó en 1986 su Sega Master System; 1988 fue el año del lanzamiento de la Sega Megadrive, que funcionaba a 16 bits y que abarcó el mercado de las consolas hasta la aparición de las consolas de 32 y 64 bits. Esta época es la de juegos tan conocimos (que se convirtieron en sagas) como Mario Bros, Sonic, Double Dragon, Bomberman, Donkey Kong, Golden Axe, Faxanadu, Mortal Combat, Final Fantasy, Lemmings, The legend of celda, etc.

En los años 90 llegaron los videojuegos en red. El primero de ellos es Doom, creado en 1993 por ID Software, que introdujo como novedad que se podía jugar en red. Posteriormente ID Software lanzó en 1996 Quake, este juego es el primero de su tipo que emplea gráficos 3D y la posibilidad de partidas multijugador a través de Internet. Quake utiliza un motor de juego propio denominado QuakeEngine. En 1998 apareció Unreal, del mismo estilo que Quake, pero con su propio motor de juego Unreal Engine.

Posteriormente han surgido muchos juegos que utilizan dichos motores. Ejemplos de juegos que utilizan el motor de Quake son Half-Life, Call of Duty y la saga Medal of Honor. Ejemplos de juegos que utilizan el motor de Unreal son Lineage II, Gears of War y la saga Tom Clancy's: Splinter Cell entre otros.

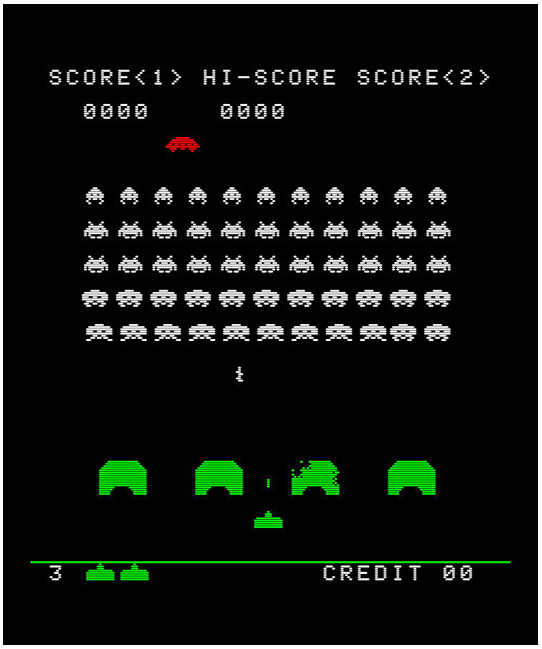


Figura 2.1: Space Invaders

Como se acaba de ver, en la actualidad es común que los nuevos juegos desarrollados utilicen motores gráficos ya existentes del mercado de tal forma que se ahorran el coste y el tiempo de desarrollo de un motor propio. Aparte del motor gráfico existen otros elementos que componen un juego tal como elementos gráficos, comunicaciones, etc.; en este capítulo se irán detallando las herramientas existentes para ayudar al desarrollo del proyecto.

2.1 Modelado 3D

¿Qué es el modelado 3D?

El modelado 3D es el proceso de desarrollo de una representación matemática de un objeto tridimensional cuyo producto es llamado modelo u objeto 3D.

Los objetos 3D representan un objeto de tres dimensiones mediante una colección de puntos en el espacio conectados mediante distintos elementos geométricos (líneas, triángulos, superficies curvas, etc). Dichos elementos pueden ser generados de forma manual o automática (por ejemplo una generar un cono se puede realizar mediante una

fórmula matemática), y la mayoría de programas de modelado ofrecen ambas posibilidades, pudiendo además en un primer paso crear un objeto 3D de forma automática y posteriormente modificarlo manualmente.

¿Qué tipos de modelos 3D existen?

Dependiendo si el modelo 3D se ha creado definiendo el volumen que el objeto representa o solamente su superficie nos encontraremos con que se trata de un modelo sólido o un modelo de cáscara (o frontera). Para identificar la diferencia imaginemos que creamos dos modelos 3D de un huevo, el modelo sólido sería un modelo macizo y por lo tanto conoceríamos todos los puntos del huevo (incluidos los de su interior); por otro lado el modelo frontera sólo indica la superficie del objeto, y por lo tanto para un punto que no pertenezca a dicha superficie ignoramos si está dentro o fuera del volumen del huevo.

Es fácil adivinar que los modelos sólidos son los más realistas y por lo tanto más difíciles de construir; las operaciones con modelos sólidos son más pesadas pero exactas, por ello son utilizados para simulaciones y aplicaciones especializadas de visualización (como la construcción sólida de figuras geométricas o iluminación de sólidos). Los modelos frontera permiten trabajar de forma más ligera con los objetos 3D (traslaciones, escalados, rotaciones, etc.) por lo cual son más adecuados cuando la mayoría de las operaciones son sobre la apariencia del objeto (posición, tamaño, color, etc.) y no sobre su volumen.

Dado que para nuestro caso lo que vamos a efectuar son operaciones sobre la totalidad de los objetos (mover jugadores, mostrar las casillas, etc.) y no sobre parte de su volumen, los modelos tipo frontera son más fáciles de construir y las operaciones sobre ellos requieren menos tiempo de cómputo.

Formas de representación de un objeto 3D

Una vez dicho lo que es un objeto 3D, para poder modelar debe utilizarse algún programa de modelado de los existentes en el mercado: 3D Studio Max, Maya, POV-Ray, Blender, TrueSpace, Rhino 3D, etc. Para elegir el programa debe saberse primero que existen diversas formas de generar y representar un modelo 3D, ya que puede hacerse mediante mayas poligonales, NURBS, Splines y parches o primitivas principalmente.

Maya poligonal: En una maya poligonal el objeto es definido como una colección de vértices, bordes y caras que representan su superficie. Las caras normalmente son triángulos o cuadriláteros ya que ello simplifica el cómputo de renderizado, pero en general podrían ser cualquier polígono. Ventajas: fácil diseño de objetos poligonales. Desventajas: El problema de las mayas poligonales es el suavizado de bordes y la representación de superficies curvas; ya que en el caso de superficies curvas se deberá fraccionar dicha superficie para realizar una aproximación mediante polígonos, y si dicha aproximación todavía resulta poco adecuada se deberá proceder a realizar de nuevo una aproximación con polígonos más pequeños aumentando de forma considerable el número de elementos de la maya; el suavizado de bordes es un problema similar, ya que unos bordes protuberantes ponen de manifiesto que el objeto está construido a base de polígonos (y lo que se desea es dar sensación de realismo) lo cual se soluciona mediante la inclusión de más polígonos de forma que se suavice el borde y aumente la sensación de realismo. Ejemplos de programas de modelado cuya principal forma de representación de modelos es mediante mayas poligonales son: Wings3D, Blender y 3D Studio Max.

NURBS (Non Uniform Rational B-Splines): Modelo matemático muy utilizado en los gráficos por ordenador para generar y representar curvas y superficies. Usar NURBS permite representar formas geométricas de forma compacta, además son manejadas de forma eficiente por el computador. Las superficies NURBS son funciones de dos parámetros que se mapean a una superficie tridimensional, para determinar la forma de dicha superficie se necesita determinar unos puntos de control, los cuales pueden tanto estar conectados directamente a la curva o superficie o unidos entre sí por una goma elástica. Normalmente la superficie del objeto modelado estará compuesta por varias superficies NURBS conocidas como parches, dichos parches deberían estar ajustados de forma que sus límites fueran invisibles. Ventajas: los modelos representados son invariantes a transformaciones de escalado y perspectiva, permiten diseñar una gran variedad de formas y requieren poco espacio para almacenar el modelo. Desventajas: El problema que tienen los objetos representados mediante NURBS es que para que su evaluación pueda llevarse a cabo en un tiempo razonable se precisa de algoritmos precisos, es decir, dado que las superficies son generadas mediante fórmulas matemáticas,

pueden ocurrir pequeños errores de redondeo o truncado que debido a efectos de propagación de errores provoquen que el tiempo de evaluación se incremente o que la superficie se muestre de forma incorrecta. Ejemplos de programas de modelado cuya principal forma de representación de modelos es mediante NURBS son: Maya y Rhino3D.

Splines y parches: son una forma de representación parecida a NURBS solo que más general, ya que para generar las superficies no se utilizan exclusivamente Splines de Bézier (Bézier Splines o B-Splines en inglés). En este caso las curvas deberán ser definidas por el usuario, al igual que la manera de ajustar (o parchear) las curvas. Las ventajas e inconvenientes son los mismos que con NURBS, contando además con el inconveniente de que es menos intuitivo crear un modelo usando este procedimiento que usando NURBS. La mayoría de programas permite generar objetos mediante splines, ejemplos de ello son 3D Studio max y Blender.

Primitivas: Representar un objeto mediante primitivas significa que a partir de unos elementos básicos, generalmente son el cubo, cilindro, esfera, el cono y el toro, se componen elementos de mayor complejidad que en fondo seguirán siendo objetos compuestos por elementos primitivos. Ventajas: El beneficio de operar con primitivas es que dichos elementos están definidos matemáticamente por lo cual las operaciones con ellos siempre son precisas y el tiempo de cálculo predecible, además algunos programas permiten al usuario definir sus propias primitivas y también ofrecen otras primitivas más complejas a parte de las mencionadas como son el elipsoide supercuadrático, fractales julia o isosurfaces (superficies utilizadas para representar puntos de un valor constante dentro de un espacio tridimensional); El modelado mediante primitivas es muy utilizado para aplicaciones técnicas (dado que los modelos son en el fondo un agregado de elementos definidos matemáticamente su el cálculo con dichos elementos es preciso y más sencillo que mediante otras representaciones). Desventajas: en el caso de modelaje de elementos orgánicos no es muy adecuado ya que es difícil encontrar la definición de una primitiva que represente el objeto y la composición no siempre resulta rentable ya que otras formas de representación ofrecen un mejor resultado y en menor tiempo. Un ejemplo de programa que opera directamente utilizando la representación mediante primitivas es POV-Ray, otro programas utilizan las

primitivas para modelar y posteriormente transforman el modelo a una maya para operar con la maya.

Otros elementos del modelado de objetos 3D: texturas y animaciones

Hasta ahora solo hemos hablado de la forma del objeto, de su representación, pero independientemente de como este representado, ¿cuál será su color? ¿Y si el objeto no tiene un solo color? ¿Y si está estampado con algún tipo de dibujo?

Materiales y texturas

Cuando a un objeto 3D le asociamos un color, se dice que se le está asociando un material, en cualquier programa de modelado se le asigna un material por defecto (aunque sea uno con el color blanco), mientras que cuando mapeamos una textura a un objeto, le estamos proporcionando no solo un color sino que también una trama de forma que puede ayudar a que el usuario tenga la sensación de que el objeto tiene una superficie irregular.

El mapeado UV es un proceso mediante el cual se transforma una imagen 2D para representar un objeto 3D. El mapa UV transforma el objeto 3D sobre una imagen conocida como textura. Mientras que en el objeto 3D las coordenadas son X Y Z, sobre la textura las coordenadas son U y V (de ahí su nombre). Este mapeado crea el efecto como si la imagen estuviera pintada sobre la superficie del objeto 3D.

Pero en la actualidad es común aplicar más de una textura a un objeto, lo cual se llama texturizado múltiple. Las técnicas más comunes cuando hablamos de texturizado múltiple son el mapeado de normales (Bump mapping), los mapas de iluminación (light map), y el texturizado mediante plantilla (stencil mapping).

Bump mapping: es una técnica en la cual se realiza una perturbación de la normal de la superficie en cada píxel, de acuerdo con un mapa de alturas (una imagen 2D donde las distintas escalas de grises representan distintas alturas, cuanto más oscuro más profundidad, y cuanto más claro significa que posee mayor altura) se modifica la normal de la superficie del objeto 3D antes de que se calcule la iluminación del objeto. Mediante esta técnica se puede conseguir que los objetos tengan la apariencia de poseer una textura compleja (como la corteza de un árbol) conseguida gracias al efecto de iluminación sobre el color base del objeto.

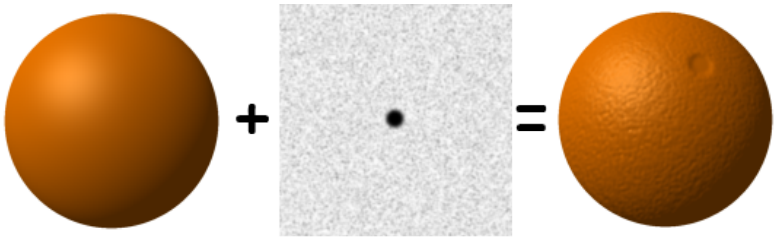


Figura 2.2: Bump Mapping

Mapeado de luces: es una técnica para determinar el brillo de las superficies de los objetos 3D mediante una imagen en escala de grises. Dependiendo del número de niveles de brillo que se empleen se tardará más o menos en renderizar el objeto, y éste tendrá más o menos resolución. Es una buena opción cuando no queremos que se esté recalculando como afecta la luz a la superficie del objeto, sobre todo cuando se trata de suelos y paredes y así controlar el brillo que emiten.

Stencil mapping: es una técnica para combinar varias texturas. El método consiste en especificar en una plantilla como se mezclan dos texturas para formar una tercera textura resultado; a esta textura resultado se le puede aplicar otra plantilla para mezclarse con otra textura consiguiendo así una nueva textura resultado, y así sucesivamente con tantas texturas como se desee. Este método es muy utilizado en el renderizado de superficies, ya que por ejemplo se desea que un terreno posea zonas de arena, zonas de césped y en la montaña zonas de tierra y nieve, así pues para combinar dichas texturas necesitaremos 3 plantillas.

Acabamos de ver las técnicas más comunes de texturizado, no es necesario continuar y cubrir todas las existentes, bastará tener en mente que el programa que elijamos para modelar deberá ofrecer al menos las técnicas de texturizado vistas.

Animación de modelos

Otra funcionalidad de los programaws de modelado a tener en cuenta es la animación de objetos. Ya que no todos los programas de modelado no ofrecen la funcionalidad de crear objetos 3D animados, debemos considerar pues que tipo de animaciones de las expuestas a continuación vamos a necesitar:
Animaciones mediante elementos 2D:

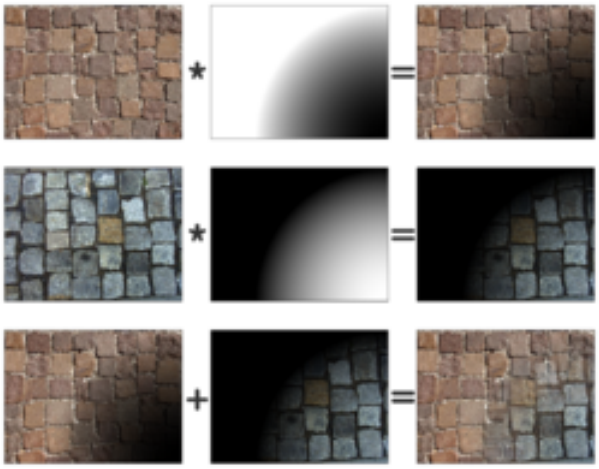


Figura 2.3: Stencil Mapping

Animación mediante Sprites: un sprite es una animación realizada mediante imágenes 2D o 3D que se integran dentro de un escenario, por lo tanto no se trata de una animación 3D sino de una animación en dos dimensiones (aunque la imagen utilizada tenga tres); esto es conveniente a tenerlo en cuenta ya que si el Sprite debe integrarse en un escenario 2D no habrá problema pero si se va a integrar en un escenario 3D entonces debemos saber que el Sprite no podrá interactúa con elementos 3D de la escena y además carecerá de perspectiva (por lo cual deberá programarse a mano si se desea). Inicialmente el término Sprite refería a la combinación de varias imágenes para los videojuegos en dos dimensiones mediante el uso de hardware especial, conforme aumentó la capacidad de cálculo y dicho hardware dejó de ser necesario, el término Sprite pasó a referirse a la combinación de imágenes 2D en sí misma. El objetivo de un Sprite es que al sucederse la combinación de imágenes se perciba la sensación de que se trata de un objeto animado y no de una combinación de imágenes. Cuando se trata de Sprites en 2D, suelen estar hechos a mano de forma artesanal, los Sprites 3D sin embargo se producen a partir de un modelo 3D que se ha ido transformando y del que se han tomado las instantáneas necesarias para quedar conformado.

Animaciones mediante elementos 3D:

Animaciones sobre objetos: son animaciones en las que el objeto de las operaciones

Figura 2.4: Animación con sprites

es el objeto en su totalidad. Las operaciones que se suelen realizar en las animaciones sobre objetos son traslaciones, rotaciones, escalados, etc. Esto es útil si se quiere hacer por ejemplo un balón que bote o se mueva, ya que se quiere que bote o se mueva todo el balón y no solo una parte. Este tipo de animación viene muy bien cuando se trata de un objeto sencillo (como un balón) o una animación sencilla (un coche que gira en un expositor).

Animaciones sobre partes de objetos: son animaciones en las que el objeto de cambio son partes específicas del objeto total. Las operaciones que se realizan en estas animaciones son las mismas que las que se realizaban en la anterior, con la diferencia que se realizan directamente sobre pixels o conjuntos de pixels. Este tipo de animación es la utilizada para animaciones complejas (una persona que anda, un brazo robótico cogiendo una pieza, etc.). Dentro de este tipo de animaciones encontramos los siguientes subtipos:

- **Morph:** Es una animación de vértices en la cual se almacenan una serie de posiciones de vértices. Cada keyframe de la animación se mueven los vértices a la posición correspondiente, según el valor almacenado. Dependiendo del como se realice el renderizado, en los frames existentes entre varias keyframes se interpolará la posición de los vértices para rellena, o al contrario se mantendrán en la última posición hasta el siguiente keyframe y cambien a su nueva posición (lo que se conoce como salto). Este tipo de animaciones tiene la ventaja de que se posee control total sobre los vértices, el problema

es que en la animación la posición del vértice debe ser calculada, y dado que los objetos suelen tener gran cantidad de vértices esto lleva su tiempo, por lo tanto las animaciones realizadas con morph suelen ser cortas (generalmente por debajo de las 100 frames). Ejemplos típicos de animaciones con morph son animaciones sobre ropa, superficies y expresiones faciales.

- **Skeletal:** Es una animación en la cual el objeto está representado mediante dos partes; la primera representa la superficie del objeto, y la segunda representa una estructura jerárquica de huesos (en inglés bones) usados exclusivamente para la animación, es lo que se llama esqueleto (en inglés skeleton, de ahí el nombre). Una vez tenemos la superficie del objeto, se construyen una serie de huesos; cada hueso tiene una transformación tridimensional que incluye posición, escalado y orientación, y opcionalmente tiene un padre; de esta forma la transformación final de un hueso hijo es el producto de la transformación del padre y la suya propia; cada hueso del esqueleto tiene asociado alguna parte de la representación visual del objeto, típicamente vértices , y un vértice puede estar asignado a varios huesos (e incluso se puede asignar el grado de dependencia al hueso y de esta forma hacer que las transformaciones de ese hueso tengan mayor efecto en el vértice que las de otros huesos). Así pues lo que se determina en los keyframes son las transformaciones de los huesos, la posición de los vértices se calcula a través del producto de las transformaciones de los huesos a los que están asociados. Como resultado en las animaciones de esqueleto, se pueden realizar transformaciones sobre conjuntos de vértices de forma más precisa y con menos fallos que si se realizaran vértice a vértice, son idóneas para representar movimiento de objetos vertebrados o máquinas, y al trabajar con conjuntos de vértices son más rápidas. La desventaja es que para tratar un vértice de forma aislada hay que crear un hueso para dicho vértice, lo cual nos lleva a que si queremos tratar muchos vértices de forma aislada este tipo de animaciones no es factible; además las transformaciones de los vértices son producto de las del hueso, lo cual no indica que el efecto obtenido sea realista (por ejemplo al tratarse de superficies complejas como músculos). Son el tipo de animación ideal para animaciones prolongadas (superiores a los 100 frames).

A priori dado que las animaciones serán con jugadores, nos interesa que el programa de modelado permita como mínimo generar animaciones de esqueleto. Estas animaciones también podrían hacerse mediante sprites, pero en este caso aumenta mucho el trabajo porque el número de animaciones a realizar es alto ya que para un movimiento (imaginemos un jugador andando) hay que repetirlo para todas las posibles direcciones, lo cual como ya hemos dicho, multiplica el número de animaciones.

Programa de modelados

Los tres programas de modelado más importantes y de mayor uso en el desarrollo de modelos para juegos son: 3D Studio Max, MilkShape 3D y Blender.

3D Studio Max

Es una herramienta desarrollada por Autodesk Media and Entertainment disponible exclusivamente para el sistema operativo Windows. Permite el modelado con mayas poligonales, NURBS y Splines, además de gran variedad de tipos de renderizado.

Entre sus funcionalidades incluye la definición de scripts para así poder automatizar tareas; la posibilidad de importar archivos DWG; multitud de formas de texturizado; modelado tanto sólido como de cáscara; y herramientas para realizar animaciones tipo skeletal.

Entre los juegos modelados con 3D Studio se encuentran: Diablo II, Lineage II, la saga Halo del 1 al 3 y World of Warcraft.

Una licencia de estudiante por 13 meses cuesta 80 euros, mientras que una perpetua son 234 euros.

MilkShape 3D

Es una herramienta propiedad de chUmbaLum sOft disponible exclusivamente para el sistema operativo Windows.

MilkShape 3D permite el desarrollo de modelos 3D de cáscara por medio de mayas poligonales. Permite todas las típicas operaciones básicas como seleccionar, rotar, mover, escalar, subdividir, etc. y además permite la edición a bajo nivel de vértices; trabaja con texturas, y sólo permite animaciones tipo skeletal.

MilkShape 3D permite importar y exportar a gran variedad de formatos (algunos son formatos de otros programas de modelado y otros son de juegos): 3D Studio ASC,

AutoCAD DXF, 3D Studio Max, DirectX, Half-Life, Maya, Ogre, Pov-Ray, Quake, VRML, etc.

MilkShape que permite 30 días de uso gratuito, posteriormente deberá adquirirse una licencia que cuesta 25 euros.

Blender

Blender es un programa de modelado 3D que se distribuye bajo GPL, y está disponible para múltiples sistemas operativos, entre los más importantes Windows, OS X, Linux y Solaris.

Blender permite el modelado con mayas poligonales, curvas Bezier, superficies NURBS y esculpido digital entre otros. Entre sus funcionalidades permite animaciones de esqueleto, animaciones tipo morph, animaciones no lineales, deformaciones basadas en curvas y mayas, texturas tipo UV stencil y Bump, la ejecución de tareas mediante scripts en Python, y la edición de audio y video.

Pese a no haberse utilizado todavía para el modelado de juegos de carácter profesional, Blender es programa de modelado con más copias instaladas del mercado, lo cual indica que es muy utilizado, quizá no por los expertos, pero si por aficionados.

2.2 Biblioteca de manejo de gráficos

En este capítulo se presentarán las dos alternativas más importantes de la actualidad dentro de las bibliotecas de manejo de dispositivos hardware: DirectX y OpenGL.

Se estudiará que son, que características ofrecen dichas librerías y su motivación histórica, ya que hoy en día es impensable el desarrollo de aplicaciones multimedia sin utilizar una librería para manejar los dispositivos gráficos.

2.2.1 DIRECTX

DirectX es un conjunto de Interfaces de Programación de Aplicaciones (o Application Programming Interfaces, API's en habla inglesa) desarrollado por Microsoft.

Son una colección de librerías dinámicas (DLL) que ofrecen funcionalidad relacionada con la programación multimedia de aplicaciones, permitiendo a los programadores manejar gráficos de forma rápida, sonido y funciones de entrada sin tener que programar sus aplicaciones para probar las capacidades de la computadora en la que

su programa se está ejecutando ya que DirectX evalúa dichas capacidades y en caso de no existir DirectX emula (en la mayoría de los casos) dicha funcionalidad utilizando software en vez de hardware.

Origen de DirectX

En tiempos de los sistemas operativos DOS, las personas que programaban aplicaciones multimedia tenían acceso directo al hardware, con completo acceso a interrupciones, tarjetas de sonido, dispositivos de entrada, al buffer de gráficos (VGA), etc. De esta manera los desarrolladores podían hacer que el hardware hiciera aquello que ellos deseaban.

El lanzamiento de Windows 3.1 no tentó a los desarrolladores debido a la gran sobrecarga que suponía para el desarrollo de aplicaciones, de todas maneras DOS tenía también sus contras. El soporte de los dispositivos hardware en DOS se convirtió en una pesadilla para los desarrolladores ya que con el auge de los ordenadores de escritorio en los hogares medios, se produjo mayor competición en los productores de hardware, dando lugar a cientos de configuraciones de PC posible. Ello produjo que se disparara el número de configuraciones que había que programar, luego cada vez se empleaba más tiempo programando el soporte para el hardware de manera que cada vez se tenía menos tiempo para el desarrollo en sí de la aplicación.

Con el lanzamiento de Windows95(C) se introdujeron algunas novedades como el "Plug and Play", y un mejorado manejo de recursos que facilitaba el manejo de dispositivos y lo hacía independiente del dispositivo hardware específico. Desafortunadamente Windows 95 (C) carecía del suficiente rendimiento para hacer a los desarrolladores interesarse por él. De esa manera muchos desarrolladores ejecutaban en modo DOS o requerían un reinicio de la computadora de manera que podían emplear su propio sistema DOS.

El objetivo de hacer Microsoft Windows una plataforma deseable para el desarrollo multimedia se convirtió en una empresa mucho mayor que lo que inicialmente Microsoft(C) pensó. Había mucha determinación en proveer el rendimiento necesario, así que DirectX necesitaría operar a través de los rápidos niveles de librerías binarias que permitieran a los desarrolladores mantener libertad creativa sobre su código.

El siguiente objetivo en los desarrolladores de DirectX era eliminar la carga del soporte hardware de los desarrolladores multimedia y llevarla a los fabricantes de hard-

ware. Esto tiene mucho más sentido ya que los fabricantes de hardware están mucho mejor capacitados para escribir drivers para sus productos que cualquier otro desarrollador de aplicaciones. Ello llevó también a unificar el estándar para la tecnología de drivers, manteniendo los aspectos esenciales de compatibilidad para todos los tipos de componentes de PC adicionales.

Otra característica de DirectX es su capacidad para ejecutar paralelamente con aplicaciones que no utilizan DirectX sin causar problemas al sistema.

Por último decir que DirectX tenía en rendimiento del que se era capaz en DOS mientras se resolvían el resto de especificaciones.

¿Que hace DirectX?

DirectX provee una serie de herramientas y comandos para el desarrollo de juegos y otras aplicaciones multimedia permitiendo al hardware y software comunicarse entre ellos con mucha mayor facilidad. El API permite a las aplicaciones multimedia mayor acceso a características hardware de alto nivel como aceración de los gráficos en 3D. También controla muchas funciones de bajo nivel, esto incluye aceleración de gráficos 2D, soporte para una gran cantidad de dispositivos de entrada (joysticks, mandos, teclado, ratón, etc.); controla también la mezcla del sonido de salida, las comunicaciones y varios formatos de streaming.

Con cada nueva revisión, se añaden nuevas funcionalidades de la tecnología emergente de manera que los desarrolladores puedan empezar a utilizar es nueva tecnología lo más pronto posible.

A continuación se listan los componentes más importantes de DirectX junto con su función relativa:

- DirectDraw: Gráficos 2D
- Direct3D: Gráficos 3D (procesado y programación de gráficos en tres dimensiones).
- DirectSound: Sonido 2D
- DirectSound3D: Sonido 3D
- DirectMusic: Reproducción de música.

- DirectPlay: Para procesado y programación de las comunicaciones en red
- DirectInput: Procesar datos de los dispositivos de entrada

2.2.1.1 DIRECT3D

El objetivo de este componente de DirectX es facilitar el manejo y trazado de entidades gráficas elementales (líneas, polígonos y texturas) en cualquier aplicación que despliegue gráficos en 3D; así como efectuar de forma transparente transformaciones geométricas sobre dichas entidades. Aparte Direct3D provee una interfaz transparente con el hardware de aceleración gráfica por lo cual el desarrollador no tiene que preocuparse de si los gráficos están o no siendo acelerados.

Direct3D está compuesto por dos grandes APIs: el modo retenido y el modo inmediato. El modo inmediato da soporte a todas las primitivas de procesamiento 3D que permiten las tarjetas gráficas (luces, materiales, transformaciones, control de profundidad, etc.); mientras que el modo retenido, que está construido sobre el anterior, presenta una abstracción de nivel superior que ofrece funcionalidades preconstruidas de gráficos tales como jerarquías o animaciones. Debido a que el modo retenido ofrece muy poca libertad a los desarrolladores, es el modo inmediato el más utilizado.

El modo inmediato de Direct3D trabaja fundamentalmente con los dispositivos, que son los encargados de realizar la renderización de la escena. El dispositivo ofrece una interfaz que permite diferentes opciones de renderización. Podemos clasificar los dispositivos en tres clases:

1. Dispositivo HAL: permite la aceleración hardware. Se trata del dispositivo más rápido.
2. Dispositivo de referencia: para su uso es necesaria la instalación previa del SDK de Direct3D. Este dispositivo permite la simulación software de un tipo de renderización, resultando muy útil cuando el hardware todavía no tiene incorporadas nuevas características de renderización.
3. Dispositivo de conexión software: permite opciones de rasterización software. Previamente se ha tenido que obtener el dispositivo.

Cada dispositivo tiene asociadas una o más cadenas de intercambio (Swap chains). Dichas cadenas están compuestas por varios buffers de superficies; hay que considerar cada superficie como el conjunto de píxels que la forman junto con todos los atributos asociados a cada uno de ellos tales como profundidad, color, transparencia, etc.

Los dispositivos tienen también asociados una colección de recursos, los cuales son necesarios para realizar la renderización. Cada recurso tiene los siguientes atributos:

- **Tipo:** define el tipo de recurso del que se trata. Ej. superficie, volumen, textura (normal, de cubo, de volumen, de superfice, etc.), buffer de vértices (vertex buffer), buffer de índices (index buffer).

- **Almacén:** describe dónde se almacena el recurso durante la ejecución. Default (D3DPOOL_DEFAULT) indica que se almacena junto con el dispositivo, en el mejor sitio disponible para un uso lo más rápido posible;
  Managed (D3DPOOL_MANAGED) es el almacén por defecto, indica que se guarda en la memoria del sistema y se copia en el dispositivo cuando éste lo necesita, como ventaja el recurso no tiene que liberado y recreado cuando se pierde el device, además los recursos de este almacén pueden ser bloqueados; System memory (D3DPOOL_SYSTEMMEM) indica que el recurso se encuentra exclusivamente en la memoria del sistema, los recursos de este almacén pueden ser bloqueados y no necesitan ser recreados cuando se pierde el dispositivo; Scratch (D3DPOOL_SCRATCH) tiene las mismas propiedades que System memory, con la diferencia de que los recursos de este almacén no tienen que cumplir las restricciones de las tarjeta gráfica, de igual manera los recursos de este almacén no pueden ser usados durante el rendering, por lo cual normalmente es utilizado para datos de pre-procesamiento del frame.

- **Formato:** describe el formato en que se almacena el recurso en memoria.

- **Uso:** es una lista de flags que indican como se va a usar el recurso. Dinámico (D3DUSAGE_DYNAMIC) indica que el recurso va a ser manipulado regularmente (por ejemplo sobrescrito cada frame) y no puede ser usado si el almacén es Managed (debería usarse con Default), si este flag no se especifíca entonces se entiende que se trata de un recurso estático y que su valor no cambia; Solo escritura (D3DUSAGE_WRITEONLY) solo puede ser especificado para vertex

e index buffers, esto le dice a DirectX que solamente vas a escribir en ellos y nunca a leerlos. Software (D3DUSAGE_SOFTWAREPROCESSING) indica que el procesamiento de los vértices debería hacerse mediante software en lugar de hardware; DepthStencil (D3DUSAGE_DEPTHSTENCIL) indica que el recurso será utilizado como un buffer de patrones de profundidad, y solo puede ser usado si el almacén es Default; RenderTarget (D3DUSAGE_RENDERTARGET) indica que el recurso será una escena renderizada (útil para recrear efectos como espejos etc.), sólo puede ser usado para texturas y superficies y sólo si el almacén es Default.

Por último a continuación se muestra una versión simplificada de la pipeline de Direct3D:

1. **Ensamblaje de entrada:** aporta los datos de entrada (líneas, puntos y triángulos).

2. **Shader de vértices:** se encarga de las operaciones de vértices (iluminación, texturas, transformaciones), tratando cada vértice por separado.

3. **Shader de geometría:** realiza operaciones con entidades primitivas (líneas, triángulos o vértices). A partir de una geometría el shader puede descartarla o devolver una o más primitivas nuevas.

4. **Flujo de salida:** almacena la salida de la etapa anterior en memoria. Resulta útil para realimentar la pipeline con datos ya calculados.

5. **Rasterización:** convierte la imagen 3D en píxels.

6. **Shader de píxeles:** operaciones con los píxels.

7. **Fusión de salida:** se encarga de combinar la salida del shader de píxeles con otros tipos de datos, como los patrones de profundidad, para construir el resultado final.

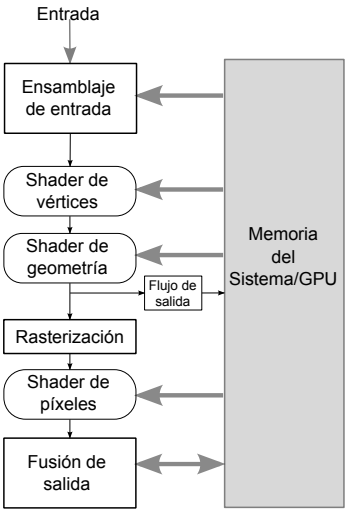


Figura 2.5: D3D pipeline

2.2.2 OpenGL

OpenGL es el acrónimo de Open Graphics Library, librería de gráficos abierta. Como su nombre indica es una especificación estándar que define una API multilenguaje y multiplataforma para el desarrollo de aplicaciones que produzcan gráficos 2D y 3D.

Fundamentalmente OpenGL describe un conjunto de funciones y el comportamiento exacto que deben tener. Partiendo de ella los fabricantes hardware crean implementaciones, que son bibliotecas de funciones que se ajustan a los requisitos de la especificación, utilizando aceleración hardware cuando es posible. Para que los fabricantes puedan calificar su implementación como conforme a OpenGL, dichas implementaciones deben superar unos test de conformidad.

Origen de OpenGL

Durante los 80 era un reto el desarrollo de software que pudiera funcionar con un amplio rango de dispositivos gráficos, ya que los desarrolladores debían escribir las interfaces y drivers para cada dispositivo específico.

A primeros de los 90 la API IRISGL (Integrated Raster Imaging System Graphics Library) de SGI (Silicon Graphics Inc.) poseía gran difusión aparte de estar muy bien

considerada debido a que permitía rendering en modo inmediato y a los desarrolladores les parecía fácil su uso. De esta manera IRISGL se convirtió en un estándar de facto, ensombreciendo otras APIs de estándar abierto como PHIGS (Programmer's Hierarchical Interactive Graphics System).

Los competidores de SGI incorporaron al mercado dispositivos 3D que soportaban una extensión del estándar PHIGS, lo cual debilitó la posición de SGI en el mercado.

Así pues SGI decidió convertir IRISGL en un estándar abierto, con vistas a mantener su influencia en el mercado y a conservar clientes mientras maduraba el soporte para dicho estándar.

Finalmente en 1992 fue lanzado OpenGL. OpenGL no es una liberación de IRISGL debido a restricciones por derechos y patentes, y también a que dicha API soportaba funcionalidades no relacionadas con los gráficos 3D como son el manejo de ventanas o entrada de ratón y teclado.

Propósitos de OpenGL

Hay implementaciones eficientes de OpenGL para Mac OS, Microsoft Windows, Linux, varias plataformas Unix y PlayStation 3. Existen también varias implementaciones en software que permiten ejecutar aplicaciones que dependen de OpenGL sin soporte de aceleración hardware.

OpenGL tiene dos propósitos esenciales:

- Ocultar la complejidad de la interfaz con las diferentes tarjetas gráficas presentando al programador una API única y uniforme.
- Ocultar las diferentes capacidades de las plataformas hardware, requiriendo que todas las implementaciones soporten la funcionalidad completa de OpenGL (utilizando emulación software si fuese necesario)

El funcionamiento básico de OpenGL consiste en aceptar primitivas tales como puntos, líneas y polígonos, convirtiéndolas en píxeles. La mayoría de los comandos de OpenGL o bien permiten primitivas a la pipeline gráfica o bien configuran cómo la pipeline procesa dichas primitivas. Hasta la aparición de la versión 2.0 cada etapa de la pipeline ejecutaba una función prefijada (resultando poco configurable), a partir de dicha versión se permite la programación de algunas etapas de la pipeline gráfica mediante el uso del lenguaje GLSL.

OpenGL describe primitivas de bajo nivel que requiere que el programador dicte los pasos exactos necesarios para renderizar una escena, esto requiere que los programadores conozcan en profundidad la pipeline gráfica a cambio de darles libertad para implementar algoritmos gráficos novedosos. Algunas de estas funcionalidades ofrecidas por OpenGL son:

- Primitivas geométricas y raster: permite utilizar todas las primitivas geométricas básicas (puntos, lineas, polígonos) y del raster (bitmap, imagen).
- B-splines: permiten dibujar líneas curvas.
- Transformaciones de vista y modelo: permiten realizar de forma sencilla translaciones, rotaciones y escalados de objetos dentro de la escena, así como mover la cámara.
- Trabajar con el color: permitiendo utilizar colores en modo RGBA o en modo indexado (de esta forma los colores se seleccionan desde una paleta)
- Eliminación de líneas y superficies ocultas.
- Doble Buffer: permitiendo seleccionar entre el uso de un buffer o dos. El modo doble buffer permite eliminar el parpadeo de las animaciones ya que cuando se está mostrando un frame en el buffer primario el siguiente frame se dibuja en el buffer secundario y cuando está terminado se copia al buffer primario.
- Mapeado de textura.
- Antialiasing: permite suavizar los bordes de polígonos y líneas. Este suavizado se realiza cambiando la intensidad de los píxels adyacentes a la línea que procesamos consiguiendo un efecto de difuminado con la consiguiente eliminación de la sensación de ruptura o zig-zag.
- Luces: permite establecer fuentes de luz, especificando su posición, intensidad, color, tipo, orientación, etc.
- Efectos atmosféricos como niebla o humo.
- Transparencia: permite hacer un objeto transparente o parcialmente transparente de forma que sean total o en parte visibles objetos situados detrás de dicho objeto.

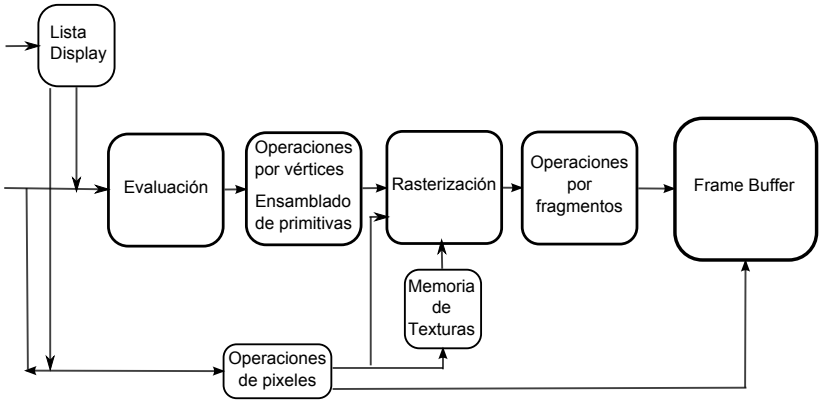


Figura 2.6: OpenGL pipeline

A continuación se realiza una descripción básica y muy general de lo que sería el proceso de pipeline gráfica:

1. Evaluación (si procede) de las funciones polinomiales que definen ciertas entradas, como las superficies NURBS, aproximando curvas y la geometría a la superficie.
2. Operaciones por vértices, transformándolos, iluminándolos según su material y recortando partes no visibles de la escena para producir su volumen de visión.
3. Rasterización (conversión de la información previa en píxeles). Los polígonos son representados con el color adecuado mediante algoritmos de interpolación.
4. Operaciones por fragmentos o segmentos (como actualizaciones) según valores venideros o ya almacenados de profundidad y de combinaciones de colores (entre otros).
5. Volcado de los fragmentos en el Framebuffer.

Como se ha dicho OpenGL es una API para producción de gráficos 2D y 3D mediante primitivas de bajo nivel, por ello junto con OpenGL los desarrolladores utilizan una serie de librerías auxiliares que complementan dicha funcionalidad:

GLU: Esta librería trabaja conjuntamente con OpenGL para ofrecer funciones más complejas como por ejemplo definir un cilindro o un disco con un solo comando, así como funciones para trabajar con splines y realizar operaciones con matrices.

GLUT: Es una librería independiente de la librería OpenGL de cada plataforma. Aunque no incluye funciones adicionales para OpenGL, nos permite utilizar funciones para el tratamiento de ventanas, teclado y ratón. GLUT nos permite crear ventanas y controlar la entrada independientemente de la plataforma utilizada.

OpenAL: Es una librería que nos permite el manejo de audio.

OpenNL: Es una librería diseñada para el manejo de las comunicaciones de red.

OpenIn: Es una librería para manejar los dispositivos de entrada.

2.3 Motor de juegos

En este apartado se explicara que es un motor de juegos y que elementos lo componen.

¿Que es un motor de juegos?

Un motor de juegos es el núcleo Software de un videojuego o de una aplicación que precise el uso de gráficos en tiempo real.

El motor de juegos se encarga de proveer las tecnologías subyacentes al manejo y creación de juegos, por lo que simplifica el desarrollo.

Aunque no hay una definición específica sobre los elementos básicos que debe ofrecer un motor de juegos, a continuación se muestra una lista de los elementos que comúnmente sí componen los motores de juegos del mercado:

- Motor de renderizado
- Grafo de escena
- Detector de colisiones
- Motor de física
- Motor de I.A.
- Motor de Sonidos
- Gestión de Redes
- Scripting

A continuación se detallará brevemente cada uno de estos elementos:

Motor de renderizado

Se encarga de las funciones de renderizado 2D, 3D y de Sprites. Normalmente está apoyado en librerías gráficas, pero dado que existen librerías gráficas que proporcionan la funcionalidad básica de un motor de renderizado (como DirectGraphics, OpenGL y Java3D), y que en ocasiones pueden encontrarse referencias a librerías bajo el término de motor gráfico (o de renderizado), lo que supone un error, debemos saber que una librería gráfica es un conjunto de funciones y algoritmos que mediante la realización de cálculos y transformaciones convierte modelos 3D en una imagen 2D. Pero un motor de renderizado además de lo anteriormente dicho, se encarga de la iluminación, el Antialiasing, el mapeado y la mezcla de texturas y la gestión de mallas entre otras cosas.

Grafo de escena

Es una estructura de datos que se encarga de ordenar la representación lógica de una escena gráfica en forma de árbol, de forma que el efecto de una transformación sobre un nodo padre se produce también en sus hijos, pero no al contrario. Gracias a esta estructura se consigue tratar conjuntos de objetos como si fueran uno solo.

Motor de física

Es el software que simula modelos de física utilizando variables del tipo velocidad, masa, etc. Su objetivo es simular y predecir los efectos que se producen sobre los objetos en caso de tratarse de objetos del mundo real (u objetos de un mundo parametrizado por el programador). El motor de física es cada vez más utilizado debido a su capacidad para aportar realismo al juego.

Detector de colisiones

Es un algoritmo empleado para detectar cuándo dos o más objetos colisionan. La detección se suele realizar por medio del cálculo de intersección de volúmenes simples. Se pueden utilizar distintos volúmenes para envolver el objeto y así detectar mejor las colisiones.

Motor de inteligencia artificial

Es el encargado de dotar a ciertos elementos del juego con un comportamiento pseudointeligente. Por regla general lo que se aplican son técnicas muy simples de

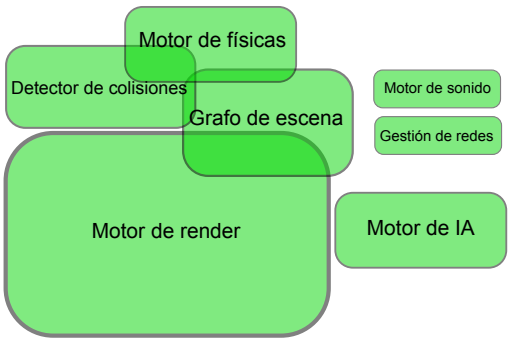


Figura 2.7: Componentes del motor gráfico

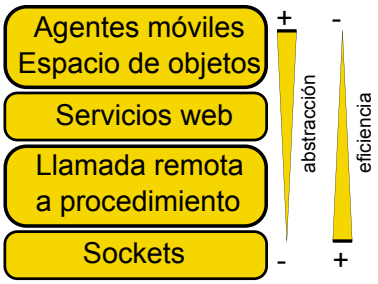


Figura 2.8: Relación eficiencia-abstracción

inteligencia artificial como máquinas de estados o algoritmos de búsquedas, aunque últimamente con objeto de obtener un comportamiento más realista se están empleando algoritmos genéticos y redes neuronales.

Motor de sonido

Es el encargado de reproducir la banda sonora del juego y los efectos de sonido (pisadas, disparos, etc.) en sincronía con el juego.

Gestión de redes

En aquellos juegos que permitan realizar partidas tanto de forma local como en red, el gestor de red tiene la responsabilidad de agrupar todas las utilidades de red para que al usuario le resulte indiferente jugar de forma local o en red.

Las ventajas de uso de motores gráficos es que facilita el desarrollo (ya que como

hemos dicho se encarga de gestionar la escena, colisiones, IA, etc.); si el motor es independiente de la plataforma entonces nuestro juego será también independiente de la plataforma; se realiza una separación entre motor y contenidos, por lo cual un mismo motor puede servir para varios juegos.

Aunque el motor de juegos se compone de muchos elementos, los elementos básicos e imprescindibles son el motor de renderizado y el grafo de escena. El resto de elementos no aparecen en todos los motores, o lo hacen como plug-ins. La elección del motor es crítica si vamos a necesitar alguno de estos elementos.

Cuando en la actualidad se desarrolla un juego, por lo general se parte de un motor de juegos ya existente, ya que el desarrollo es mucho más rápido, no es necesario reinventar la rueda, y económicamente es posible que otra opción no fuera viable.

Motores gráficos del mercado

Existen multitud y gran variedad de motores gráficos en el mercado, pero podemos dividirlos en dos tipos: Motores comerciales y motores open source.

La ventaja de utilizar un motor comercial es que se dispone de mejor soporte y herramientas adicionales respecto a los motores open source, pero tiene el inconveniente de que su coste es mucho mayor y la funcionalidad es cerrada.

La ventaja de utilizar un motor open source es que normalmente no tiene coste, o es más bajo en relación con los motores comerciales, además los motores open source pueden modificarse; tienen el inconveniente de que no hay un soporte oficial (en la mayoría de casos es una comunidad que ofrece su apoyo a través de un foro), y escasean las herramientas para el desarrollo de elementos del juego.

Ejemplos de motores comerciales son: Torque, TrueVision3D y 3D Game Studio.

Ejemplos de motores open source son: OGRE, Irrlicht y Crystal Space.

Se han escogido tres motores de juegos de entre todos disponibles en el mercado (que son muchos) en base a tres características: su funcionalidad, la variedad de lenguajes en las que pueden programarse y si pueden usarse de forma gratuita. Los tres motores resultantes han sido: TrueVision3D, Irrlicht y OGRE.

A continuación se describen las características de cada motor.

TrueVision3D

Es un motor de juego comercial escrito en VisualBasic 6 y posteriormente reescrito C++. El precio de su licencia son 150 dólares americanos si lo vamos a utilizar en un único título, y 500 dólares americanos si se va a utilizar en varios títulos, pero la versión prerelease se puede obtener de forma gratuita, de forma que puede utilizarse libremente, con el inconveniente de que aparece el logotipo de TrueVision3D en la esquina de la pantalla.

TrueVision3D funciona exclusivamente con DirectX 8, pero la prerelease soporta DirectX 9 (pero no con DirectX 10 ni XNA), y es compatible con los lenguajes de programación C++, C#, Delphi, VisualBasic 6 y VisualBasic .NET.

Herramientas:

- Editor de Shader
- Visor de modelos
- Plugins de exportación para: 3D Studio Max, Maya and Milkshape3D
- Editor de efectos de partículas
- Varios conversores

Funcionalidades:

- Soporte para HLSL Shading
- Soporta el renderizado de terrenos por partes, y su deformación en tiempo real por medio de arrays o puntos de actualización.
- Permite la creación en tiempo de ejecución de mayas
- Soporte para animaciones tipo morph y esqueleto
- Iluminación a nivel de vértice o de píxel
- Efectos especiales
- Motor de física integrado
- Motor de Audio integrado

- Motor de video integrado
- Motor de Red integrado

Irrlicht

Irrlicht es un motor de juegos orientado a objetos y escrito en lenguaje C++ que oficialmente puede utilizarse para las plataformas: Windows, Mac OS X, Linux y Windows Compact Edition.

Irrlicht se distribuye bajo la licenzia Zlib, tratándose de un software libre y de código abierto.

Funcionalidades:

- Permite renderizar usando Direct3D y OpenGL entre otros
- Existen extensiones para usar Irrlicht de forma compatible con otros lenguajes como: .NET, Java, Perl, Ruby, Python, Lua y Delphi
- Ofrece dos motores de renderizado software, uno orientado a conseguir velocidad en el renderizado y otro orientado a lograr calidad en el renderizado
- Un sistema de Interfaz de usuario 2D incorporado.
- Importa directamente modelos en formato: Maya (.obj), 3DStudio (.3ds), COLLADA (.dae), Milkshape (.ms3d) y DirectX (.X) entre otros
- Importa directamente texturas en formato: Windows Bitmap (.bmp), Portable Network Graphics (.png), Adobe Photoshop (.psd), JPEG (.jpg) y TrueVision-Targa (.tga) entre otros
- Posee un rápido y sencillo sistema de detección de colisiones
- Integra un rápido parseador XML
- Manejo de escenarios de interior y exterior de forma indiferente así como el uso de nodos de escena, mayas, cargadores de textura y elementos de Interfaz de usuario propios
- Uso de animaciones Morph y de esqueleto
- Permite Pixel y Vertex Shaders, así como el uso de scripts HLSL y GLSL

OGRE

OGRE (Object-Oriented Graphics Rendenring Engine) como dice su nombres es un motor gráfico orientado a objetos. OGRE también incorpora de forma nativa un grafo de escena y detección de colisiones y a lo largo de su existencia han ido apareciendo multitud de plugins para integrar en OGRE funcionalidades para operaciones con física de objetos, sonidos e interfaz de usuario entre otras; es por ello que ahora estemos tratando OGRE como un motor de juegos.

OGRE está escrito en lenguaje C++, y actualmente existen distribuciones para Linux, MaC OSX y Windows, distribuyéndose bajo licencia LGPL.

Funcionalidades:

- Permite renderizar con OpenGL o DirectX
- Evalúa de forma automática los requisitos de renderizado
- Lenguaje de definición de materiales que permite el mantenimiento de materiales de forma independiente al código
- Soporte de Vertex y Pixel Shading a bajo nivel y mediante scripts Cg, HLSL o GLSL
- Soporta operaciones con texturas múltiples y la mezcla de texturas entre otras operaciones con texturas
- Importacion de texturas en formato: Windows Bitmap (.bmp), Portable Network Graphics (.png), JPEG (.jpg), TrueVisionTarga (.tga) o DirectDraw Surface (.dds).
- Uso de animaciones Morph y de esqueleto, además de animaciones de nodos de escenario para la realización de vuelos de cámara y técnicas similares
- Manejo de escenas de forma sencillo y personalizable
- Uso de efectos especiales

2.4 Lenguaje de programación

De un lenguaje de programación debemos valorar los siguientes aspectos:

- Eficiencia y rendimiento: es la cantidad de recursos (principalmente tiempo de procesador y espacio de memoria) que generalmente se requiere para ejecutar un programa en un lenguaje de programación.
- Fiabilidad: es la capacidad del lenguaje de prevenir que errores de propagación o por conversión de datos generen resultados
- Su robusted: es la capacidad del lenguaje de anticiparse a situaciones de conflicto por incompatibilidad entre tipos de datos y su capacidad para el manejo de excepciones.
- Su portabilidad: es el rango de arquitecturas y sistemas operativos sobre el cual el código puede ser compilado (o interpretado) y ejecutado.
- Su escalabilidad: cómo de fácil es incluir nueva funcionalidad o de aumentar la funcionalidad del código existente.
- Como se realiza la gestión de memoria, si se realiza de forma automática o es el programador quien la gestiona.
- La facilidad de desarrollo de programas.
- La facilidad de corregir errores.
- La adecuación el paradigma al que pertenece al problema que queremos resolver.
- La existencia de herramientas que ayuden al desarrollo.

A continuación examinamos los lenguajes utilizados comúnmente en el desarrollo de juegos junto con sus ventajas e inconvenientes:

Ensamblador

El lenguaje ensamblador tiene la gran ventaja de que ofrece una gran eficiencia y un alto rendimiento de sus programas debido a que utiliza elementos de muy bajo nivel. El problema es que es un lenguaje complicado, es prácticamente un arte y su uso para desarrollo de programas complejos requiere mucho tiempo, además cuando no se es un experimentado programador de lenguaje ensamblador se tiende a producir muchos errores en el código, lo cual unido a la falta de robustez y fiabilidad del lenguaje además de la lo complejo que es la depuración llevan al programador a estar más tiempo depurando errores que programando. Por otro lado, otro gran inconveniente es que el lenguaje ensamblador es específico de la arquitectura del ordenador, lo cual obligaría que tuviera que programarse el juego para cada una de las distintas arquitecturas. Por último decir que la gestión de memoria se realiza a mano por parte del programador, el cual debe reservar y liberar los espacios en la pila de memoria; además si se quiere incluir nueva funcionalidad en una función, el programador necesitará tomarse su tiempo para recordar (o averiguar si se trata de otro programador diferente) que es lo que se está haciendo ya que el código ensamblador no es muy legible.

C

C es un lenguaje de gran calidad que sigue siendo muy usado frente al auge de los nuevos lenguajes orientados a objetos. C pertenece a los lenguajes de paradigma estructural y es un lenguaje que ha sido ampliamente utilizado para el desarrollo de juegos. Además de tener la ventaja de ser portable a casi cualquier arquitectura y sistema operativo conocido, posee una alta eficiencia y rendimiento frente a los lenguajes más nuevos, por no hablar de la extensa gama de herramientas y librerías disponibles para este lenguaje (ya que como se ha dicho lleva utilizándose mucho tiempo para el desarrollo de juegos). Entre las desventajas de usar C está en primer lugar que no tiene manejo de errores ni de excepciones, lo cual puede llevar a una propagación de errores que termine con la parada del programa; por otra parte, y es el mayor problema para los programadores, es que el manejo de memoria en C es manual, el programador se encarga de reservar el espacio de memoria para las nuevas estructuras y arrays, y debe encargarse de liberarlos cuando ya no se utilicen o de lo contrario se producirán memory leaks, es decir, fallos por falta de memoria libre en el sistema. Aunque los memory leaks no son culpa del lenguaje sino del programador, su depuración es complicada y hace

que se tenga muy en cuenta a la hora de elegir o no C como el lenguaje para el juego. Existen herramientas que ayudan al depurado de errores en C, y también a la gestión del código, lo cual delega el problema de la escalabilidad en cómo el programador ordena, nombra y almacena las funciones, estructuras y librerías.

C++

Digamos que se trata (como su nombre indica) del siguiente paso en la evolución del lenguaje C. Básicamente se trata del mismo lenguaje pero con funcionalidad añadida: funciones virtuales, tipos genéricos, la posibilidad de declarar variables en cualquier punto del programa, el uso de clases y la herencia múltiple. Es un lenguaje portable a un gran número de arquitecturas y sistema operativos, un lenguaje eficiente que tiene un alto rendimiento. Debido a que también es uno de los lenguajes más utilizados en el desarrollo de juegos, existe una amplia gama de herramientas y librerías disponibles. Además como ventaja frente a C, C++ posee herramientas para la captura y el manejo de excepciones, y también se trata de un lenguaje fuertemente tipado lo cual nos ayudará a evitar propagación de errores por fallos en conversión de tipo. Pero en C++ sigue existiendo el problema de los memory leaks ya que el manejo de memoria se lleva a cabo de forma manual. Existen numerosas herramientas que ayudan al desarrollo y depuración de programas en C++. Por último decir que C++ se trata de un lenguaje más escalable que C debido a que pertenece la paradigma de orientación a objetos.

Java

Java surgió a principios de los 90 como búsqueda de un lenguaje para dispositivos electrónicos (lavadoras, microondas, etc.). Java comparte muchas características con C++: declarar funciones en cualquier punto del programas, uso de clases y herencia, funciones virtuales, etc. Pero Java rompe con algunos elementos de los llamados característicos de C y C++ como son el uso de punteros, las variables globales, la sentencia 'goto', y la conversión de tipos insegura. Además frente al manejo manual de memoria realizado en C y C++, Java incorpora un manejo automático por lo cual el programador ya no debe encargarse reservar la memoria y liberarla cuando ya no la necesite, sino que existe un proceso llamado 'recolector de basura' que cuando determina que la memoria ya no va a poder ser accedida la libera de forma automática. De esta forma Java simplifica el desarrollo de aplicaciones al no tener que preocuparse el programador

por el manejo de punteros, los errores por conversión de datos, o por el manejo de la memoria; además Java es un lenguaje robusto con un fuerte tipado de objetos, con herramientas para el manejo y depuración de errores, y gracias a su fuerte orientación a objetos gran facilidad para el escalado de la aplicación gracias a los paquetes (encapsulaciones de clases). Otra novedad respecto a C y C++ es que los programas Java se ejecutan sobre una máquina virtual llamada JVM (JavaVirtual Maquine, en castellano máquina virtual de Java), por lo que una vez se ha compilado el código java a objeto bytecode, no vuelve a compilarse aunque se cambie el objeto de arquitectura o sistema operativo, lo cual hace que la instalación de programas Java sea más rápida. El uso de la JVM hace el programa Java tenga que interpretarse en tiempo de ejecución, y esto hace que sea más lento que programas escritos en C o C++, es un motivo por el cual muchas veces se descarta Java frente a otros lenguajes cuando se trata de juegos destinados a computadores personales; sin embargo el mismo motivo hace a Java perfecto para funcionar en dispositivos portables como PDAs o móviles, siendo Java el principal lenguaje de programación de juegos para móviles. Existen muchas herramientas que ofrecen un entorno de desarrollo integrado de proyectos con el lenguaje Java, las más conocidas son NetBeans y Eclipse.

C#

C# Es otro lenguaje que hereda de C (o mejor dicho C++), pero digamos que no extiende de C sino que se basa en él y también en Java. C# fusiona la capacidad de combinar operadores de C++ (sin soportar herencia múltiple) con la orientación a objetos de Java. C# es el lenguaje estrella del entorno .NET lanzado por Microsoft en el año 2000, dicho entorno actúa de máquina virtual, ya que el código en C# es compilado a código CLR (Common Language Runtime), y cuando se ejecuta el programa, la máquina virtual de .NET es la encargada de ejecutar el código CRL. Durante la ejecución del programa algunas partes son compiladas durante el momento de ejecución, y pese a poder dar la impresión de lo contrario no se ralentiza la ejecución del programa; sin embargo si es verdad que el entorno .NET consume bastantes recursos. Tanto el lenguaje como el entorno .NET son un estándar abierto, por lo cual los programas en C# no son exclusivos de la plataforma .NET de Windows, pero no existen muchas plataformas que implementen dichos estándares (la más conocida es la plataforma MONO para Linux), por lo cual aunque estrictamente es un lenguaje

portable, de facto solo existen dos alternativas donde ejecutarlo. C# es un lenguaje robusto, fuertemente tipado, con una gestión automática de memoria gracias al recolector de basura, con herramientas para el manejo y depuración de errores, y su orientación a objetos permite que los programas sean fácilmente escalables y fáciles de desarrollar, en concreto las aplicaciones de ventana ofrecen mucha funcionalidad y manejo frente a otros lenguajes. Entre las herramientas de entorno de desarrollo integrado encontramos el Visual Studio que ofrece Microsoft, y también la herramienta gratuita SharpDevelop.

Visual Basic

Visual Basic, es una extensión de BASIC con el objetivo de permitir la creación de interfaces gráficas. Hoy en día se encuentra en desuso debido a la aparición de Visual Basic .NET (con el cual existen incompatibilidades) y C#, por lo cual se prima el uso de dichos lenguajes en lugar de Visual Basic.

Eiffel, Smalltalk, ADA

Aunque se trata de lenguajes empleados en el desarrollo de juegos, son lenguajes no usados comúnmente por los desarrolladores, ya que precisan de un alto grado de conocimiento experto tanto del lenguaje como del desarrollo de juegos, y además existen pocas herramientas que ayuden al desarrollo de juegos con estos lenguajes.

Lua, Python

Son lenguajes de scripting, normalmente no se utilizan para desarrollar el núcleo de programación del juego donde el aspecto de rendimiento sea importante, sino más bien scripts o librerías que no se utilizan de forma continua sino de manera ocasional.

2.5 Elección de la tecnología para las comunicaciones

En este apartado se explicarán los mecanismos de comunicación que utilizaremos en el proyecto.

Tecnologías de comunicación

Socket

Es un método de comunicación entre un programa cliente y un programa servidor. Típicamente el término socket refiere al extremo de una comunicación. Para realizar la comunicación el cliente debe conocer la localización exacta del programa servidor, determinada normalmente por su dirección IP, un protocolo de transporte y un puerto; conocida su localización el programa cliente podrá establecer la comunicación y llevar a cabo un intercambio de octetos con el servidor, finalmente una vez terminada la comunicación se cierra la conexión.

El que la comunicación sea orientada a conexión, y que se garantice el orden y la correcta llegada de los octetos no es responsabilidad de los sockets, sino del protocolo de transporte elegido para la comunicación haciendo de los socket una forma simple y versátil de comunicar procesos.

La mayoría de programas tienen APIs para la programación de sockets.

Llamada remota de métodos

Es un mecanismo que permite ejecutar funciones que pueden estar en una máquina distinta a la que realiza la invocación. El proceso de invocación remota puede variar según la tecnología empleada pero básicamente consiste en tres pasos: un primer paso es averiguar la localización del recurso remoto; una vez localizado, el siguiente paso es solicitar la ejecución de la función remota (lo cual puede requerir que se envíen los parámetros de esta), y por último obtener el resultado de la ejecución de la función.

Existen varias herramientas que permiten llevar a cabo este tipo de operaciones, ejemplos de ellos son los RPC (Remote Procedure Call o llamada de procedimiento remoto) de Sun, RMI (Remote Method Invocation) de Java y CORBA (Common Object Request Broker Architecture) de OMG (Object Management Group); que aunque de forma general podría decirse que los tres realizan una comunicación mediante llamada remota de un método, existen notables diferencias tanto en la forma de llevarlas a cabo como en que consiste el elemento remoto al que se llama.

RPC: En los RPCs el elemento remoto es una función que se encuentra alojada como un servicio en una máquina, además existe un servicio en la máquina que actúa de registro de las funciones remotas que se alojan en ella, por lo que cuando un proceso quiere hacer una llamada de procedimiento (o función) remoto primero

acude al registro de la máquina para obtener el puerto en el cual está el servicio específico de ese procedimiento (o función) y posteriormente realiza la llamada remota a ese puerto.

RMI: En RMI el elemento remoto es un objeto, por lo cual lo que se hace realmente es ejecutar un método de un objeto remoto. Para que los demás procesos puedan acceder al objeto remoto debe declararse una interfaz de métodos remotos, posteriormente una implementación de dicha interfaz y de esta última generar un resguardo. El resguardo es el representante local del objeto remoto, y se invocan sus métodos como si de un objeto local se tratara, cuando en realidad interiormente se realiza la llamada al objeto remoto y se obtiene el resultado. La instancia del resguardo debe obtenerse de un registro (llamado rmiregistry) en el cual se ha guardado previamente el resguardo junto con una URL que la identifica, por lo cual para obtener el resguardo el cliente debe saber la dirección IP y puerto del rmiregistry y la URL de registro.

CORBA: En CORBA el elemento remoto es también un objeto, pero la forma de invocación es un tanto especial ya que mientras que en RMI hay un resguardo al que se realiza la llamada como si fuera una instancia de un objeto local, en CORBA se tiene un ORB (Object Request Broker) que actúa como intermediario o middleware, lo cual permite a la aplicación acceder potencialmente a varios objetos remotos (o locales). Además el ORB puede actuar como mediador para objetos heterogéneos permitiendo la interacción entre objetos implementados sobre diferentes plataformas. Para poder ser accedidas, las referencias a objetos remotos necesitan estar asociadas a un nombre; el Servicio de Nombres es el encargado de mantener una base de datos con los nombres y los objetos asociados a dichos nombres.

Espacio de objetos

Este mecanismo consiste en que los participantes de una aplicación convergen en una entidad lógica llamada espacio de objetos en el cual un suministrador coloca objetos como entidades dentro de un espacio de objetos y los solicitantes que se subscriben al espacio pueden acceder a dichas entidades. Este mecanismo oculta los detalles referentes a la búsqueda de recursos u objetos que eran explícitos en otros mecanismos, y además

garantiza la exclusión mutua porque un objeto en el espacio solo puede ser usado por un participante a la vez.

Un conjunto de herramientas que se basan en el concepto de espacio de objetos es JavaSpaces de Java.

Agentes móviles

Un agente móvil es un programa u objeto transportable. Este mecanismo consiste en que un agente lanzado desde un determinado ordenador viaja de forma automática de un ordenador a otro de acuerdo con un itinerario que posee, y en cada parada el agente accede a los recursos o servicios necesarios y realiza las tareas correspondientes para completar su misión. Mediante este mecanismo en lugar de intercambiar los datos mediante mensajes los datos son transportados por el programa/objeto mientras que el objeto se transfiere entre los participantes.

El sistema Concordia y Aglet ofrecen soporte para agentes móviles.

Servicios Web

Podría decirse que un servicio web es una API que puede ser accedida a través de una red (como Internet) y ejecutada en un computador remoto que aloja el servicio solicitado. Podría pues considerarse como otra forma de llamada a procedimiento remoto, pero la novedad que ofrecen los servicios web es que dichos servicios son independientes del sistema operativo del servidor en que se alojan y del lenguaje de programación que se ha utilizado para implementarlo.

Para que semejante grado de interoperabilidad sea posible, es necesario que el proceso solicitante del servicio web y el proceso que ofrece el servicio acuerden un protocolo mediante el cual se define como se invoca el servicio, como se pasan los parámetros, como se recibe un resultado, como se manejan errores, etc.; típicamente esta descripción se ha venido realizando mediante el protocolo SOAP (Simple Object Access Protocol) aunque existen otros protocolos para la descripción de servicio.

SOAP está diseñado para llevar a cabo intercambios de información en XML para sistemas altamente distribuidos. SOAP utiliza un lenguaje 'neutro' llamado WSDL (Web Services Description Language) para describir las funcionalidades del servidor, las especificaciones de servicios descritas mediante WSDL funcionan como un contrato que las aplicaciones clientes deben cumplir para utilizar el servicio. Esta descripción

neutra de los servicios web permite por lo tanto no solo aislar el lenguaje específico de la implementación del servidor, sino también de los clientes.

Además, mediante el uso de UDDI (Universal Description, Discovery and Integration Directory) se pueden publicar servicios web de forma centralizada en un directorio, de forma que otros usuarios puedan conocer la existencia de dicho servicio.

En SOAP, los datos a transmitir se serializan con XML, de forma que se salvan las incompatibilidades que pudieran existir entre otros protocolos de representación de datos; y los mensajes transmitidos normalmente se envían utilizando otros protocolos de aplicación (típicamente http, ya que permite una enviar mensajes a través de proxies y firewall de forma más sencilla), los cuales se encargan de la negociación y la transmisión.

Las ventajas de los servicios web es que son independientes de la plataforma, del lenguaje, y que si se utiliza sobre HTTP permite una fácil comunicación a través de proxies y firewalls.

Las desventajas es que al estar los mensajes serializados con XML, el parseo de dichos mensajes provoca que los servicios web sean más lentos que otros mecanismos como CORBA o RMI. Esta diferencia no es muy apreciable cuando se trata de mensajes pequeños, pero a medida que el tamaño de datos de los mensajes aumenta, el tiempo de resolución de la petición aumenta considerablemente frente a los otros mecanismos citados. Además, al utilizar el servicio sobre HTTP obliga a que la operación sea obligatoriamente one-way o request-response; haciendo imposible operaciones del tipo solicit-response o notification.

Comparativa de abstracción/eficiencia

A continuación se realizará un análisis de los mecanismos descritos anteriormente en base a su nivel de abstracción y su eficiencia.

La eficiencia de un método de comunicación viene dada por la rapidez con que se lleva a cabo dicha comunicación. Por ejemplo un socket es más eficiente que una invocación remota de método ya que un socket directamente envía los octetos de información, mientras que con la invocación remota se pasa dicha información al resguardo, que la prepara y posteriormente la envía.

La abstracción de un método de comunicación viene dado por la cantidad de detalles que se ocultan al programador. Con el mismo ejemplo anterior, los sockets poseen menor abstracción que una invocación remota, ya que mientras que con la invocación remota

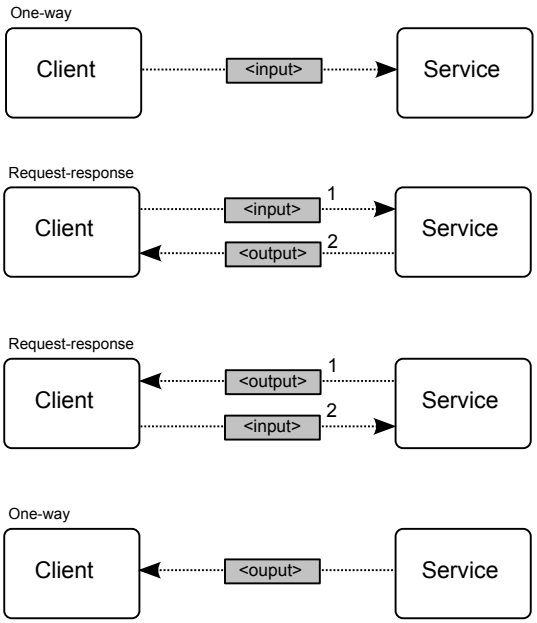


Figura 2.9: Operaciones de un servicio web

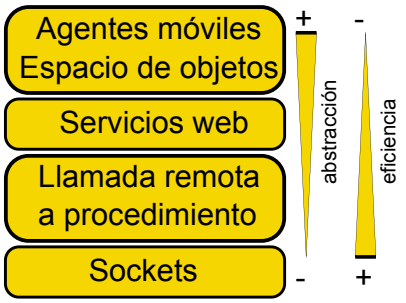


Figura 2.10: Relación eficiencia-abtrascción

el programador simplemente ejecuta un método en el resguardo (el cual internamente coge los parámetros, los empaqueta y envía al objeto remoto, y posteriormente obtiene el resultado) como si se tratara de una invocación de método normal; con los sockets el programador debe guardar la información a enviar en un array de bytes, mandar dicho array (asegurándose de que se envía entero), luego debe esperar la respuesta que será almacenada en otro array de bytes, y por último debe interpretar los datos recibidos.

En la figura 2.10 puede apreciarse la relación entre los distintos mecanismos de comunicación, y su relación eficiencia/abstracción:

Cuando se elija el método de comunicación, debe buscarse un equilibrio dentro de la relación eficiencia/abstracción ya que un mecanismo muy eficiente requerirá de mucho control mediante código, y uno muy abstracto puede ralentizar las comunicaciones de forma que el usuario se percate de ello y crea que el juego no funciona.

Comparativa de escalabilidad

Cuando se habla de escalabilidad de un mecanismo de comunicación, no se hace referencia a la facilidad con que el mecanismo puede adaptarse al aumento de carga, sino a la facilidad con la que el mecanismo de comunicación permite añadir o eliminar nueva funcionalidad. Por ejemplo se tiene una calculadora distribuida que permite hacer sumas, y posteriormente se desea que además de sumas puedan hacerse restas y multiplicaciones; un mecanismo poco escalable requerirá muchos cambios en el código de la calculadora para poder añadir las nuevas funcionalidades, mientras que un mecanismo muy escalable requerirá menos cambios.

Dentro de los mecanismos de comunicación descritos, la escalabilidad es prácticamente la misma excepto en el caso de los sockets que es menor. La razón es que mientras que para los otros mecanismos, añadir funcionalidad consiste en añadir un nuevo método o un nuevo objeto, para el caso de los sockets añadir funcionalidad significa introducir cambios en la función de control de las comunicaciones, lo cual no es mucho problema cuando la funcionalidad es poca; pensando en el caso de la calculadora, añadir a la funcionalidad de suma, la de resta y multiplicación no tiene mucha complicación, pero si seguimos añadiendo funcionalidad (derivadas, integrales, división, cambios de base, operaciones con matrices, etc.) llega un punto en el que sería muy difícil si quisiéramos eliminar por ejemplo la funcionalidad de suma.

Modelos de comunicación

Antes de abordar los modelos de comunicación se intentará ilustrar cuál es la diferencia entre un modelo de comunicación y un mecanismo de comunicación. Mientras que un mecanismo de comunicación es la forma en la que se lleva a cabo la comunicación (de forma verbal, por carta, mediante MORSE, etc.); mientras que un modelo de comunicación establece el tipo de relación que existe entre las partes integrantes de la comunicación (jefe y empleado, amigos, cliente y dependiente, etc.).

Existen multitud de modelos de comunicación, pero estudiaremos solamente los dos más importantes: el modelo cliente-servidor y el modelo peer-to-peer.

Modelo Cliente-Servidor

Una relación cliente-servidor consiste en un programa, denominado cliente (de servicio), que hace una petición de servicio a otro programa, denominado servidor (de servicio). Este es el modelo de funcionamiento de herramientas muy conocidas como el envío de correo electrónico y la navegación web.

En este modelo, cada programa cliente puede enviar peticiones a uno o más programas servidores, los cuales a su vez pueden aceptar estas peticiones, procesarlas y enviar las respuestas a los programas clientes.

Por lo tanto cada interlocutor asume un rol: uno actúa como proveedor y otro como solicitante, por lo cual toda la información y funcionalidad está contenida en el servidor.

Modelo Peer-to-peer

Una comunicación peer-to-peer funciona básicamente como cualquier comunicación cliente-servidor: un programa envía una petición a otro programa, el cual resuelve la petición y le envía la respuesta.

La principal diferencia entre una comunicación cliente-servidor y una comunicación peer-to-peer es que no hay distinción entre clientes y servidores, no hay roles dentro de los programas, por lo cual del mismo modo que el programa A envía una petición al programa B, podría ser el programa B quien enviara la misma petición al programa A; de tal modo todos los elementos de un modelo peer-to-peer son iguales y ofrecen los mismos servicios.

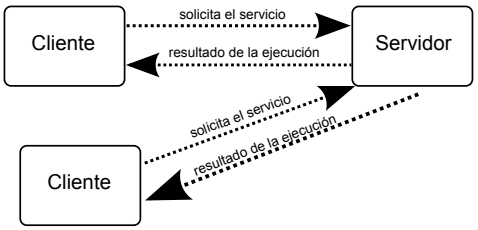


Figura 2.11: Modelo cliente-servidor

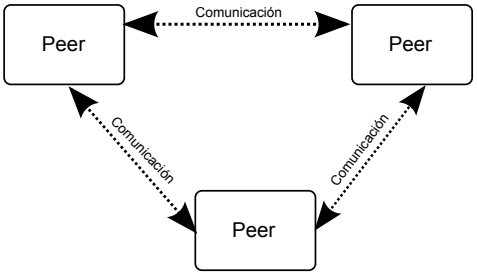


Figura 2.12: Modelo peer-to-peer

Otra diferencia entre una comunicación cliente-servidor y otra peer-to-peer es que mientras en la primera el servidor es el que tiene toda la información (y por ello los clientes son los únicos que hacen peticiones), en la segunda la información está distribuida entre los distintos peers.

Para ilustrar mejor estas diferencias puede pensarse en la relación banco-cliente (el banco sería el servidor), en una relación banco-cliente el banco tiene toda la información acerca de las cuentas del cliente, su saldo, sus beneficios, etc. y el cliente solo sabe lo que le dice el banco, si el cliente pide su saldo entonces sabe cual es el saldo en el instante en que lo solicitó pero no puede por ejemplo decirle al banco que saldo quiere tener; de igual manera pensemos ahora en la relación banco-banco, ambos tienen clientes, un banco A puede pedirle a un banco B el cobro de un recibo a cargo de un cliente de A y viceversa.

3

Análisis

Este capítulo se desarrollará la fase de análisis del sistema, que servirá como base para el diseño detallado del sistema.

Por lo tanto lo que se busca en el análisis no es tanto resolver cómo se va a implementar el juego, sino identificar que acciones necesitamos que realice el juego y expresar dichas necesidades de forma explícita.

Expresar mediante el análisis la funcionalidad del juego permitirá que una vez se haya realizado el diseño, pueda comprobarse fácilmente que por cada elemento del análisis existe al menos un elemento del diseño destinado a implementar dicha funcionalidad, reduciendo así el riesgo de que olvidarse de la implementación de algún aspecto del juego.

3.1 Diagramas de clase

En esta sección se mostrarán los diagramas de clase resultante del análisis tanto de la parte servidor como de la parte cliente de la aplicación.

3.1.1 Diagrama de clases de la parte servidor

La figura 3.1 muestra el diagrama de clases del servidor.

3.1.2 Diagrama de clases de la parte cliente

La figura 3.2 muestra el diagrama de clases de la parte cliente.

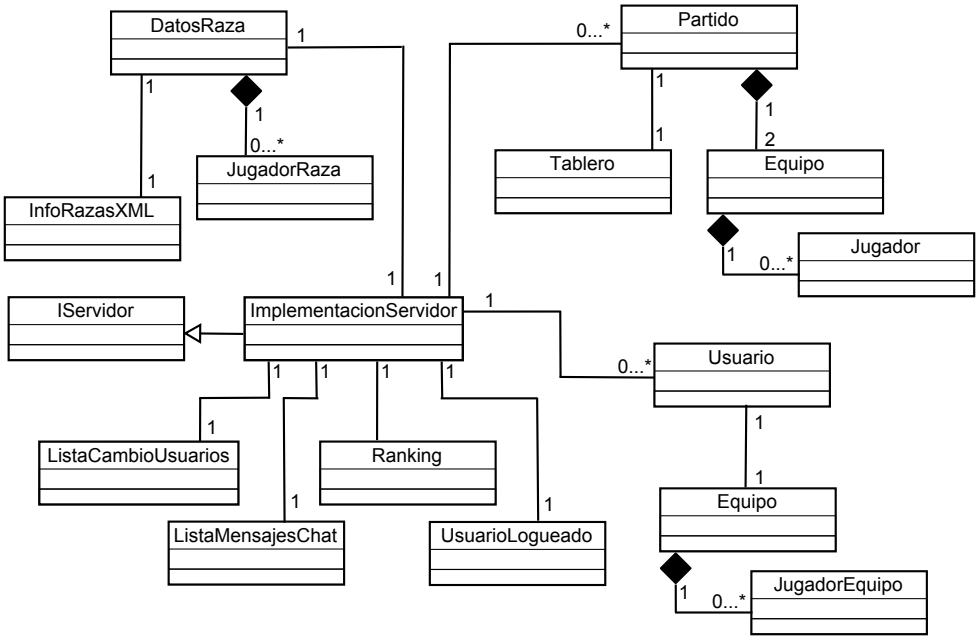


Figura 3.1: Diagrama de clases del servidor

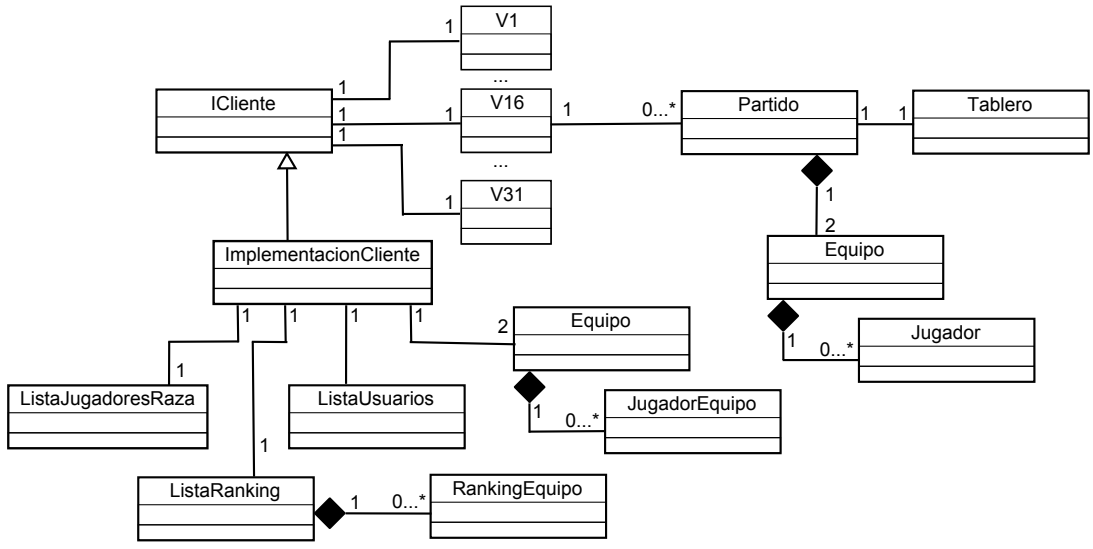


Figura 3.2: Diagrama de clases de la parte cliente

ID	CU-XYZ	**NOMBRE**	Información que describa el caso de uso de forma breve.
ACTORES		Actor o actores involucrados en la situación descrita.	
OBJETIVOS		Qué pretende conseguir el caso de uso.	
ESCENARIO		Secuencia de acciones que debe realizar el actor para conseguir el resultado esperado.	
PRECONDICIONES		Todas aquellas condiciones necesarias que deben cumplirse para poder realizar el caso de uso.	
POSTCONDICIONES		Aquellas condiciones que se producen al efectuar el caso de uso.	
COND. DE FALLO		Condiciones que afectan al escenario y las respuestas del sistema en caso de no poder realizarse el propósito descrito.	

Tabla 3.1: Modelo tabla casos de uso.

3.2 Casos de uso

En este apartado se realizarán unos casos de uso que describan cuál es la interacción entre los usuarios y la aplicación.

Estos casos de uso se presentarán de manera gráfica y de forma textual, con el objetivo de proporcionar toda la información posible para lograr así una correcta interpretación de los mismos.

La identificación de dichos casos de uso se realizará por su nombre y por un atributo identificador único. En la representación gráfica se podrá ver una imagen distintiva de una persona que interacciona con el sistema (en adelante nos referiremos a él como *actor*); y enmarcadas dentro de un globo las acciones que puede realizar. En la descripción textual de los casos de uso tomará como formato la tabla mostrada a continuación, en la cual se rellenarán los diversos puntos especificados:

Como puede observarse en la tabla 3.1, los casos de uso se identificarán por las letras "CU-" seguidas de un número único de tres dígitos empezando por "001" que se incrementará de forma secuencial para servir de identificador a los siguientes casos de

uso.

Como puede verse, en el sistema existen dos tipos de actores: el usuario sin loguear y el usuario logueado. El usuario logueado es un usuario que se ha identificado frente al servidor por medio de su Nick y su contraseña; un usuario sin loguear es un usuario que no se ha identificado frente al servidor, y por lo tanto se desconoce su identidad.

A continuación vamos a proceder a la descripción textual de los casos de uso.

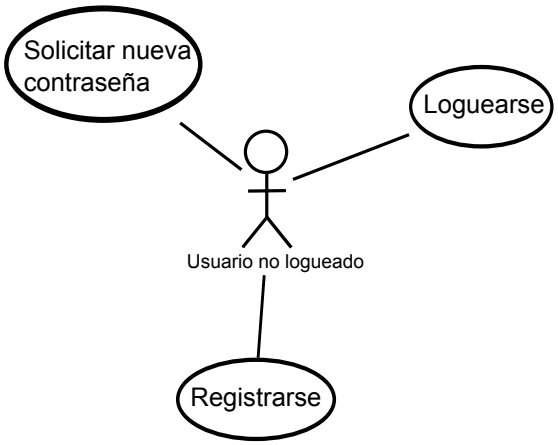


Figura 3.3: Casos de uso de usuario no logueado

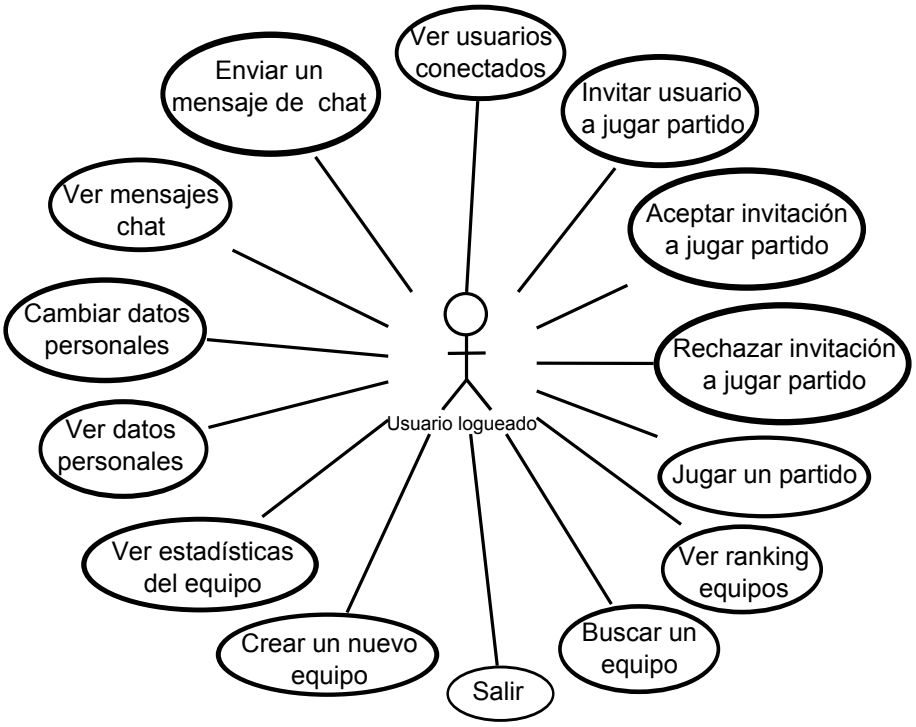


Figura 3.4: Casos de uso de usuario logueado

ID	CU-001	**NOMBRE**	Registro de nuevo usuario.
ACTORES		Usuario no logueado.	
OBJETIVOS		El objetivo es que usuario cree una nueva cuenta en el sistema.	
ESCENARIO		1. El actor debe navegar hasta la pantalla de crear una nueva cuenta. 2. Debe rellenar los datos de Nick, contraseña, repetir contraseña, nombre, apellidos y email. 3. Pulsar el botón enviar para mandar la solicitud de creación de cuenta. 4. Se notifica al usuario que la cuenta ha sido creada y se vuelve a la pantalla de login.	
PRECONDICIONES		Ninguna.	
POSTCONDICIONES		Se ha creado la cuenta de usuario con la información especificada.	
COND. DE FALLO		Si se da el caso de que la contraseña y su repetición sean distintas, o que el texto del email no sea una dirección válida, o que se haya dejado algún campo vacío o que ya exista el Nick especificado, entonces no se procederá a la creación de la cuenta y se informará al usuario de el error o errores que se han producido.	

Tabla 3.2: Caso de uso CU-001

ID	CU-002	**NOMBRE**	Solicitar una nueva contraseña.
ACTORES		Usuario no logueado.	
OBJETIVOS		El objetivo es que un usuario que ha olvidado su contraseña reciba por email una nueva contraseña de acceso al sistema.	
ESCENARIO		1. El actor debe navegar hasta la pantalla de solicitar nueva contraseña. 2. Introduce su Nick de usuario del sistema. 3. Pulsa el botón enviar.	
PRECONDICIONES		Ninguna.	
POSTCONDICIONES		Al usuario se le ha asignado una nueva contraseña y se le ha enviado a su correo electrónico.	
COND. DE FALLO		Si se ha producido algún error al intentar hacer el cambio de contraseña, o al enviar la contraseña por email, o el Nick especificado no existe; entonces se notifica al usuario que se ha producido un error y no se procede al cambio y envío de la nueva contraseña.	

Tabla 3.3: Caso de uso CU-002

ID	CU-003	**NOMBRE**	Loguearse en el sistema.
ACTORES		Usuario no logueado.	
OBJETIVOS		El objetivo es que el usuario se autentique en el sistema y pueda acceder a las funcionalidades de usuario logueado.	
ESCENARIO		1. El usuario se encuentra en la ventana de login. 2. Introduce su Nick y su contraseña. 3. Pulsa el botón entrar.	
PRECONDICIONES		Ninguna.	
POSTCONDICIONES		El usuario queda logueado en el sistema y pasa a visualizarse la página principal de la aplicación.	
COND. DE FALLO		Si falla la autenticación del usuario, bien porque el Nick es incorrecto o bien porque la contraseña sea incorrecta, entonces se comunica al usuario que ha fallado la autenticación y no se loguea al usuario.	

Tabla 3.4: Caso de uso CU-003

ID	CU-004	**NOMBRE**	Ver mensajes de chat.
ACTORES		Usuario logueado.	
OBJETIVOS		El objetivo es visualizar los mensajes de chat que se han producido desde que el usuario inició sesión.	
ESCENARIO		1. El usuario navega hasta la ventana de chat.	
PRECONDICIONES		El usuario debe estar logueado, pero no debe estar jugando un partido, ni esperando la respuesta o a la espera de responder a una invitación a jugar un partido.	
POSTCONDICIONES		Ninguna.	
COND. DE FALLO		Ninguna.	

Tabla 3.5: Caso de uso CU-004

ID	CU-005	**NOMBRE**	Enviar un mensaje de chat.
ACTORES		Usuario logueado.	
OBJETIVOS		El objetivo es que el usuario escriba una cadena de texto que es enviada a todos los usuarios conectados y se muestra en la pantalla de chat.	
ESCENARIO		1. El usuario navega hasta la ventana de chat. 2. El usuario escribe el texto del mensaje. 3. Pulsa el botón enviar.	
PRECONDICIONES		El usuario debe estar logueado, pero no debe estar jugando un partido, ni esperando la respuesta o a la espera de responder a una invitación a jugar un partido.	
POSTCONDICIONES		El texto del mensaje se visualiza en la pantalla de chat de todos los usuarios conectados.	
COND. DE FALLO		Si el texto del mensaje tiene longitud cero, entonces no se procede a su envío, ni se notifica al usuario que no se ha enviado.	

Tabla 3.6: Caso de uso CU-005

ID	CU-006	**NOMBRE**	Ver la lista de los usuarios conectados.
ACTORES		Usuario logueado.	
OBJETIVOS		El objetivo es visualizar en un listado los Nicks de los usuarios que están actualmente logueados en la aplicación.	
ESCENARIO		1. El usuario navega hasta la ventana de chat.	
PRECONDICIONES		El usuario debe estar logueado, pero no debe estar jugando un partido, ni esperando la respuesta o a la espera de responder a una invitación a jugar un partido.	
POSTCONDICIONES		Ninguna.	
COND. DE FALLO		Ninguna.	

Tabla 3.7: Caso de uso CU-006

ID	CU-007	**NOMBRE**	Invitar a otro usuario a jugar un partido.
ACTORES		Usuario logueado.	
OBJETIVOS		El objetivo es enviar a otro usuario una invitación para jugar un partido de rugby.	
ESCENARIO		1. El usuario navega hasta la ventana de chat. 2. Seleccionar el Nick del usuario al que desea invitar. 3. Seleccionar el tipo de juego (oficial o amistoso) que desea jugar. 4. Pulsar el botón Invitar.	
PRECONDICIONES		El usuario debe estar logueado, pero no debe estar jugando un partido, ni esperando la respuesta o a la espera de responder a una invitación a jugar un partido.	
POSTCONDICIONES		El usuario se queda bloqueado hasta obtener la respuesta del usuario invitado.	
COND. DE FALLO		Si no se ha elegido ningún Nick de la lista de conectados, o no se ha elegido el tipo de partido entonces no se procede a enviar la invitación, de igual manera tampoco se le notifica de ninguna forma al usuario que no se ha hecho. Si el Nick elegido de la lista de conectados es el del propio jugador que realiza la invitación, o uno de los dos usuarios (tanto el que realiza la invitación como el invitado) no tiene creado ningún equipo, se rechaza automáticamente la invitación. Si el usuario al que se ha invitado está esperando la respuesta a una invitación, o tiene pendiente una invitación que responder, o está jugando un partido entonces se rechaza automáticamente la invitación.	

Tabla 3.8: Caso de uso CU-007

ID	CU-008	**NOMBRE**	Aceptar invitación a jugar partido.
ACTORES		Usuario logueado.	
OBJETIVOS		El objetivo es aceptar la oferta de juego y pasar a jugar un partido de rugby.	
ESCENARIO		1. Pulsar Aceptar en la ventana de invitación al partido.	
PRECONDICIONES		El usuario debe estar logueado, pero no debe estar jugando un partido, ni esperando la respuesta o a la espera de responder a una invitación a jugar un partido. Otro usuario logueado ha invitado a este usuario a jugar un partido.	
POSTCONDICIONES		Al usuario que realizó la invitación se le informa que ha sido declinada. Ambos usuarios (el que realizó la invitación y el que ha aceptado) pasan a la ventana de partida de rugby.	
COND. DE FALLO		Si ha pasado el tiempo de espera de la invitación, entonces se volverá a la pantalla previa a la invitación como si se hubiera rechazado.	

Tabla 3.9: Caso de uso CU-008

ID	CU-009	NOMBRE	Rechazar invitación a jugar partido.
ACTORES		Usuario logueado.	
OBJETIVOS		El objetivo es rechazar la oferta de juego y volver a la ventana en la que se estaba antes de que llegara la invitación para jugar el partido.	
ESCENARIO		1. Pulsar Declinar en la ventana de invitación al partido.	
PRECONDICIONES		El usuario debe estar logueado, pero no debe estar jugando un partido, ni esperando la respuesta o a la espera de responder a una invitación a jugar un partido. Otro usuario logueado ha invitado a este usuario a jugar un partido.	
POSTCONDICIONES		Al usuario que realizó la invitación se le informa que ha sido declinada.	
COND. DE FALLO		Ninguna.	

Tabla 3.10: Caso de uso CU-009

ID	CU-010	**NOMBRE**	Jugar un partido de rugby.
ACTORES		Usuario logueado.	
OBJETIVOS		El objetivo es jugar un partido de rugby.	
ESCENARIO		Ninguno.	
PRECONDICIONES		El jugador debe haber aceptado previamente una invitación a jugar un partido, o haber realizado una invitación a jugar un partido que hayan aceptado.	
POSTCONDICIONES		El jugador pasa a la ventana de juego de partidos.	
COND. DE FALLO		Ninguna.	

Tabla 3.11: Caso de uso CU-010

ID	CU-011	**NOMBRE**	Ver ranking equipos.
ACTORES		Usuario logueado.	
OBJETIVOS		El objetivo es ver el ranking de equipos donde se muestra información del entrenador, el puesto, las victorias, derrotas, empates, valoración, raza y nombre del equipo.	
ESCENARIO		1. Navegar hasta la pantalla de ranking.	
PRECONDICIONES		El usuario debe estar logueado, pero no debe estar jugando un partido, ni esperando la respuesta o a la espera de responder a una invitación a jugar un partido.	
POSTCONDICIONES		Ninguna.	
COND. DE FALLO		Ninguna.	

Tabla 3.12: Caso de uso CU-011

ID	CU-012	**NOMBRE**	Buscar un equipo.
ACTORES		Usuario logueado.	
OBJETIVOS		El objetivo es obtener un listado que indique el puesto, el nombre del entrenador, el nombre del equipo, la raza, las victorias, los empates, las derrotas y la valoración de los equipos que cumplan los criterios de búsqueda.	
ESCENARIO		1. Navegar hasta la pantalla de ranking. 2. Introducir al menos uno de los siguientes parámetros de búsqueda: (a) Puesto. (b) Entrenador. (c) Equipo. (d) Raza. (e) Victorias. (f) Derrotas. (g) Empates. (h) Valoración. 3. Pulsar el botón Buscar.	
PRECONDICIONES		El usuario debe estar logueado, pero no debe estar jugando un partido, ni esperando la respuesta o a la espera de responder a una invitación a jugar un partido.	
POSTCONDICIONES		Se muestran en pantalla un listado con los equipos que cumplen las especificaciones.	
COND. DE FALLO		Si no se introduce ningún criterio de búsqueda no se envía la petición de búsqueda y no se muestra ningún resultado. Si el tipo de datos del valor introducido en un criterio de búsqueda no coincide con el tipo de datos del criterio de búsqueda entonces se muestra un aviso en pantalla.	

Tabla 3.13: Caso de uso CU-012

ID	CU-013	**NOMBRE**	Salir de la aplicación.
ACTORES		Usuario logueado.	
OBJETIVOS		El objetivo es terminar la sesión en el sistema y cerrar la aplicación.	
ESCENARIO		1. Pulsar el botón Salir.	
PRECONDICIONES		El usuario debe estar logueado, pero no debe estar jugando un partido, ni esperando la respuesta o a la espera de responder a una invitación a jugar un partido.	
POSTCONDICIONES		El usuario deja de estar logueado en la aplicación. Se cierran todas las ventanas.	
COND. DE FALLO		Ninguna.	

Tabla 3.14: Caso de uso CU-013

ID	CU-014	NOMBRE	Crear un nuevo equipo.
ACTORES		Usuario logueado.	
OBJETIVOS		El objetivo es crear un equipo nuevo para el jugador.	
ESCENARIO		1. Navegar hasta la pantalla de estadísticas y equipo. 2. Pulsar el botón de nuevo equipo. 3. Aceptar la confirmación. 4. Introducir el nombre del equipo y elegir la raza. 5. Pulsar seguir. 6. Comprar los jugadores, las segundas oportunidades de equipo y el factor de hinchas. 7. Pulsar hecho.	
PRECONDICIONES		El usuario debe estar logueado, pero no debe estar jugando un partido, ni esperando la respuesta o a la espera de responder a una invitación a jugar un partido.	
POSTCONDICIONES		Se ha creado el equipo según los datos especificados a lo largo de los pasos descritos en el escenario. Cualquier equipo que el jugador pudiera tener anteriormente se ha borrado.	
COND. DE FALLO		Si en el paso 3 del escenario se declina la confirmación, se vuelve a la pantalla de estadísticas y equipo. Si en el paso 4 se introduce un nombre de equipo que ya existe se vuelve a la pantalla de estadísticas y equipo.	

Tabla 3.15: Caso de uso CU-014

ID	CU-015	**NOMBRE**	Ver estadísticas del equipo.
ACTORES		Usuario logueado.	
OBJETIVOS		El objetivo es ver los datos del equipo del usuario: nombre, raza, victorias, derrotas, empates, valoración, dinero, segundas oportunidades y factor de hincas; así como los jugadores del equipo y su ficha de información.	
ESCENARIO		1. Navegar hasta la pantalla de estadísticas y equipo.	
PRECONDICIONES		El usuario debe haber creado al menos un equipo con anterioridad. El usuario debe estar logueado, pero no debe estar jugando un partido, ni esperando la respuesta o a la espera de responder a una invitación a jugar un partido.	
POSTCONDICIONES		Ninguna.	
COND. DE FALLO		Ninguna.	

Tabla 3.16: Caso de uso CU-015

ID	CU-016	**NOMBRE**	Ver datos personales.
ACTORES		Usuario logueado.	
OBJETIVOS		El objetivo es ver los datos del usuario de la cuenta: Nick, nombre, apellidos y email.	
ESCENARIO		1. Navegar hasta la pantalla de Datos cuenta.	
PRECONDICIONES		El usuario debe estar logueado, pero no debe estar jugando un partido, ni esperando la respuesta o a la espera de responder a una invitación a jugar un partido.	
POSTCONDICIONES		Ninguna.	
COND. DE FALLO		Ninguna.	

Tabla 3.17: Caso de uso CU-016

ID	CU-017	**NOMBRE**	Cambiar datos personales.
ACTORES		Usuario logueado.	
OBJETIVOS		El objetivo es modificar al menos uno de los elementos de la cuenta: contraseña, nombre, apellidos o email.	
ESCENARIO		1. Navegar hasta la pantalla de estadísticas y equipo.	
PRECONDICIONES		El usuario debe estar logueado, pero no debe estar jugando un partido, ni esperando la respuesta o a la espera de responder a una invitación a jugar un partido.	
POSTCONDICIONES		Los datos que no estaban vacíos se han cambiado.	
COND. DE FALLO		En el caso que la antigua contraseña sea incorrecta, la nueva contraseña y su repetición no coincidan o el formato de la dirección email sea incorrecto, se le notifica al usuario y se cancela el proceso de modificar los datos de la cuenta.	

Tabla 3.18: Caso de uso CU-017

3.3 Elección de la tecnología a utilizar

En este apartado se elegirán cuales de los componentes descritos a lo largo del capítulo 2, serán utilizados en el proyecto.

3.3.1 Elección del programa de modelado

Para el tema que nos atañe, las opciones que mejor se adecúan a nuestro objetivo de diseñar los elementos del juego de rugby es el modelado mediante mayas poligonales o mediante NURBS. Ya que en el apartado 2.1 se han expuesto las ventajas e inconvenientes de cada uno, se ha decidido utilizar la representación mediante maya poligonal. El motivo es que a pesar de sacrificar el acabado realista que obtendríamos con NURBS, el objeto modelado es más que aceptable y su manejo en tiempo de cómputo en ejecución es mucho más eficiente que utilizando NURBS.

Se precisa además que el programa permita modelar objetos de tres dimesiones de tipo cáscara mediante mayas poligonales; con posibilidad de aplicar texturas UV, Bump y stencil; y que además permita crear animaciones de esqueleto.

Los programas que se detallaron en el apartado 2.2 son 3D Studio Max, MilkShape 3D y Blender. En una primera comparativa puede observarse que excepto por la cantidad de formatos soportados, MilkShape 3D ofrece mucha menos funcionalidad que 3D Studio Max o Blender, por lo cual la discusión sobre el programa de modelado a elegir se centrará en estos dos últimos.

3D Studio Max es una herramienta más completa, con soporte profesional y de la que existen multitud de libros, y con una licencia que cuesta 80 euros; por otro lado Blender no cuenta con soporte alguno, aunque si posee un manual de uso en la página oficial y multitud de tutoriales por internet.

Dado que el interés es académico, y que no es indispensable el uso de Software propietario, se utilizará Blender para el modelado de los elementos del juego.

3.3.2 Elección de la biblioteca de manejo de gráficos

En el apartado 2.2 nos hemos adentrado en el estudio de las dos principales librerías para el manejo de hardware existentes en la actualidad: DirectX y OpenGL. En ambos casos el objetivo de la librería es abstraer al desarrollador de las características específicas

de los dispositivos hardware para ofrecerle una serie de funciones que puedan usar independientemente del tipo de dispositivo que lo ejecute por debajo.

Como se ha visto, mientras que OpenGL ofrece un estándar para la generación de gráficos 2D y 3D, DirectX ofrece un API para los dispositivos tanto gráficos, como de sonido, comunicaciones, etc., siendo Direct3D es la parte de DirectX encargada del manejo de los gráficos 3D.

De ambos se ha visto de forma muy general cual es el pipeline para la renderización de gráficos, lo cual ayudará a entender cómo funcionan por debajo los motores gráficos.

En el caso de Direct3D se ha prestado también atención a los diferentes tipos de dispositivos que utiliza así como a los atributos de los recursos asociados a estos, y aunque parezca que lo mencionado sobre ellos no será de utilidad a la hora de desarrollar el juego (puesto que el desarrollo está a un mucho más alto nivel de abstracción), este conocimiento marca la diferencia entre saber y desconocer, poder y no poder realizar funciones y efectos complejos por medio del motor gráfico.

Una ventaja notable de OpenGL es su independencia sobre la plataforma, ya que DirectX solo funciona sobre Windows; pero también es cierto que la mayoría de los hogares que tienen ordenadores, utilizan este sistema operativo. También es una ventana de OpenGL el permitir la programación de algunas etapas de pipeline, cosa que no permite Direct3D (que solo permite reconfigurar su pipeline). Pero DirectX ofrece además de manejo de gráficos manejo de sonido, entradas, etc.; mientras que con OpenGL debemos recurrir a librerías externas para manejar el sonido, ventanas y demás elementos, lo cual limita la portabilidad de OpenGL a la de los sistemas para los que dichas librerías están disponibles. Por este motivo se utilizará DirectX como librería para el manejo de hardware.

3.3.3 Elección del motor de juego

En el apartado 2.3 se presentaron tres motores de juego: TrueVision 3D, Irrlitch y OGRE. Ahora debe elegirse cuál de los tres motores será el utilizdo en el juego.

El primero que eliminamos es TrueVision ya que no es portable mientras que Irrlicht y OGRE si lo son. Así que ahora debemos decidir si utilizar Irrlicht u OGRE, que a primera vista son bastante parecidos, pero OGRE cuenta con una comunidad más amplia y tras evaluar los ejemplos de ambos motores se ve como OGRE ofrece una

mayor calidad de imagen, por lo cual se elige OGRE como el motor de juego que utilizaremos.

3.3.4 Elección del lenguaje de programación

Una vez vistas los posibles lenguajes en los que puede programarse el juego, las mejores opciones son C, C++, Java y C#.

Así pues hay que elegir entre dos lenguajes con los que es más laborioso el desarrollo pero con los que los programas obtenidos son muy eficientes como son C y C++, y dos lenguajes que facilitan mucho el desarrollo pero cuya eficiencia es menor como son Java y C#. Para resolver esta disyuntiva se ha preferido primer a los lenguajes orientados a objetos porque ofrecen una mayor claridad sobre la distribución de la funcionalidad y mayor escalabilidad.

Una vez eliminados C y C++, la elección debe realizarse entre Java o C#. Ambos lenguajes ejecutan sobre una máquina virtual, y ambos cuentan con unos entornos de desarrollo muy avanzados; NetBeans y Eclipse sirven para el desarrollo de aplicaciones Java, ambos son gratuitos y quizá la diferencia más notable entre ellos es que Eclipse no permite la edición de formularios mientras que NetBeans si, y que NetBeans requiere gran cantidad de recursos para su ejecución frente a Eclipse; Visual Studio es la herramienta de pago de Microsoft para el desarrollo de programas para .NET (no solo mediante el lenguaje C#) aunque pueden conseguirse licencias gratuitas para estudiantes, y SharpDevelop es la herramienta gratuita disponible tanto para Windows como para Linux.

Java es el lenguaje utilizado en juegos para móviles y PDAs, mientras que C# (aunque puede utilizarse en móviles mediante el .NET Compact Framework en sistemas Windows Mobile) su uso es más comúnmente utilizado para el desarrollo de juegos para PC y XBox (mediante el framework XNA).

Como hemos dicho, C# permite un fácil desarrollo y configuración de formularios, y en este aspecto gana puntos frente a Java. Por lo cual elegimos C# como el lenguaje que utilizaremos para el desarrollo del juego.

3.3.5 Elección del mecanismo de comunicación

Para implementar las comunicaciones del proyecto podrían usarse cualquiera de los mecanismos descritos en el apartado 2.5, sin embargo para facilitar la implementación

la elección del mecanismo estará guiada por los criterios de escalabilidad, eficiencia y abstracción descritos.

El criterio de escalabilidad elimina el mecanismo de socket de entre los posibles mecanismos a utilizar.

Guiados por el criterio de eficiencia se elimina la posibilidad de elegir los mecanismos de espacio de objetos y agentes móviles, de tal manera que hay que elegir entre el mecanismo de llamada remota de procedimientos y de servicios web.

La diferencia más significativa entre los servicios web y la llamada a procedimiento remoto es que los servicios web son independientes de la plataforma y del lenguaje, lo cual facilita la interoperabilidad cuando existen elementos heterogéneos en la comunicación. Dado que este no es el caso de nuestro proyecto, se empleará pues la llamada a procedimiento remoto dado que ofrece una mayor eficiencia.

3.3.6 Elección del modelo de comunicación

La elección del modelo de comunicación a utilizar resulta sencilla, ya que en nuestro proyecto existen dos roles diferenciados, por un lado está el servidor que contiene la información de las cuentas, los equipos, las normas de juego, los partidos, etc. y por otro está el usuario, que se conecta al servidor para enviar mensajes, jugar partidas, etc..

No podemos considerar que exista una relación de iguales entre ambos, sino de cliente y servidor, que será el modelo a utilizar en las comunicaciones del proyecto.

3.4 Comunicaciones

Cuando en el apartado 3.3.5 se ha discutido el mecanismo de comunicación a utilizar, donde se eligió la llamada a procedimientos remotos como mecanismo de comunicación a utilizar para el proyecto.

Existen múltiples implementaciones de dichos mecanismos para distintos lenguajes como java, o que permitan incluso abstraerse del lenguaje en que se implementa la llamada remota, pero dado que el lenguaje de programación que se utilizará en la implementación será C#, se utilizará .NET Remoting, que es la herramienta que proporciona el entorno .NET para realizar llamadas a procedimientos remotos.

3.4.1 Introducción a la tecnología .NET Remoting

Aunque se van a explicar los conceptos de .NET Remoting, y la forma en que se crean los objetos remotos, este capítulo no está destinado a ser un tutorial sobre .NET Remoting. Ello quiere decir que los detalles técnicos de la tecnología resultarán extraños al lector que no esté familiarizado con su uso, pero esta carencia se suplirá siempre por medio de una descripción general de la funcionalidad, con la cual dicho lector podrá hacerse una idea básica de su funcionamiento.

¿Que es .NET Remoting?

Es una herramienta integrada en el entorno .NET con el objetivo de permitir construir aplicaciones distribuidas de forma sencilla, sin importar que los componentes de la aplicación estén todos en el mismo equipo, repartidos entre los equipos de una red local o en equipos en distintas redes.

Al estar integrada dentro del Framework .NET es independiente del lenguaje pudiendo utilizarse con Visual Basic .NET, C#, Eiffel, Smalltalk, etc.

Elementos de .NET Remoting:

- **Canales**: son los elementos que permiten el transporte de datos desde y hasta los objetos remotos, toda invocación y su correspondiente respuesta viaja por esta vía. Los canales más utilizados son el canal TCP y el canal HTTP: el canal TCP tiene mucho mayor rendimiento ya que se conecta directamente a un socket indicado; el canal HTTP funciona de manera más lenta pero al ir la información encapsulada como HTTP, se pueden establecer comunicaciones a través de firewalls.

- **Serialización**: para poder realizar la transferencia de los objetos por los canales, estos deben ser previamente transformados en un formato que pueda ser transportado e identificado desde el otro sitio, es el proceso que se conoce como serialización. Existen dos formateadores predefinidos en .Net Remoting: SOAP o Binario. De forma predeterminada, el formateador SOAP se utiliza con canales HTTP, y el formateador Binario con canales TCP.

- **Objetos Proxy**: Cuando un cliente crea una instancia de un objeto remoto recibe un proxy (o resguardo) de la clase instanciada en el servidor, de forma que para el cliente usar el objeto remoto no es muy diferente de usar un objeto local.

- **Paso de objetos**: Los objetos creados remotamente deben heredar de MarshalByRefObject. Los objetos pasados como parámetros pueden ser pasador por valor o por referencia.
- **Tiempo de vida**: en .Net Remoting un objeto es controlado por un mecanismo de leasing, que establece un tiempo permitido de uso, cuando dicho tiempo se acaba el objeto es desconectado y será destruido por el recolector de basura.
- **Modelos de activación**: Existen tres formas de activar un objeto con .NET.
  - *SingleCall*: por cada invocación recibida, el servidor crea una instancia del objeto remoto para resolver dicha invocación, y una vez resuelta destruye el objeto.
  - *Singleton*: se utiliza el mismo objeto para resolver todas las invocaciones de forma que se comparte la información entre las distintas invocaciones. El objeto remoto no se instanciará hasta recibir la primera invocación.
  - *Client-Activated Objects*: esta modalidad permite a la aplicación instanciar el objeto para su uso exclusivo. Cuando el cliente crea una instancia utilizando el operador "new" un pedido de activación es enviado al servidor. Luego el servidor crea el objeto y devuelve una referencia al mismo.

4

Diseño

En este capítulo se explicará la manera en que se han implementado e incorporado las reglas del juego en el sistema.

También se explicará como se ha realizado el diseño de los modelos 3D.

4.1 Reglas del partido

Para crear el reglamento de un partido de SpaceRugby se ha partido del reglamento de BloodBowl, que puede obtenerse en (2).

SpaceRugby utiliza todas las reglas básicas del reglamento de BloodBowl, y algunas reglas opcionales tales como:

- El movimiento ¡A por ellos!
- Apoyar un placaje.
- Intercepciones y balones perdidos.
- Las siguientes habilidades: Abrirse paso, Agallas, Atrapar, Brazos adicionales, Cola prensil, Defensa, Dos cabezas, Escabullirse, Esquivar, Mano grande, Manos seguras, Mantenerse firme, Nervios de acero, Pasar, Pase seguro, Piernas largas, Pies firmes, Placaje defensivo, Placar, Precisión, Profesional, Saltar y Tentáculos.

4.1.1 Secuencia del partido

En la figura 4.1 está representado el diagrama de flujo de lo que sería un partido en SpaceRugby. En primer lugar el servidor determina cuál es el equipo lanzador y cuál es el equipo receptor, a continuación el equipo lanzador coloca en primer lugar y posteriormente coloca el equipo receptor.

Después de que ambos equipos hayan colocado a sus jugadores, el equipo lanzador realiza la patada inicial, y tras la patada inicial comienza la sucesión de turnos de acción de los equipos. El primer equipo en recibir el turno de movimiento es el equipo lanzador y cuando éste finaliza, el turno pasa al equipo receptor; una vez que termina el equipo receptor el turno vuelve a ser del equipo lanzador y así sucesivamente hasta que ambos equipos hayan jugado ocho turnos cada uno.

En ese momento se llega a la media parte, en la media parte ambos equipo recuperan todos sus puntos de segunda oportunidad (al valor inicial del partido) y se realizan tiradas para ver si los jugadores inconscientes se recuperan. Al finalizar la media parte los roles de equipo lanzador y equipo receptor se intercambian y se procede a comenzar la segunda parte del partido. Una vez finalizada la media parte el equipo lanzador (que era el equipo receptor de la primera parte) coloca los jugadores, a continuación es el equipo receptor quién coloca a sus jugadores.

Una vez ambos equipos han colocado a sus jugadores el equipo lanzador realiza la patada inicial y comienza la sucesión de turnos de acción de la segunda parte. El primero en poseer el turno de acción es el equipo lanzador, cuando el equipo lanzador finaliza el turno comienza el turno del equipo receptor, cuando el equipo receptor termina su turno, éste vuelve al equipo lanzador y así sucesivamente hasta que ambos equipos hayan jugado ocho turnos cada uno en esta segunda parte del partido, en ése momento el partido se da por terminado y se va al postpartido.

El postpartido consiste en una serie de pasos que muestran al usuario los efectos o resultados del partido en su equipo. En primer lugar se muestra el resultado del partido, cuantos touchdowns ha marcado cada equipo y cuál ha sido el equipo ganador (salvo que se haya producido un empate). A continuación se muestra la recaudación del encuentro que ha efectuado el equipo.

Después de la recaudación se muestra la tabla de heridas de los jugadores del equipo. En dicha tabla se muestra el nombre del jugador, el tipo de herida que ha recibido (que

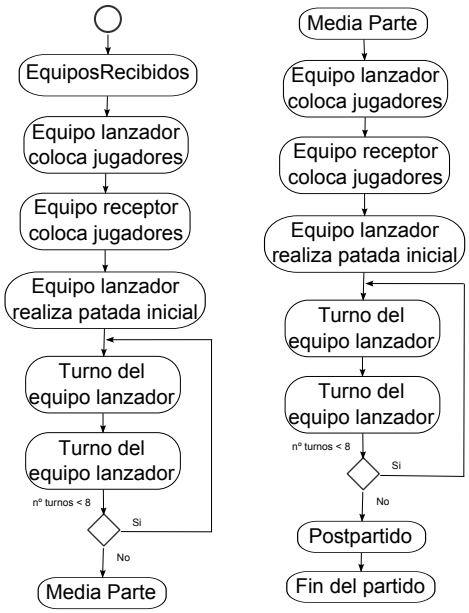


Figura 4.1: Secuecia del partido

puede ser leve, grave o muerto) y el efecto de la herida en el jugador. Cuando el jugador ha recibido una herida leve el efecto producido es que dicho jugador no percibe puntos de experiencia por el partido, si la herida es grave además de no percibir puntos de experiencia el jugador no podrá disputar el siguiente partido de juegue el equipo, si el jugador resulta muerto obviamente el jugador no podrá disputar ya ningún partido.

A continuación de la tabla de heridas se muestra el factor de hinchas del equipo resultante del partido. Por último el postpartido finaliza con la ventana de compra de nuevos jugadores. Una vez se ha terminado el postpartido se vuelve a la pantalla de Chat de la aplicación.

Cuando un equipo (llámese A) marca un touchdown, se finaliza inmediatamente el turno de acción y los jugadores dejan el campo. En el caso que ambos equipos hayan jugado 8 turnos cada uno se procedería a la media parte en el caso que el touchdown se marque en la primera parte, o al postpartido si es que el touchdown se ha marcado en la segunda parte. En el caso que alguno de los equipos no haya alcanzado los 8 turnos jugados en la parte, el equipo que encajó el touchdown (B) procede a colocar sus jugadores en el campo. A continuación es el equipo A quien coloca sus jugadores;

y por último el equipo B realiza la patada inicial, procediendo a ser el turno del equipo B. Una vez que el equipo B comienza su turno, se prosigue con la sucesión de turnos hasta que ambos equipos han jugado 8 turnos cada uno.

Si durante un turno un equipo decide abandonar el partido, se va directamente a la parte de postpartido.

4.1.1.1 Los estados del partido en la parte servidor y cliente

En el apartado anterior se ha definido la secuencia del partido. La forma en la que la parte servidor y la parte cliente controlan el momento del partido se encuentran es por medio de una máquina de estados.

La máquina de estados que utiliza la parte servidor es prácticamente una traducción de la secuencia del partido ilustrada en la imagen 4.1, sin embargo la máquina de estados de la parte cliente resulta más interesante pues su forma de actuar depende de si el equipo es el equipo lanzador o el equipo receptor. En la figura 4.2 se ilustra la máquina de estados de la parte cliente. En el estado inicial se esperan los equipos que van a jugar el partido (con la lista de sus respectivos jugadores). Una vez se reciben, se espera a que se determine si se está en el caso del equipo receptor o del equipo lanzador, como dijimos en el apartado anterior el equipo lanzador coloca en primer lugar (por ello si se es equipo receptor se esperan las posiciones del equipo lanzador) y es quien realiza la patada inicial (el equipo receptor después de colocar sus jugadores pasa directamente a esperar el rebote del balón. Una vez recibido el rebote del balón, los equipos deben esperar a que se determine si es o no su turno comenzando de esta manera la primera parte del partido; cuando ambos han jugado sus 8 turnos, se llega a la mitad del partido y se repite todo el proceso de colocar a los jugadores, realizar la patada inicial y turno de acción como en la primera parte. Cuando termina la segunda parte se pasa a la fase de postpartido, y cuando se ha terminado el postpartido se llega al estado final, el fin del partido.

4.1.1.2 La secuencia de un turno

En este apartado se explicará la secuencia que se lleva a cabo dentro del turno de un equipo, que podemos ver de forma resumida en la figura 4.3.

En un turno lo primero que debe hacerse es elegir el jugador con el que se quiere actuar. Cuando se selecciona un jugador se le solicita al servidor las acciones que

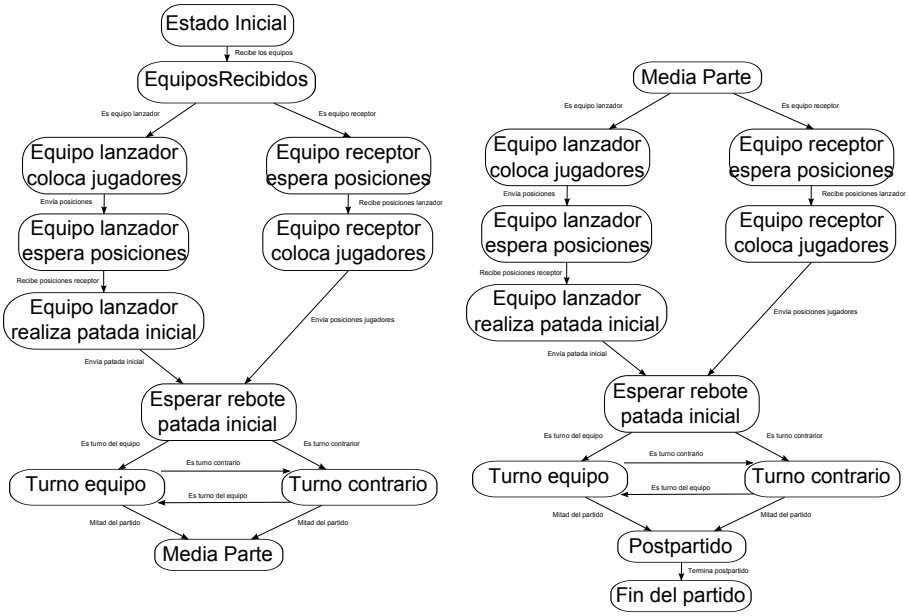


Figura 4.2: Maquina de estados de la parte cliente

puede hacer un jugador: mover, placar, pasar, girar, ponerse en pie o recoger el balón del suelo.

A continuación el usuario tiene dos opciones, seleccionar otro jugador o elegir ejecutar una acción del jugador que tiene seleccionado. Si elige ejecutar la acción se enviará la petición al servidor para que comience a resolver la acción, a continuación tendrá lugar una secuencia de acciones por parte del usuario que dependerá de la acción, por ejemplo si se ha elegido mover al jugador se deberán indicar al servidor las casillas por las que se desea que se mueva el jugador. Si se ha elegido pasar el balón entonces deberá indicarse el jugador que realizará la recepción, etc.

Una acción puede terminar debido a dos causas, la más común es que la acción se termine con éxito (no se ha fallado ninguna tirada y se ha conseguido completar con éxito la acción) y entonces el servidor comunica que la acción ha terminado. La otra opción es que el jugador falle (por ejemplo al placar un jugador o al moverse), y el servidor comunique se que produce un cambio de turno, en este caso el turno del equipo finalizará. Cuando se ha recibido por parte del servidor que la acción ha terminado, se tiene la posibilidad de elegir un nuevo jugador (o el mismo jugador que acaba de

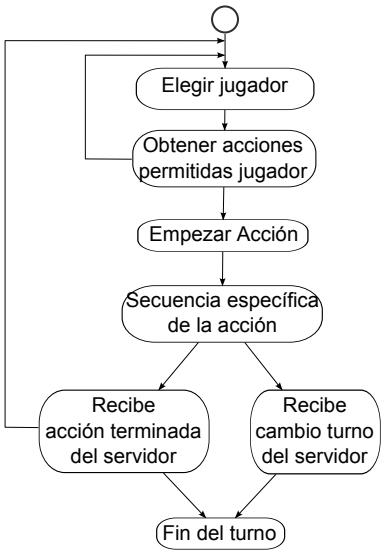


Figura 4.3: Secuecia de un turno

realizar la acción) para realizar otra acción, o finalizar el turno.

4.1.2 Formalización de las acciones

Dentro de este apartado se explicará detalladamente cuales son los pasos que se llevan a cabo en el servidor cuando se solicita que un jugador de un equipo realice una acción indicada.

Colocar jugadores

Colocar los jugadores no es una acción de un jugador sino del entrenador, que solicita al servidor que valide sus posiciones. Para que las posiciones de los jugadores se consideren correctas por parte del servidor, éstos deben colocarse en la mitad del campo destinado a ellos, esto quiere decir que si los jugadores pertenecen al equipo lanzador deberán colocarse en su respectiva mitad del campo (e igualmente para los jugadores del equipo receptor), pudiéndose colocar hasta un máximo de 11 jugadores, los cuales deben cumplir que:

- Dos jugadores deben estar situados a lo largo de la primera zona ancha.
- Otros dos jugadores deben estar situados a lo largo de la segunda zona ancha.

- Al menos tres jugadores deben estar colocados en la línea de defensa del campo.

De las reglas anteriores puede inferirse que como mínimo un equipo necesita siete jugadores para jugar.

Patada inicial

Para realizar la patada inicial se indica al servidor el jugador que va a realizar la patada y la casilla destino. Si el jugador es del equipo lanzador entonces la casilla debe pertenecer al campo del equipo receptor, en caso contrario la casilla debe pertenecer al campo del equipo lanzador. A continuación se calcula el rebote del balón tres veces, a la hora de calcular el balón se tiene en cuenta que durante el rebote no se atraviesa en ninguna casilla del equipo cuyo jugador realiza la patada.

Mover

Cuando el servidor recibe la petición por parte del usuario para mover un jugador, recibe juntamente con el nombre del jugador y el equipo al que pertenece, la secuencia de casillas por las que se quiere que se mueva el jugador. El servidor se encarga de evaluar el éxito del movimiento del jugador para cada una de las casillas.

El proceso que realiza el servidor para evaluar si el jugador se mueve a una casilla es el descrito a continuación. En primer lugar el servidor comprueba que se trata de un número de casilla válido (entre 0 y 390), a continuación se comprueba que el jugador no haya placado, ni realizado un pase, que no haya terminado su acción y que se encuentre de pie; seguidamente se comprueba que la casilla destino está vacía. El siguiente paso es medir la distancia entre la casilla actual del jugador y la casilla destino, si la distancia es mayor que dos, entonces no se realiza el movimiento, si la distancia es dos y el jugador no posee la habilidad saltar entonces no se realiza el movimiento, si la distancia es cero no se realiza el movimiento, en cualquier caso se sigue con las comprobaciones. Si la distancia era de 2 se elimina la habilidad saltar y se restan 2 puntos de movimiento al jugador, si la distancia era de 1 entonces solamente se resta 1 punto de movimiento.

En el caso que el jugador no posea suficientes puntos de movimiento, pero aún le queden puntos de movimiento extra, entonces se resuelve una tirada utilizando como atributo la agilidad del jugador, si la supera entonces el jugador puede moverse, si la falla entonces el jugador posee la habilidad profesional (y el entrador da permiso para usarla) o el equipo posee fichas de segunda oportunidad y todavía no se ha usado

ninguna durante este turno (y el entrenador da permiso para usarla) se repite la tirada, en caso contrario el jugador cae derribado en la casilla destino.

En caso de superar la tirada por el uso de movimiento extra, o si no se necesitó usar puntos de movimiento extra, se examinan cuantos jugadores del equipo adversarios afectan la casilla actual del jugador, si el resultado es 0 entonces el jugador se mueve a la casilla destino.

Si hay adversarios alrededor de la posición actual del jugador, se suma 1 al modificador de la tirada si ningún adversario posee la habilidad placaje defensivo, o si el jugador posee la habilidad dos cabezas; si el jugador no posee la habilidad escabullirse entonces se resta 1 al modificador de la tirada por cada adversario a distancia 1 del jugador, además se le resta un 1 adiciona si el adversario posee la habilidad cola prensil.

Si algún jugador adversario que está a distancia 1 del jugador posee la habilidad Tentáculos se compara la suma de la fuerza del adversario y un valor aleatorio del 1 al 6 con la suma de la fuerza del jugador mas un número aleatorio entre 1 y 6; si la primera suma es mayor a la segunda entonces el jugador no podrá efectuar el movimiento.

A continuación se resuelve la tirada usando como atributo la agilidad (o la fuerza si el jugador tiene el atributo abrirse paso y el valor de su fuerza es mayor que su valor de agilidad) aplicándole los modificadores, si la supera entonces el jugador de mueve a la casilla; en caso de fallar la tirada el jugador puede repetirla en los siguientes casos: posee la habilidad pies firmes, posee la habilidad profesional y el entrenador autoriza su uso, o si no se ha usado ninguna ficha de segunda oportunidad, aún le quedan al equipo y el entrenador autoriza su uso. En caso de no cumplirse ninguna de estas condiciones el jugador caerá derribado en la casilla actual en la que se encuentra.

El servidor evalúa el éxito del movimiento hasta que se terminan las casillas o hasta que el movimiento falla. En ese momento comunica el movimiento con éxito de las casillas por las que ha superado el movimiento, y luego (si se ha fallado el movimiento) se comunica el derribo del jugador.

Placar

Cuando el servidor recibe la petición para placar a un jugador, en la petición se indica el nombre del jugador que realizará el placaje y el de su equipo junto con el nombre del jugador que se va a placar.

A continuación el jugador evalúa si el jugador atacante ha movido con anterioridad, en caso afirmativo para continuar con el placaje el equipo no debe haber realizado

ningún movimiento de penetración; también comprueba que el jugador atacante y el jugador defensor están a una distancia de 1 una casilla de la otra y que ambos jugadores están en pie.

Por cada jugador del equipo del jugador atacante situado a una distancia de uno de su casilla, y que no posee ningún adversario en su zona de defensa se suma uno al modificador de la tirada, si además el jugador posee la habilidad defender se suma un uno adicional.

Por cada jugador del equipo del jugador defensor situado a distancia de uno de su casilla y que no posee ningún adversario en su zona de defensa, se resta uno al modificador de la tirada, además se resta un uno adicional si el jugador posee la habilidad cola prensil.

Si además el jugador atacante posee la habilidad escabullirse y el jugador defensor no posee la habilidad placaje defensivo, se suma uno al modificador de la tirada.

A continuación se evalúa la fuerza del los jugadores enfrentados, si la fuerza de un jugador es superior al doble de la de su contrincante entonces se calculan tres resultados posibles, si sus fuerzas son iguales entonces se calcula solo un resultado posible, y si son distintas se calculan dos resultados. Los resultados pueden ser: atacante derribado, ambos derribados, defensor derribado, defensor empujado, defensor empujado y derribado y ninguno derribado. Si hay varios resultados entre los que elegir, el entrenador del jugador con más fuerza elegirá el resultado.

Se considera que se ha fallado el placaje si el resultado obtenido es el de atacante derribado o ambos derribados, en caso contrario se considera que el placaje ha tenido éxito.

Si se falla el placaje los resultados pueden volver a calcularse si el jugador posee la habilidad profesional y el entrenador permite su uso o si se puede usar una ficha de segunda oportunidad y el entrenador permite su uso.

Cuando un jugador es derribado, si posee el balón, se calcula el rebote de una casilla para el balón; en todos los casos se realiza una tirada contra armadura (tal y como se detalla en el reglamento) y se procede a realizar un cambio de turno.

Cuando el jugador defensor es empujado, se pregunta al entrenador del equipo atacante que elija la casilla destino, las posibles casillas destino son aquellas casillas libres que sitúan al jugador defensor entre el jugador atacante y ella misma.

Pasar

Cuando el servidor recibe la petición para realizar un pase, el usuario le indica el nombre del jugador que realizará el pase y el nombre del equipo al que pertenece. A continuación el servidor comprueba que el jugador se encuentra en pie, que posee el balón y que no ha terminado su acción ni ha placado durante el turno.

A continuación el servidor pregunta al entrenador con qué jugador quiere recibir el pase, mostrándole a continuación la dificultad calculada para el pase. Si la distancia entre las casillas del jugador lanzador y el receptor es igual o menor que 3 entonces es un pase rápido, si está entre 4 y 6 es de tipo pase corto, si está entre 7 y 9 se trata de un pase largo, y si es mayor que 10 es pase es de categoría bomba larga. Se solicita que el entrenador confirme el pase, de lo contrario se le vuelve a solicitar que elija el jugador que recibirá el pase hasta que confirme un pase.

Una vez sabemos el jugador receptor, el servidor resuelve el lanzamiento utilizando el valor de la agilidad del jugador. Al modificador de la tirada se le suma uno si el pase es de categoría pase corto, se le resta uno si es de categoría pase largo y se le resta un dos si el pase es de categoría bomba larga; si el jugador posee la habilidad precisión, el modificador se incrementa en uno. Se resuelve la tirada, si la supera se continúa con las comprobaciones, si la falla puede repetirse en uno de los siguientes casos: el jugador posee la habilidad pasar, el jugador posee la habilidad profesional y el entrenador permite su uso o si se puede usar una ficha de segunda oportunidad y el entrenador permite su uso. En caso de fallar el lanzamiento, se calcula el rebote del balón por tres casillas desde la casilla del jugador lanzador y se produce un cambio de turno.

A continuación se calculan las casillas que atravesará el balón como trayectoria del pase, si hay algún jugador adversario en alguna de dichas casillas entonces podrá realizarse una intercepción; en este caso el servidor solicita al entrenador adversario que elija el nombre del jugador con el que interceptará el pase; seguidamente se resuelve la intercepción. Al modificador de la tirada se le resta uno por cada jugador del equipo lanzador en la zona de defensa del jugador interceptor; si el jugador interceptor posee la habilidad piernas largas entonces se le suma uno al modificador de la tirada. Si se supera la tirada, en el caso que el jugador lanzador posea la habilidad pase seguro se cancela la intercepción y no se continúa con el pase, en caso contrario el jugador interceptor pasa a poseer el balón y se produce un cambio de turno. Si se falla la tirada se continúa con la evaluación de la recepción del pase.

Si el jugador receptor posee la habilidad manos grandes, entonces resuelve la tirada, si la supera entonces el jugador recoge el balón. Si se falla la tirada entonces se repetirá la tirada si se cumple uno de los siguientes casos: el jugador receptor posee la habilidad manos grandes, o el jugador receptor posee la habilidad profesional y el entrenador permite su uso, o se puede usar una ficha de segunda oportunidad y el entrenador permite su uso.

Si el jugador receptor no posee la habilidad manos grandes, y si el jugador receptor tampoco posee la habilidad de nervios de acero, se resta uno al modificador de la tirada por cada jugador adversario a distancia uno del jugador receptor; se suma uno al modificador de la jugada si el jugador receptor tiene la habilidad brazos adicionales. A continuación se resuelve la tirada, si la supera entonces el jugador recibe el balón, si la falla se repetirá la tirada si se cumple alguna de las siguientes condiciones: el jugador receptor posee la habilidad manos seguras o atrapar, el jugador receptor posee la habilidad profesional y el entrenador permite su uso, o se puede usar una ficha de segunda oportunidad y el entrenador permite su uso.

Si finalmente se ha fallado la tirada, entonces el balón rebotará tres casillas desde la posición del jugador receptor y se producirá un cambio de turno. Si en cambio la tirada se supera, entonces el jugador receptor ha completado con éxito el pase.

Recoger balón

Cuando el servidor recibe la petición para recoger el balón, el usuario le indica el jugador y el equipo del jugador con el cual realizar la acción. El servidor comprueba que el jugador se encuentra de pie y que no ha terminado su acción por este turno.

Si el jugador posee la habilidad manos grandes, entonces resuelve la tirada, si la supera entonces el jugador recoge el balón. Si se falla la tirada entonces se repetirá en uno de los siguientes supuestos: el jugador posee la habilidad manos grandes, el jugador posee la habilidad profesional y el entrenador permite su uso, se puede usar una ficha de segunda oportunidad y el entrenador permite su uso. Si finalmente se ha fallado la tirada, entonces el balón rebotará una casilla.

Si el jugador no posee la habilidad manos grandes, si el jugador no posee la habilidad de nervios de acero, se resta uno al modificador de la tirada por cada jugador adversario a distancia uno del jugador; se suma uno al modificador de la jugada si el jugador tiene la habilidad brazos adicionales. A continuación se resuelve la tirada, si la supera entonces el jugador recoge el balón, si la falla se repetirá la tirada si se cumple alguna

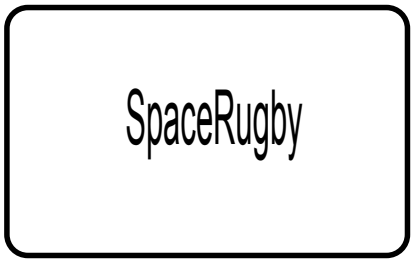


Figura 4.4: Pantalla de introducción

de las siguientes condiciones: el jugador posee la habilidad manos seguras o atrapar, el jugador posee la habilidad profesional y el entrenador permite su uso, se puede usar una ficha de segunda oportunidad y el entrenador permite su uso. Si finalmente se ha fallado la tirada, entonces el balón rebotará una casilla.

4.2 Interfaces de usuario

En este apartado se realizará la descripción del formato de las pantallas individuales. Esta descripción se realizará desde un punto de vista estático, es decir, elementos que aparecerán siempre en la pantalla de la misma forma.

A continuación se describirán cada una de las diferentes pantallas que se podrán encontrar en la aplicación:

4.2.1 Pantalla de introducción

En la figura 4.4 podemos apreciar la pantalla de introducción. La pantalla se muestra de 5 a 10 segundos y luego se pasa a V2.

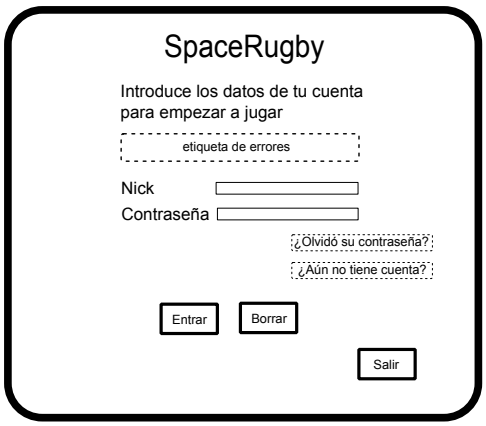


Figura 4.5: Pantalla de bienvenida

4.2.2 Pantalla de bienvenida

En la figura 4.5 podemos apreciar la pantalla de bienvenida.

El textbox de Nick soporta un máximo de 10 caracteres, el de contraseña de 30.

El Nick debe estar formado por letras, números y guión bajo; debe tener una longitud máxima de 10 caracteres y empezar siempre por una letra; no se hacen distinciones entre mayúsculas y minúsculas.

La contraseña puede estar formada por letras, números, guión bajo, guión alto, punto, coma, signo de cierre de exclamación, signo de cierre de interrogación, paréntesis, corchetes, comillas simples, comillas dobles, espacio, carácter dólar, carácter ampersand, caracteres de menor y mayor que, igual, dos puntos y punto y coma. La contraseña debe tener una longitud máxima de 30 caracteres.

El botón Entrar realiza las gestiones de autenticación, si es válida se irá a V7, en caso contrario se mostrará en la etiqueta de errores (que al cargarlo permanece sin texto) el siguiente mensaje: "Error al validar: Nick y/o contraseña incorrectos".

El botón Borrar elimina los textos que pudieran haber en los textbox de Nick y contraseña, dejándolos vacíos.

La etiqueta "¿Olvidó su contraseña?" es una etiqueta que al hacer click nos lleva a V3, mientras que al hacer click sobre la etiqueta "¿Aún no tiene cuenta?" vamos a V4. El botón salir va a V29.

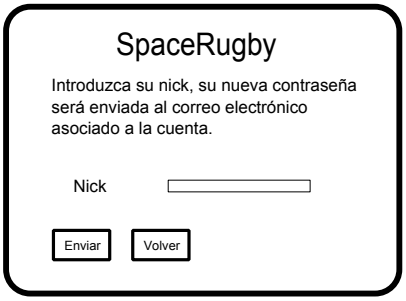


Figura 4.6: Pantalla de recuperación de contraseña

Figura 4.7: Pantalla de nueva cuenta

4.2.3 V3: Pantalla de recuperación de contraseña

En la figura 4.6 podemos apreciar la Pantalla de recuperación de contraseña.

El textbox de Nick y el Nick introducido deben cumplir lo especificado en V2.

El botón Enviar procesa la petición de recuperación de contraseña y vuelve a V2. El botón Volver vuelve a V2 sin procesar la petición de recuperación de contraseña.

4.2.4 V4: Pantalla de nueva cuenta

En la figura 4.7 podemos apreciar la pantalla de nueva cuenta.

El textbox de Nick y su contenido deben cumplir lo especificado en V2 al respecto.

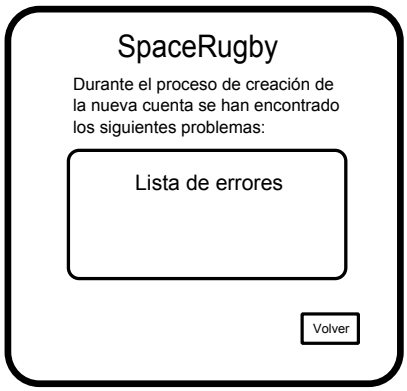


Figura 4.8: Pantalla de problema al crear cuenta

Los textbox de contraseña y repetir contraseña y su contenido deben cumplir las restricciones del textbox contraseña y el formato de la contraseña definidos en V2.

El nombre solo puede estar formado por caracteres alfabéticos (acentos incluidos) y el carácter espacio; tiene una longitud máxima de 25 caracteres.

El textbox de nombre permite hasta un máximo de 25 caracteres.

Los apellidos pueden estar formados por caracteres alfabéticos (acentos incluidos), el carácter espacio y el carácter guión alto; tiene una longitud máxima de 50 caracteres.

El textbox de apellidos permite hasta un máximo de 50 caracteres.

El email tiene una longitud máxima de 50 caracteres y debe guardar la estructura: [texto]@[texto].[texto], donde [texto] son 1 o más caracteres.

El textbox de email permite hasta un máximo de 50 caracteres.

El botón Enviar procesa la petición de creación de nueva cuenta, si tiene éxito va a V6 sino va a V5.

El botón Borrar elimina los textos que pudiera haber en los textbox, dejándolos vacíos.

El botón Volver nos lleva a V2 sin procesar la petición de creación de nueva cuenta.

4.2.5 V5: Pantalla de problema al crear cuenta

En la figura 4.8 podemos apreciar la pantalla de problema al crear cuenta.

La lista contiene los errores que el servidor ha detectado.

Figura 4.9: Pantalla de cuenta creada

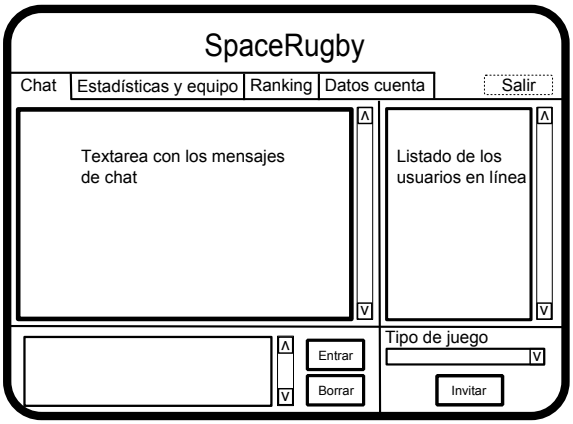


Figura 4.10: Pantalla de chat

El botón Volver nos lleva de nuevo a V4.

4.2.6 V6: Pantalla de cuenta creada

En la figura 4.9 podemos apreciar la pantalla de cuenta creada.

El botón Volver nos lleva a V2.

4.2.7 V7: Pantalla de chat

En la figura 4.10 podemos apreciar la pantalla de chat.

El menú ofrece cuatro opciones: el chat es la opción por defecto, estadísticas y equipo nos lleva a V9, ranking a V8 y datos cuenta a V30.

El botón Salir nos lleva a V29.

El textarea con los mensajes de chat muestra hasta un máximo de los 200 últimos mensajes. El formato de los mensajes es:

[fecha] [hora] Nick: [texto]

donde [fecha] es la fecha en formato dd/mm/aa, [hora] es la hora en formato hh:mm:ss, y [texto] pueden ser 1 o más caracteres.

El textarea de escritura de mensaje tiene una capacidad máxima de 180.

El botón Enviar (solo se activa si hay texto en el textarea de escritura de mensajes) envía el contenido del textarea de escritura de mensajes al servidor para que comunique el mensaje al resto de usuarios, y añade el mensaje al textarea con los mensajes de chat; por último borra el texto del textarea de escritura de mensajes.

El botón de Borrar (solo se activas si hay texto en el textarea de escritura de mensajes) borra el contenido de dicho textarea.

El listado de usuarios en línea muestra los usuarios por orden alfabético de Nick. La capacidad máxima del listado es de 50 usuarios.

El desplegable de tipo de juego tiene dos valores: oficial o amistoso.

El botón Invitar (que solo se activa si hay un jugador seleccionado en el listbox y un tipo de juego en el desplegable) realiza una invitación al jugador seleccionado en el listado para jugar un encuentro del tipo seleccionado y nos lleva a V10.

4.2.8 V8: Pantalla de ranking

En la figura 4.11 podemos apreciar la pantalla de ranking.

En las tablas: puesto, victorias, derrotas, empates y valoración son enteros positivos de 0 a (2^{32}-1).

Entrenador debe cumplir con lo expresado en V2 a propósito del Nick.

Raza debe cumplir lo expresado al respecto en V9.

Se mostrará en la lista de ranking, en la parte más alta los 10 mejores equipos, a continuación los 5 equipos justo por delante del equipo del jugador, a continuación el equipo del jugador y debajo de este los 5 equipos justo por detrás del equipo del jugador. En ningún caso se mostrarán quipos repetidos por lo cual si por ejemplo el equipo del jugador es el mejor, sólo se mostrarán los 10 primeros, si el equipo del jugador fuera el nº11, se mostrarán los 10 primeros, el equipo del jugador y los 5 siguientes, etc. Nótese

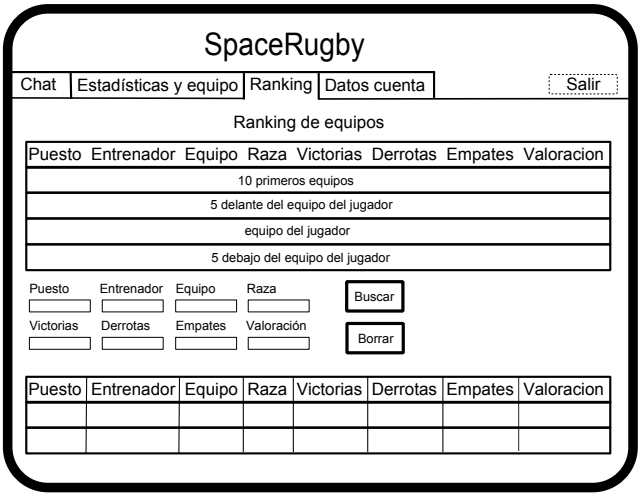


Figura 4.11: Pantalla de ranking

que si el equipo del jugador es el primero, no hay equipos por delante, y si es el último no hay equipos por detrás.

El botón Buscar busca de forma que intenta hallar el equipo que cumple con todas las condiciones especificadas, mostrando en la tabla inferior los resultados.

4.2.9 V9: Pantalla de estadísticas y equipo

En la figura 4.12 podemos apreciar la pantalla de estadísticas y equipo.

El nombre del equipo puede estar compuesto por letras, números y el carácter espacio; tiene como máximo una longitud de 30 caracteres.

La raza del equipo debe ser una de las siguientes: amazonas, caos, enanos del caos, elfos oscuros, enanos, goblin, halfling, altos elfos, humanos, hombres lagarto, norse, orcos, skaven, no muertos y elfos silvanos. La longitud máxima son 15 caracteres.

Victorias, derrotas, empates, valoración y dinero son valores enteros positivos que van desde 0 a (2^{32}-1).

Nº es un entero con valores de 1 a 2^{32}.

Nombre son letras y números (también permite caracteres espacio) con una longitud máxima de 30 caracteres.

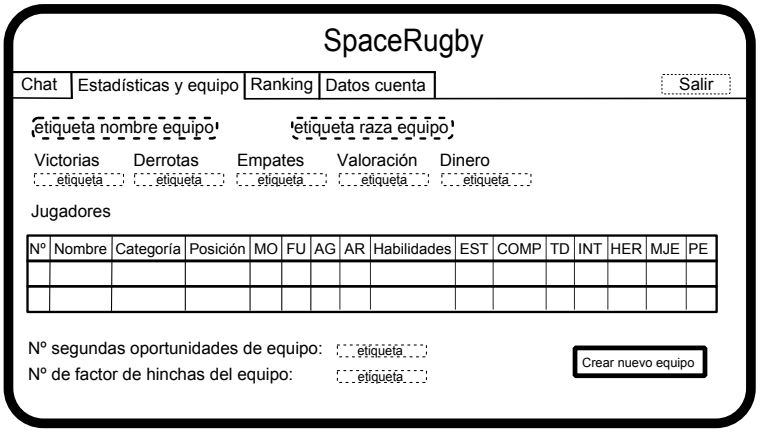


Figura 4.12: Pantalla de estadísticas y equipo

La categoría tiene una longitud máxima de 16 caracteres, y debe ser una de las siguientes: novato, experimentado, veterano, promesa, jugador estrella, superestrella, megaestrella y leyenda.

La posición tiene una longitud máxima de 25 caracteres, y dependiendo de la raza puede ser:

- Enanos del caos: hobgoblin, enano del caos, centauro.
- Caos: hombre bestioa, guerrero del caos.
- Elfos oscuros: línea, lanzador, blitzer, elfa bruja.
- Enanos: barbaslargas, corredor, blitzer, matatrolls.
- Goblins: goblin.
- Halflings: halfling.
- Altos elfos: línea, guerrero fénix, guerrero león, guerrero dragón.
- Humanos, amazonas, norse: línea, receptor, lanzador, blitzer.
- Orcos: línea, goblin, lanzador, orconegro, blitzer.

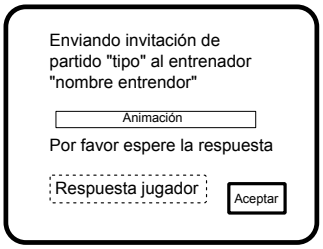


Figura 4.13: Pantalla de espera de respuesta a la invitación

- Skavens: línea, lanzador, corredor de alcantarillas, blitzer.
- No muertos: esqueleto, necrófago, zombie, tumulario, momia.
- Elfos silvanos: línea, receptor, lanzador, bailarín guerrero.
- Hombres lagarto: lagartos, skinks, sauros.

MO, FU, AG, AR son número enteros no negativos entre 1 y 16.

Habilidades: un jugador puede tener de 0 a n habilidades. La lista de todas las habilidades es la siguiente: Abrirse paso, Agallas, Atrapar, Brazos adicionales, Cola prensil, Defensa, Dos cabezas, Escabullirse, Esquivar, Mano grande, Manos seguras, Mantenerse firme, Nervios de acero, Pasar, Pase seguro, Piernas largas, Pies firmes, Placaje defensivo, Placar, Precisión, Profesional, Saltar, Tentáculos.

EST: son 5 caracteres de máximo; LP (lesión permanente); LPE (lesionado próximo encuentro), LPELP (lesionado próximo encuentro y lesión permanente) y S (sano).

COMP, TD, INT, HER, MJE y PE son valores positivos que van de 0 a (2^{32}-1), y lo mismo pasa con las segundas oportunidades y el factor de hinchas.

El botón Crear nuevo equipo va a V12.

El número máximo de jugadores de un equipo es 16.

4.2.10 V10: Pantalla de espera de respuesta a la invitación

En la figura 4.13 podemos apreciar la pantalla de espera de respuesta a la invitación.

La animación es una barra de progreso.

En la etiqueta aparecerá si el entrenador invitado ha aceptado o rechazado la invitación, o si expiró la invitación (no se recibió respuesta pasados dos minutos). Si se

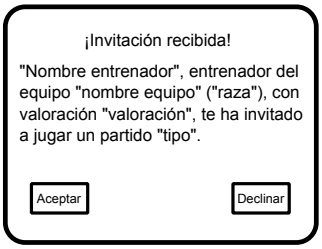


Figura 4.14: Pantalla de invitación a un partido

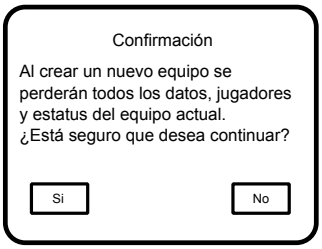


Figura 4.15: Pantalla de confirmación creación de nuevo equipo

acepta la invitación el resultado se muestra y al pulsar el botón aceptar se va a V16; en los otros casos se muestra el resultado y al pulsar el botón aceptar se vuelve a V7.

4.2.11 V11: Pantalla de invitación a un partido

En la figura 4.14 podemos apreciar la pantalla de invitación a un partido.

Si pulsa el botón Aceptar va a V16, si pulsa el botón Declinar vuelve a la pantalla en la que estaba.

El valor "nombre entrenador" debe cumplir lo descrito sobre Nick en V2. "nombre Equipo", "raza" y "valoración" están descritos en V9. El "tipo" de juego está descrito en V7.

4.2.12 V12: Pantalla de confirmación creación de nuevo equipo

En la figura 4.15 podemos apreciar la pantalla de confirmación creación de nuevo equipo.

El botón Si nos lleva a V13.

El botón No nos lleva a V9.

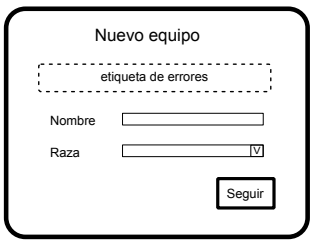


Figura 4.16: Pantalla de creación de nuevo equipo

4.2.13 V13: Pantalla de creación de nuevo equipo

En la figura 4.16 podemos apreciar la pantalla de creación de nuevo equipo.

Las restricciones respecto al nombre del equipo así como las distintas razas disponibles están en V9.

El botón Seguir lleva a V15.

4.2.14 V15: Pantalla de compra de jugadores

En la figura 4.17 podemos apreciar la pantalla de compra de jugadores.

En el desplegable posición se cargarán los valores correspondientes de la raza del equipo ocmo ya se explicó en V9. Sobre restricciones en el textbox del nombre del jugador ir a lo descrito en V9 al respecto.

Al seleccionar la posición se cargan los valores iniciales de MO, FU, AG, AR, Habilidades, Nº Máximo por equipo y precio. Además en la lista de habilidades se marcan de manera automática los checkbox de aquellas habilidades que puede poseer el jugador y se mantienen sin marcar las que no puede poseer. De ninguna manera el usuario puede modificar estos checkbox.

La información sobre las características de MO, FU, AG, AR, Habilidades y las habilidades que el jugador puede o no poseer según su posición y raza están definidos en el reglamento (2).

El botón Comprar jugador realiza la petición de compra del jugador, en el caso que el nombre del jugador no cumpla las especificaciones necesarias, no se disponga del suficiente dinero o ya se haya alcanzado el máximo de jugadores permitidos en dicha posición por equipo, no se completa la operación de compra; en caso de que la petición

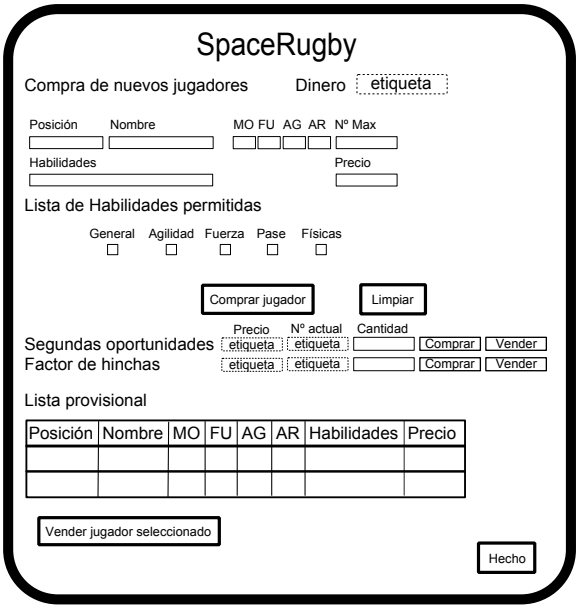


Figura 4.17: Pantalla de compra de jugadores

sea procesada con éxito se añade el jugador junto con sus características a la lista provisional de jugadores, y se reinicia el desplegable, el textbox, las etiquetas de MO, FU, AG, AR, Habilidad, Nº Máximo por equipo y precio; y los chechBox se mantienen sin marcar.

Tanto el valor de las etiquetas de las segundas oportunidades como del factor de hinchas (y de sus precios), así como el de los textbox para indicar cuantos se compran pueden tener valores numéricos entre 0 y (2^{32}-1).

La etiqueta que indica el dinero que tiene el equipo toma valores enteros entre 0 y (2^{32}-1).

El botón Limpiar lo que hace es reiniciar el desplegable, el textbox, las etiquetas de MO, FU, AG, AR, Habilidad, Nº Máximo por equipo, precio y los chechBox se mantienen desmarcados.

El botón de Comprar/Vender una segunda oportunidad o factor de hinchas comprará/venderá tantas como lo indicado en el textbox; en caso de compra, si no se dispone del suficiente dinero no se efectuará la compra; en caso de venta si se especifica

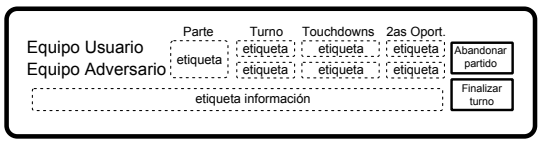


Figura 4.18: Pantalla de partido

Figura 4.19: Pantalla de resultado del partido

más cantidad de la que dispone, no se efectuará la venta.

El botón Hecho realiza la petición de que se cree el equipo y nos envía a V9.

4.2.15 V16: Pantalla de partido

En la figura 4.18 podemos apreciar la pantalla de partido.

El valor de la parte será 1 o 2.

El valor de turno irá de 0 a 8.

El valor de Touchdowns y segundas oportunidades puede ir de 0 a (2^{32}-1).

El botón Fin de turno da por terminado el turno del equipo y el turno pasa a ser del otro equipo. El fin de turno puede implicar o no un cambio de parte. Se produce un cambio de parte cuando ambos equipos han jugado 8 turnos en la 1ª parte. El partido se finaliza cuando ambos equipos han jugado 8 turnos en la 2ª parte, en ese momento se va a V17.

Si se pulsa el botón de Abandonar partido o escape se va a V27.

4.2.16 V17: Pantalla de resultado del partido

En la figura 4.19 podemos apreciar la pantalla de resultado del partido.

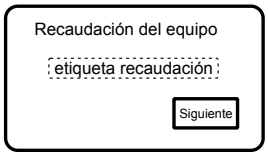


Figura 4.20: Pantalla de recaudación del partido

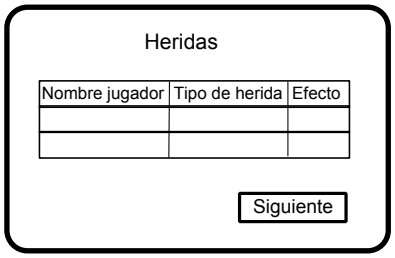


Figura 4.21: Pantalla de heridas

Las etiquetas con los nombres de los equipos deben cumplir lo descrito en V9 sobre nombres de equipo.

Las etiquetas de TouchDowns tendrán un valor entero positivo de 0 a (2^{32}-1).

La etiqueta resultado mostrará el texto "¡Has ganado!" si se es el equipo ganador, o "¡Perdiste!" si se es el equipo perdedor.

El botón Siguiente lleva a V18.

4.2.17 V18: Pantalla de recaudación del partido

En la figura 4.20 podemos apreciar la pantalla de recaudación del partido.

La etiqueta recaudación tiene un valor entero positivo de 0 a (2^{32}-1).

El botón Siguiente nos lleva a V19.

4.2.18 V19: Pantalla de heridas

En la figura 4.21 podemos apreciar la pantalla de heridas.

El nombre del jugador debe cumplir lo especificado en V9 al respecto.

Los tipos de herida están especificados en el reglamento (2).

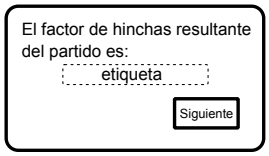


Figura 4.22: Pantalla de factor de hinchas resultante

El efecto puede tomar los siguientes valores: pierde próximo partido, no consigue puntos de experiencia o muerto.

El botón Siguiente lleva a V22.

4.2.19 V22: Pantalla de factor de hinchas resultante

En la figura 4.22 podemos apreciar la pantalla de factor de hinchas resultante.

La etiqueta contiene un valor entero positivo de 0 a (2^{32}-1).

El botón Siguiente nos lleva a V23.

4.2.20 V23: Pantalla de compra de nuevos jugadores

En la figura 4.23 podemos apreciar la pantalla de compra de nuevos jugadores.

Igual que V15 excepto que cuando se despide un jugador se va a V28 y si se contesta Si se elimina sin recibir ningún dinero, y si se contesta No entonces no se procede al despido.

El botón Hecho nos lleva a V24.

4.2.21 V24: Pantalla de valoración del equipo

En la figura 4.24 podemos apreciar la pantalla de valoración del equipo.

La etiqueta tendrá un valor entero positivo de 0 a (2^{32}-1).

El botón Fin nos lleva a V7.

4.2.22 V27: Pantalla confirmación de abandono de partida

En la figura 4.25 podemos apreciar la pantalla confirmación de abandono de partida.

Devuelve a la pantalla que la llamó si se ha pulsado Sí o No.

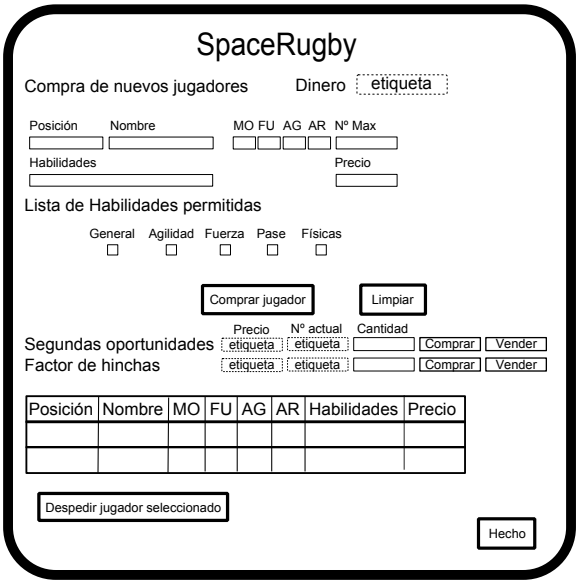


Figura 4.23: Pantalla de compra de nuevos jugadores

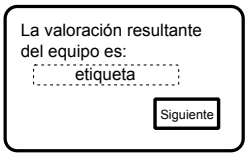


Figura 4.24: Pantalla de valoración del equipo

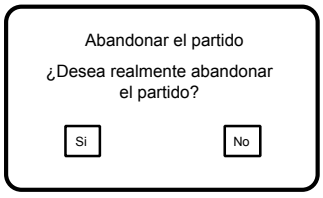


Figura 4.25: Pantalla confirmación de abandono de partida

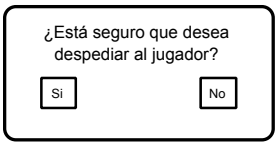


Figura 4.26: Pantalla confirmación despido de jugador

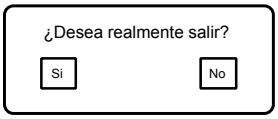


Figura 4.27: Pantalla confirmación salir

4.2.23 V28: Pantalla confirmación despido de jugador

En la figura 4.26 podemos apreciar la pantalla confirmación despido de jugador.

Devuelve a la pantalla que la llamó si se ha pulsado Sí o No.

4.2.24 V29: Pantalla confirmación salir

En la figura 4.27 podemos apreciar la pantalla confirmación salir.

Si se pulsa Si se sale de la aplicación, si se pulsa No se vuelve a la ventana en la que estaba.

4.2.25 V30: Pantalla de datos de la cuenta

En la figura 4.28 podemos apreciar la pantalla de datos de la cuenta.

El Nick, nombre, apellidos y email deben respetar lo especificado al respecto en V4.

4.2.26 V31: Pantalla de editar datos de la cuenta

En la figura 4.29 podemos apreciar la pantalla de editar datos de la cuenta.

Los campos contraseña, repetir contraseña, nombre, apellidos y email deben cumplir lo especificado en V4.

El botón Cambiar procesa la petición de cambio de datos.

El botón Borrar borra el posible contenido de los textbox dejándolos vacíos.

El botón Volver va a V30 sin haber procesado la petición de cambio de datos.

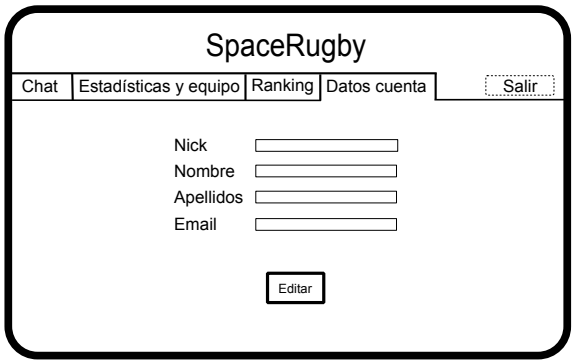


Figura 4.28: Pantalla de datos de la cuenta

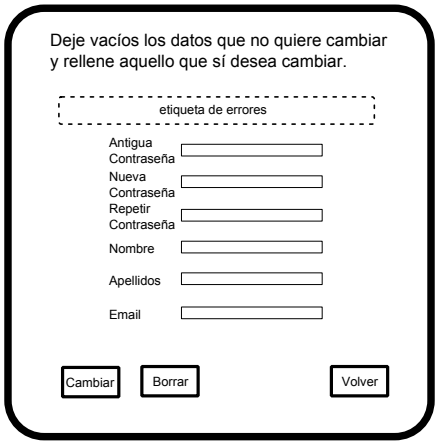


Figura 4.29: Pantalla de editar datos de la cuenta

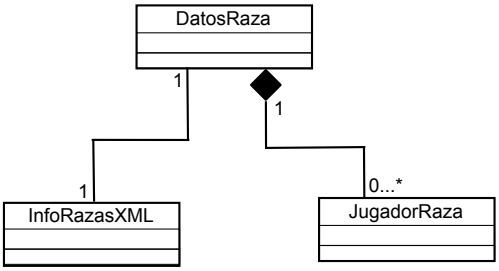


Figura 4.30: Clases de manejo de los datos de razas

4.3 Persistencia de datos

El primer aspecto en el cual se va a profundizar es la manera en que se ha realizado la persistencia de los datos. Estos datos que debe almacenar la aplicación son de dos tipos: datos de usuarios y datos de relativos a razas.

Los datos de usuarios no solamente comprenden los datos personales de los usuarios, comprende toda la información que posee la cuenta de un usuario. Esto incluye la información de su equipo y sus jugadores.

Los datos relativos a las razas son los datos de las distintas razas de que se dispone cuando se va a crear un equipo y las distintas posiciones disponibles para las distintas razas.

En la figura 4.30 puede verse las clases involucradas en la inclusión de los datos relativos a las razas de los equipos. Dicha información se encuentra almacenada en un fichero XML y que es cargado por el servidor cuando éste es lanzado.

4.3.1 Persistencia de los datos de las razas

Datos de la raza

De una raza solo se precisa almacenar tres cosas: su Nombre, el precio de las segundas oportunidades y las posiciones disponibles para los jugadores.

```
1 public string Nombre;
  public List<JugadorRaza> Posiciones;
3 public int PrecioSegundasOportunidades;
```

Jugador de una raza

En esta clase se almacena la información relativa a las posiciones que pueden ocupar los jugadores. La información que contiene la posición de un jugador es el nombre de la posición, la cantidad máxima de jugadores que pueden ocupar dicha posición en un equipo, el precio en monedas de oro que cuesta un jugador que juega en esta posición; los valores iniciales de movimiento, fuerza, agilidad y armadura de los jugadores en esta posición; también contiene información sobre qué habilidades puede y no puede aprender, y una lista de las habilidades que inicialmente poseen los jugadores en juegan en esa posición.

```
1  public int CantidadMaxima;
   public string Posicion;
3  public int Precio;
   public int Movimiento;
5  public int Fuerza;
   public int Agilidad;
7  public int Armadura;
   public bool HabilidadesGeneralesPermitidas;
9  public bool HabilidadesAgilidadPermitidas;
   public bool HabilidadesFuerzaPermitidas;
11 public bool HabilidadesPasePermitidas;
   public bool HabilidadesFisicasPermitidas;
13 public List<string> Habilidades;
   public string ConvertirEnCadena()
```

El método ConvertirEnCadena convierte en una cadena toda la información de la posición del equipo de forma que resulta fácil transmitirla a la parte cliente.

Información de razas XML

Como se ha dicho, la información de las razas se encuentra en un archivo XML. El formato del fichero está definido en el siguiente DTD:

```
   <?xml version="1.0" encoding="UTF-8"?>
2  <!ELEMENT RAZAS (Raza)* >
     <!ELEMENT Raza (Nombre, Jugadores, SegundasOportunidades)>
4      <!ELEMENT Nombre (#PCDATA)>
       <!ELEMENT Jugadores (Jugador)*>
6          <!ELEMENT Jugador (Cantidad, Posicion, Precio, Movimiento, Fuerza,
               Agilidad, Armadura, Habilidades)>
                 <!ELEMENT Cantidad (#PCDATA)>
8                <!ELEMENT Posicion (#PCDATA)>
                 <!ELEMENT Precio (#PCDATA)>
10               <!ELEMENT Movimiento (#PCDATA)>
                 <!ELEMENT Fuerza (#PCDATA)>
```

```
12                  <!ELEMENT Agilidad (#PCDATA)>
                    <!ELEMENT Armadura (#PCDATA)>
14                  <!ELEMENT Habilidades (Habilidad)*>
                            <!ELEMENT Habilidad (#PCDATA)>
16      <!ELEMENT SegundasOportunidades (#PCDATA)>
```

Los elementos del DTD son los mismos que los que se encuentran en la clase *JugadorRaza* por lo que no requieren de mucha explicación. Lo que hace el método *InfoRazasXML* es leer el fichero con la información de las distintas razas, devolviendo una lista de instancias de *DatosRaza* con la información de la raza y sus distintas posiciones. *InfoRazasXML* se encarga de comprobar que el formato del fichero es correcto y que los tipos de los datos son correctos, por ello posee los métodos *ComprobarInt* y *ComprobarString*.

Una vez se ha parseado se puede comprobar mediante el atributo correcto que el parseo ha sido satisfactorio.

```
    //Atributos
2   public bool Correcto;
    //Métodos
4   public List<DatosRaza> parsear()
    private int ComprobarInt(string valor)
6   private string ComprobarString(string valor)
```

4.3.2 Persistencia de los datos del usuario

Los datos de los usuarios se encuentran almacenados en una base de datos *Access*. El esquema de la base de datos creada es el que se muestra en la figura 4.31. En total son cuatro tablas: *Usuarios* contiene los datos personales de los usuarios, *Equipos* contiene los datos de los equipos existentes en el sistema, JugadoresEquipos contiene los datos de los jugadores de los equipos del sistema y *HabilidadesJugadorEquipo* se utiliza para guardar la relación de habilidades que posee un jugador.

Para manejar la inserción, actualización y borrado de la información almacenada en la base de datos, se han creado la clase *Usuario*, *Equipo* y *JugadorEquipo*, cuyo diagrama podemos ver en la figura 4.32. En ellas apreciamos que todas poseen un método Cargar, Guardar, Crear, Eliminar y Dispose; a excepción de Dispose (que tiene relación con la instancia del objeto) los otros métodos son los encargados de obtener, actualizar, crear y eliminar los datos de la base de datos; el resultado booleano devuelto

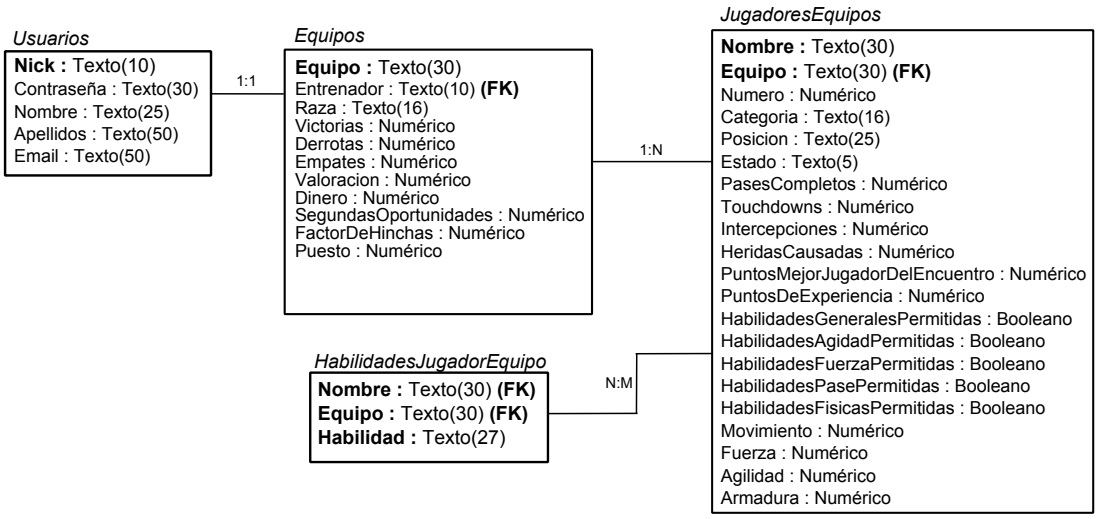


Figura 4.31: Esquema de la base de datos

por los métodos es verdadero en caso de llevar a cabo la operación con éxito y falso en caso contrario.

Debemos tener en cuenta que la operación Eliminar realiza un borrado en cascada, esto quiere decir que por ejemplo al eliminar un equipo se eliminan también los jugadores de la tabla JugadoresEquipos que pertenecen a dicho equipo; de igual manera al borrar un jugador se produce un borrado en cascada de aquellas habilidades de la tabla HabilidadesJugadorEquipo que estuvieran relacionadas con dicho jugador.

Las clases tienen atributos públicos que se corresponden con los campos de base

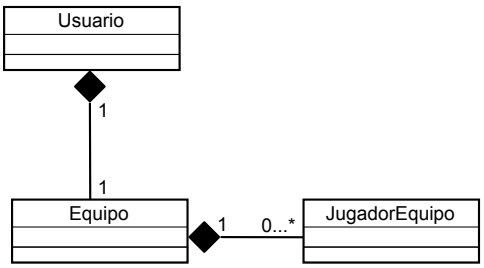


Figura 4.32: Clases de manejo de los datos de usuario

de datos a los cuales representan, y que por lo tanto ya han sido explicados. Resulta interesante fijar la atención en los atributos privados de las clases, las cuales siempre tienen un atributo privado Conexión (utilizado para localizar y acceder a la base de datos) y _contenidoValido que se utiliza para saber si los datos de la clase coinciden con los que hay en la base de datos o han cambiado.

En el atributo Conexión se indica que la base de datos se llama SpaceRugby.mdb y que necesariamente se encuentra dentro de una carpeta denominada File en el mismo directorio de la aplicación servidor.

Ya que el atributo Contraseña es de ambiente privado, la clase Usuario posee dos métodos para manejar la comprobación de contraseñas y su cambio. ContraseñaCorrecta devuelve verdadero si la contraseña pasada por parámetro coincide con la contraseña almacenada por la clase, y falso en caso contrario; ello evita que se pueda acceder directamente al valor de la contraseña, ya que el resto de clases del servidor no tienen acceso a él. Cuando se llama al método CambiarContraseña, se comprueba si la contraseña actual del usuario coincide con el valor especificado en contraseñaVieja, y en caso afirmativo actualiza el valor de la contraseña del usuario al valor especificado en el parámetro contraseñaNueva y devuelve verdadero; en cualquier otro caso devuelve falso.

El método ConvertirEnCadena de la clase Equipo se utiliza para guardar en una cadena de caracteres toda la información del equipo (incluida la información de los jugadores y las habilidades de los jugadores), de esta forma resulta sencillo transmitirla a la parte cliente, quien interpretará la cadena y obtendrá la información del equipo.

Usuario

```
//Atributos
public string Nick;
private string \_nickBD;
private string Contraseña;
public string Nombre;
public string Apellidos;
public string Email;
public string NombreEquipoUsuario;
public Equipo Equipo;
private bool \_contenidoValido;
private string Conexion = @"Provider=Microsoft.Jet.OLEDB.4.0; Data Source
    =.\\Files\\SpaceRugby.mdb";
public bool ContenidoValido
```

```
    public string Entrenador
14
    //Métodos
16  public bool Cargar()
    public bool Guardar()
18  public bool Crear(string contraseña, string nombre, string apellidos,
        string email)
    public bool Eliminar(String contraseña)
20  public bool ContraseñaCorrecta(string contraseña)
    public bool CambiarContraseña(string contraseñaVieja, string
        contraseñaNueva)
22  public void Dispose()
```

Equipo

```
    //Atributos
 2  public string Nombre;
    private string _nombreBD;
 4  private string _entrenador;
    public string Raza;
 6  public int Victorias;
    public int Derrotas;
 8  public int Empates;
    public int Valoracion;
10  public int Dinero;
    public int SegundasOportunidades;
12  public int FactorDeHinchas;
    public int Puesto;
14  public List<JugadorEquipo> Jugadores;
    private bool _contenidoValido;
16  private string Conexion = @"Provider=Microsoft.Jet.OLEDB.4.0; Data Source
        =.\Files\SpaceRugby.mdb";
    public bool ContenidoValido
18  public string Entrenador

20  //Métodos
    public bool Cargar()
22  public bool Guardar()
    public bool Crear(String entrenador, String raza, int victorias, int
        derrotas, int empates, int valoracion, int dinero, int
        segundasOportunidades, int factorDeHinchas, int puesto)
24  public bool Eliminar()
    public string ConvertirEnCadena()
26  public void Dispose()
```

Jugador de un equipo

```
//Atributos
public string Nombre;
public string _nombreBD;
public string _equipo;
public int Numero;
public string Categoria;
public string Posicion;
public List<string> Habilidades;
public string Estado;
public int PasesCompletos;
public int Touchdowns;
public int Intercepciones;
public int HeridasCausadas;
public int PuntosMejorJugadorDelEncuentro;
public int PuntosDeExperiencia;
public bool HabilidadesGeneralesPermitidas;
public bool HabilidadesAgilidadPermitidas;
public bool HabilidadesFuerzaPermitidas;
public bool HabilidadesPasePermitidas;
public bool HabilidadesFísicasPermitidas;
public int Movimiento;
public int Fuerza;
public int Agilidad;
public int Armadura;
private string Conexion = @"Provider=Microsoft.Jet.OLEDB.4.0; Data Source
    =.\Files\SpaceRugby.mdb";
private bool _contenidoValido;
public string Equipo
public bool ContenidoValido
```

La clase *JugadorEquipo*, aparte de una clase ConvertirEnCadena, la cual almacena la información de un jugador (y sus habilidades) en una cadena de caracteres, posee dos métodos para manejar Habilidades. TieneHabilidad simplemente comprueba que la habilidad si la habilidad especificada por parámetro está en la lista de habilidades del jugador, en caso afirmativo devuelve verdadero y en caso de no existir en la lista devuelve falso. AñadirHabilidad es un método algo más interesante, primero llama al método TieneHabilidad para comprobar que la habilidad no existe ya en la lista de habilidades del jugador, y en ese caso la incluye en la lista de habilidades del jugador y la inserta en la tabla HabilidadesJugadorEquipo.

```
//Métodos
```

```
2 public bool Cargar()
  public bool Guardar()
4 public bool Crear(int numero, string categoria, string posicion, List<
    string> habilidades, string estado, int pasesCompletos, int touchdowns
    , int intercepciones, int heridasCausadas, int
    puntosMejorJugadorDelEncuentro, int puntosDeExperiencia, bool
    habilidadesGeneralesPermitidas, bool habilidadesAgilidadPermitidas,
    bool habilidadesFuerzaPermitidas, bool habilidadesPasePermitidas, bool
     habilidadesFisicasPermitidas, int movimiento, int fuerza, int
    agilidad, int armadura)
  public bool Eliminar()
6 public bool TieneHabilidad(string valor)
  public void AñadirHabilidad(string valor)
8 public string ConvertirEnCadena()
  public void Dispose()
```

4.4 Formalización del partido

Este apartado se centrará en la explicación de cómo se han transformado las reglas del partido en lógica dentro del programa. No interesa excesivamente explicar la reglas concretas que se han implementado sino ilustrar las clases generadas para su implementación.

Debe entenderse que dado que en nuestro juego existen dos roles (el de cliente y el de servidor), la formalización de las reglas será distinta en cada uno, debido a que la lógica debe estar contenida en la parte servidor en la medida de lo posible, y la parte cliente debe encargarse de solicitar al servidor que resuelva los movimientos y acciones. Por ello se enfocará la formalización del partido desde el punto de vista de la parte servidor y desde el punto de vista de la parte cliente.

4.4.1 Formalización de la parte servidor

Tablero

El tablero está formado por 390 casillas dispuestas en 26 columnas y 15 filas. Para poder representar el tablero, se le ha asignado a cada casilla un número de manera que el tablero se convierte en un array de números.

La forma de numeración no es casual ya que permite determinar matemáticamente las distintas zonas del tablero de juego.

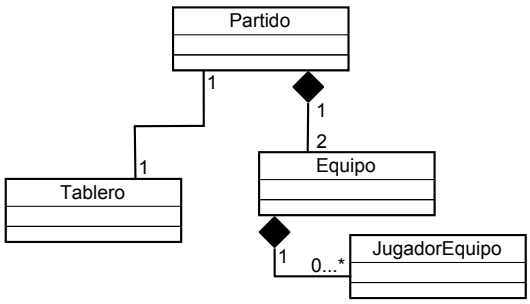


Figura 4.33: Formalización del partido en el servidor

El campo del equipo lanzador es el rango de casillas de la 0 a la 194 ambas inclusive. La línea de touchdown del campo del equipo receptor son las casillas de la 0 a la 14, la línea de medio campo está determinada por las casillas de la 184 a la 190, una casilla pertenece a la primera zona ancha del equipo lanzador si su valor es menor que 184 y el módulo resultante al dividir su valor entre 15 está entre 0 y 3; una casilla pertenece a la segunda zona si su valor es menor que 194 y el módulo resultante al dividir su valor entre 15 está entre 11 y 14.

El campo del equipo receptor está determinado por el rango de casillas de la 195 a la 389 ambas inclusive. La línea de touchdown del campo del equipo lanzador son las casillas de la 375 a la 189, la línea de medio campo está determinada por las casillas de la 195 a la 209; una casilla pertenece a la primera zona ancha del equipo receptor si su valor es mayor 194 y el módulo resultante de dividir su valor entre 15 está entre 0 y 3; una casilla pertenece a la segunda zona de defensa del equipo receptor su valor es superior a 206 y el módulo resultante al dividir su valor entre 15 está entre 11 y 14.

Los atributos de la clase se encuentran detallados a continuación:

```
1 public const int NUMERO_FILAS = 15;
  public const int NUMERO_COLUMNAS = 26;
3 public const int[,] Casillas = new int[NUMERO_FILAS, NUMERO_COLUMNAS];
```

Para representar las casillas del campo de juego se emplea una matriz bidimensional, cuyas dimensiones están determinadas por las constantes que indican el número de filas de la matriz (que vendría a definir el ancho del campo) y el número de columnas (el largo del campo). El contenido de la matriz serían los número ya indicados anteriormente.

A continuación se encuentran enumerados los métodos de la clase tablero:

```
1  public bool CasillaEnCampoEquipoLanzador(int casilla)
   public bool CasillaEnCampoEquipoReceptor(int casilla)
3  public bool CasillaEnLineaDefensaEquipoLanzador(int casilla)
   public bool CasillaEnLineaTouchDownEquipoLanzador(int casilla)
5  public bool CasillaEnLineaDefensaEquipoReceptor(int casilla)
   public bool CasillaEnLineaTouchDownEquipoReceptor(int casilla)
7  public bool CasillaEnPrimeraZonaAncha(int casilla)
   public bool CasillaEnSegundaZonaAncha(int casilla)
9  public List<int> ObtenerZonaDefensa(int casilla)
   public List<int> ObtenerCasillasCamino(int casillaOrigen, int
       casillaDestino)
11 public int Distancia(int casillaReferencia, int casilla)
   public int CalcularRebote(int numeroCasilla)
13 public int CalcularReboteCampoEquipoLanzador(int numeroCasilla)
   public int CalcularReboteCampoEquipoReceptor(int numeroCasilla)
```

Se realizará un breve repaso sobre su funcionalidad. Encontramos varios métodos que comienzan por "CasillaEn" y que sirven para determinar si la casilla correspondiente con el número pasado por parámetro se encuentra en las distintas líneas de touchdown, zonas de defensa o campo del equipo lanzador o equipo receptor; esto será importante para determinar por ejemplo si un jugador ha marcado un touchdown, o si se intenta colocar un jugador en el campo equivocado.

El método ObtenerZonaDefensa será útil cuando se quiere saber cuales son las casillas afectadas por un jugador que está defendiendo, así mismo ObtenerCasillasCamino es utilizada para calcular las casillas que atraviesa el balón en su trayectoria desde el jugador que efectúa el pase hasta el jugador que efectúa la recepción. Otro método que tiene que ver con los pases es el método Distancia, ya que mediante la distancia de dos casillas pueden determinarse la dificultad del pase.

Por último se encuentran los métodos de calcular rebote, que comienzan por "CalcularReboteCampo" y que dada una casilla calcula de forma aleatoria la casilla adyacente destino del rebote. Según el método, la casilla devuelta puede estar restringida a estar en el campo del equipo lanzador, del equipo receptor, o en cualquiera de los dos.

Partido

Es necesario que el servidor posea una clase partido, ya que necesita distinguir los distintos partidos que se están llevando a cabo en ese momento. Sobre la información que necesita el servidor acerca del partido, puede verse perfectamente en los atributos de la clase:

```
Equipo Equipo1;
Equipo Equipo2;
public bool Equipo1Preparado;
public bool Equipo2Preparado;
public Equipo EquipoLanzador;
public Equipo EquipoReceptor;
Tablero mesa;
public int Turno;
public int Parte;
public int Estado;
public int Balon;
```

Por un lado están Equipo1 y Equipo2, que son los participantes en el partido, pero también EquipoLanzador y EquipoReceptor, cuyo uso solo tiene sentido durante el partido. También se tiene dos booleanos que indican si los respectivos equipos están preparados, ya que una vez que se acepta la invitación a jugar el partido, se debe esperar a que se arranque la ventana de gráficos 3D en la parte cliente (lo cual requiere algún tiempo) y cuando ya se ha inicializado el cliente avisa que está listo; una vez que ambos equipos están preparados se comienza el partido.

Por último se encuentran los atributos que modelan los distintos componentes de la partida: el tablero, el contador de turnos, el contador de parte, la posición del balón y el estado del partido.

A continuación se encuentran enumerados los métodos de la clase partida:

```
public void ResetearEquipoPreparado()
public void ConfirmarEquipoFinalizado(string nombre)
public void ConfirmarEquipoPreparado(string nombre)
public void MediaParte()
public int D6()
public int D12()
public bool resolverTirada(int atributo, int tirada)
public bool PosicionesJugadores(string equipo, string posiciones)
public bool CasillaCampoEquipoReceptor(int casilla)
public bool CasillaCampoEquipoLanzador(int casilla)
public int CalcularRebote(int casilla)
public int CalcularReboteCampoEquipoLanzador(int casilla)
public int CalcularReboteCampoEquipoReceptor(int casilla)
public bool AccionMoverPermitida(string equipo,string jugador)
public bool AccionPlacarPermitida(string equipo, string jugador)
public bool AccionRecogerBalonPermitida(string equipo, string jugador)
public bool AccionPasarPermitida(string equipo, string jugador)
```

```
    public bool AccionGirarPermitida(string equipo, string jugador)
19  public bool AccionPonerEnPiePermitida(string equipo, string jugador)
    public void CalcularTiradaContraArmaduraMover(string equipo, Jugador
        jugador)
21  public void CambiarTurno(string equipo)
    public void Mover(string equipo, string jugador, string casillas)
23  public void Placar(string nombreEquipoAtacante, string
        nombreJugadorAtacante, string nombreJugadorDefensor)
    public void Pasar(string nombreEquipoPase, string nombreJugadorPase)
25  public void RecogerBalon(string nombreEquipo, string nombreJugador)
    public void Girar(string nombreEquipo, string nombreJugador)
27  public void PonerEnPie(string nombreEquipo, string nombreJugador)
    public void DatosJugador(string nombreEquipo, string nombreJugador)
```

Empecemos por los últimos métodos, que definen las acciones de los jugadores durante el partido. Para poder mover, pasar, recoger balón, girar, poner en pie y obtener los datos del jugador es preciso que se especifique el nombre del jugador y el nombre del equipo al que pertenece. En el caso del placaje es necesario especificar también el nombre del jugador defensor, del cual se asume que pertenece al equipo contrario al especificado.

Pueden encontrarse también los métodos que indican si el jugador puede realizar o no los distintos movimientos, estos métodos serán utilizados cuando el cliente solicite al servidor las acciones disponibles para un jugador.

Hay también tres métodos para calcular rebotes, que interiormente utilizan los métodos de la clase tablero, igual que sucede con los métodos que comienzan por "CasillaEnCampoEquipo".

El método PosicionesJugadores recibe como parámetros el nombre del equipo, y la relación de los jugadores y sus casillas; el método se encarga de comprobar que las posiciones de los jugadores pertenecen al campo correcto y que se cumple el reglamento.

El método ResolverTirada recibe como parámetros el valor del atributo y el valor de la tirada. Si el atributo es 1 entonces se necesita una tirada de 6, si es 2 la tirada se necesita que sea de 5 o superior, para un valor de 3 del atributo la tirada debe ser de 4 o más, para un valor de 4 la tirada tiene que ser de 3 o superior, para 5 se precisa un 2 o más en la tirada, y para un atributo de 6 vale cualquier valor superior o igual a 1. Para calcular el valor de la tirada puede usarse el método D6, que genera un número aleatorio entre 1 y 6 (ambos inclusive) y lo devuelve como resultado. El método D12 es igual que D6 pero genera valores del 1 al 12.

El método MediaParte es el método que se ejecuta cuando el servidor comprueba que se ha llegado a la mitad del partido.

ConfirmarEquipoPreparado es utilizado para saber cuando los equipos están preparados para empezar el partido, hasta que los dos equipos no confirman que están preparados no se procede a determinar cuál es el equipo lanzador y cuál el receptor, y empezar el partido. ConfirmarEquipoFinalizado se utiliza para que un equipo ha finalizado tanto el partido como el postpartido, cuando ambos equipos confirman que han finalizado el partido(las pantallas del postpartido incluidas), el servidor sabe que puede destruir el partido y que ambos usuarios pueden recibir y realizar invitaciones a para jugar partidos.

Cuando una tirada falla, salvo que se emplee una ficha de segunda oportunidad o que alguna habilidad indique lo contrario, se realiza una tirada contra armadura, que determinará el daño que ha recibido el jugador especificado por el nombre y el equipo al que pertenece. Aunque el método se denomine "CalcularTiradaContraArmaduraMover" se utiliza para los fallos ocurridos en cualquier acción.

Equipo

La clase equipo es la clase que maneja los jugadores del partido, las segundas oportunidades, y las acciones que lleva realizadas el equipo en el turno. Los atributos de la clase Equipo son los siguientes:

```
public string NombreEquipo;
public List<Jugador> Jugadores;
public int SegundasOportunidades;
public int \_segundasOportunidades;
public int FactorHinchas;
public int TouchDowns;
public bool SegundaOportunidadTurno;
public bool PaseTurno;
public bool PenetracionTurno;
```

Como puede verse tiene una lista de jugadores, que son los jugadores presentes en el partido (se encuentren o no en el terreno de juego); también tiene dos variables para las segundas oportunidades, la primera lleva la cuenta de las segundas oportunidades restantes del equipo (cuando el usuario emplea una ficha de segunda oportunidad se decrementa el valor) y la segunda lleva la cuenta de las segundas oportunidades iniciales, de esta manera cuando llega el cambio de turno se restablece el valor de las segundas oportunidades del equipo.

También posee un valor Factor de Hinchas que se utiliza en el postpartido, y la cuenta de touchdowns anotados en el partido por el equipo.

Por último se tienen una serie de atributos boolean que sirven para determinar si una acción ya ha sido realizada en dicho turno, el valor de dichas variables se restablece a falso en el cambio de turno de manera que cuando el equipo comienza un turno dichos valores son siempre falsos. Es necesario el uso de dichas variables porque un equipo solo puede usar una ficha de segunda oportunidad, realizar una acción de pasar o de penetración una única vez durante un turno.

Los métodos de la clase Equipo son:

```
1 public Jugador Jugador(string nombre)
  public Jugador Jugador(int casilla)
3 public List<Jugador> JugadoresEnZonaDefensa(int casilla)
  public List<Jugador> JugadoresEnPieEnZonaDefensa(int casilla)
5 public void CambioDeTurno()
  public void MitadPartido()
7 public void TouchDown()
  public bool ConfirmacionUso2aOP()
9 public bool ConfirmacionUsoProfesional(string nombreJugador)
  public bool ConfirmacionOcuparCasilla()
```

Los métodos con nombre "Jugador" se utilizan para obtener la instancia de un jugador del equipo por medio de su nombre o de la casilla que ocupa. De igual manera los métodos JugadoresEnZonaDefensa y JugadoresEnPieEnZonaDefensa devuelven en una lista las instancias de los jugadores del equipo que se encuentran en la zona de defensa determinada por la casilla dada por parámetro; la peculiaridad está en que JugadoresEnPieEnZonaDefensa sólo devuelve los jugadores que están en pie, esto es utilizado por ejemplo para calcular los modificadores.

El método CambioDeTurno se encarga de establecer a falso los atributos booleanos SegundaOportunidadTurno, PaseTurno y PenetraciónTurno.

El método MitadPartido se encarga de reestablecer el valor de SegundasOportunidades al valor de _segundasOportunidades, y de eliminar los jugadores del terreno de juego estableciendo su posición en la casilla -1.

El método TouchDown incrementa el valor de los TouchDown, elimina los jugadores del terreno de juego y llama al método CambioTurno.

Jugador

Son muchos los atributos que posee la clase Jugador, en primer lugar resaltaremos una serie de constantes que definen los estados en los que puede estar un jugador, y que son: en pie, derribado (tumbado boca abajo en el terreno de juego), aturdido (que se representa con el jugador boca arriba en lugar de boca abajo), inconsciente, herido leve, herido grave y muerto. En los casos en que el jugador se encuentre inconsciente, herido o muerto, se le retira del campo. El jugador tiene una variable Estado cuyo valor coincidirá con uno de los valores expresados en las líneas de la 1 a la 7:

```
public const int EN\_PIE = 0;
public const int DERRIBADO = 1;
public const int ATURDIDO = 2;
public const int INCONSCIENTE = 3;
public const int HERIDO\_LEVE = 4;
public const int HERIDO\_GRAVE = 5;
public const int MUERTO = 6;
public Equipo Equipo;
public string Nombre;
public int Casilla;
public int Estado;
public bool PoseeBalon;
```

En primer lugar el Jugador necesita tener una referencia a la instancia de su Equipo, almacenar su nombre, la casilla en la que se encuentra y su estado actual; también necesitamos determinar si un jugador posee o no el balón.

Las características como el movimiento (el movimiento extra), fuerza, agilidad, armadura y habilidades están almacenadas en variables. Dado que el movimiento del jugador se decrementa cuando éste se mueve (lo mismo pasa cuando se realiza un movimiento extra), y que una habilidad desaparece de la lista cuando se utiliza (para así evitar que vuelva a utilizarse), tenemos una copia del valor del movimiento, del movimiento extra y de las habilidades en _mo, _moExtra y _habilidades respectivamente.

```
public int MO;
public int MOExtra;
public int \_mo;
public int \_moExtra;
public int FU;
public int AG;
public int AR;
private List<string> Habilidades;
private List<string> \_habilidades;
```

La clase jugador almacena también datos sobre las acciones llevadas a cabo por el jugador para realizar estadísticas, se cuentan las intercepciones llevadas a cabo por el jugador, el número de pases completos, los placajes realizados con éxito y las heridas causadas.

```
1 public int Intercepciones;
  public int Pases;
3 public int Placajes;
  public int Heridas;
```

Por último están las variables sobre las acciones que ha realizado el jugador durante el turno, ya que existen algunas acciones que están restringidas como consecuencia de las acciones realizadas previamente, por ejemplo un jugador no puede moverse si previamente ha realizado y placaje. Estas variables se establecen como falsas al comienzo del turno.

```
  public bool HaPlacado;
2 public bool HaMovido;
  public bool HaPasado;
4 public bool HaPenetrado;
  public bool TerminoAccion;
```

Los métodos de la clase Jugador están orientados, como es lógico, a facilitar información acerca del jugador, por ejemplo saber si una casilla pertenece a la zona de defensa de un jugador mediante el uso de ZonaDefensa, si el jugador posee una habilidad gracias al método TieneHabilidad, u obtener la relación de las habilidades del jugador enumeradas en una cadena (CadenaHabilidades). Pero también se tienen métodos que actúan sobre el jugador, por ejemplo UsaHabilidad elimina la habilidad pasada por parámetro de la lista contenida en el atributo Habilidades, de esta forma si se pregunta por una habilidad del jugador que se ha usado, esta ya no estará; el método CambioTurno notifica que se produce un cambio de turno y que deben restaurarse aquellos atributos del jugador puedan haberse visto afectados por las acciones del jugador: MO se restablecerá al valor de _mo, MOExtra con _moExtra, Habilidades con el contenido de _habilidades, y los atributos booleanos (HaPlacado, HaMovido, HaPasado, HaPenetrado y TerminoAcción) se establecen con valor falso.

```
1 public bool ZonaDefensa(int casilla)
  public bool TieneHabilidad(string habilidad)
3 public string CadenaHabilidades()
```

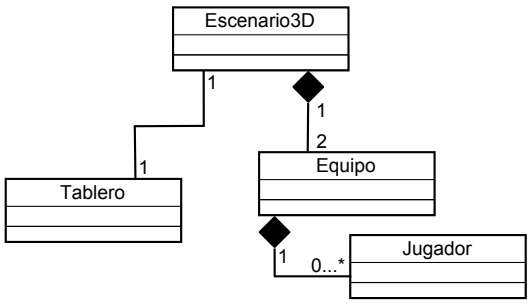


Figura 4.34: Formalización del partido en el cliente

```
    public void UsaHabilidad(string habilidad)
5   public void CambioTurno()
```

4.4.2 Formalización de la parte cliente

Lo primero que se observa es el hecho de que las clases de la parte cliente tienen menos métodos y atributos que sus correspondientes en la parte servidor, ya que mientras en el servidor se lleva a cabo todo el control de la lógica y reglas del partido, en la parte de cliente se realiza el control mínimo y necesario para poder jugar. Además en la parte cliente no existe una clase Partido como existía en la parte servidor, y es lógico ya que mientras que el servidor debe controlar multitud de partidas a la vez, la parte cliente solo se encarga únicamente de la partida que está jugando por lo que no se necesita una clase para representar el partido; y ya que no hay clase partido, la lógica está controlada desde la clase principal de manejo de gráficos Escenario3D, esto tiene sentido por que además de la lógica se controla también parte de los gráficos del juego con dichas clases.

Tablero

La clase tablero es en parte parecida a la clase tablero, se tiene una matriz bidimensional cuyo tamaño está definido por una constante NUMERO_FILAS de valor 15 y que representa el ancho del terreno de juego; y otra constante NUMERO_COLUMNAS de valor 26 y que representa el largo, además de otra constante TAMAÑO, que es utilizada para determinar el tamaño de las casillas y posición de las casillas.

La clase tiene una matriz denominada casillas, cuyos elementos no son números como en la parte servidor, sino una instancia de SceneNode que son los elementos

gráficos del juego. Se tiene también dos matrices de nombre Entidades1 y Entidades2 que tienen el mismo tamaño que la matriz de Casillas, y que son elementos Entity, también relacionado con los elementos gráficos del juego. Para que el lector pueda hacerse una idea, una entidad contiene la información de representación de un elemento gráfico (normalmente la definición de un conjunto de puntos y planos); el que haya dos matrices de entidades es debido a que hay dos tipos de casillas.

Pero las casillas no son los únicos elementos gráficos que forman el tablero, ya que la clase Tablero también se encarga de visualizar las líneas de campo por medio de los atributos LineasCampo (el SceneNode) y Líneas (las entidades). Estas líneas son las que marcan la separación de la mitad de campo, las líneas de touchdown y las líneas de separación de las zonas anchas.

```
1 public const int NUMERO\_FILAS = 15;
  public const int NUMERO\_COLUMNAS = 26;
3 public const int TAMAÑO = 50;
  public Entity[,] Entidades1 = new Entity[NUMERO\_FILAS, NUMERO\_COLUMNAS];
5 public Entity[,] Entidades2 = new Entity[NUMERO\_FILAS, NUMERO\_COLUMNAS];
  public List<Entity> Lineas;
7 public SceneNode[,] Casillas = new SceneNode[NUMERO\_FILAS, NUMERO\
    _COLUMNAS]
  private SceneNode LineasCampo;
```

La clase tablero ofrece un método ZonaDefensa, que dada una casilla devuelve en una lista los enteros correspondientes a las casillas afectadas por su zona de defensa, y que es equivalente al método ObtenerZonaDefensa de la clase tablero de la parte servidor. El resto de métodos de la clase tablero están orientados a mostrar u ocultar casillas, así pues CasillasOcultas oculta todas las casillas del tablero, contrariamente a CasillasVisibles que las muestra todas las casillas del terreno de juego; los métodos que comienzan por "MostrarCampoEquipo" muestran respectivamente las casillas del campo del equipo según sea lanzador o receptor respectivamente, ocupándose además de ocultar las casillas del campo del equipo contrario; estos métodos son usados para ayudar al usuario a identificar en cuales casillas puede colocar a los jugadores, o la casilla a la cual puede enviar el balón en la patada inicial.

```
  public List<int> ZonaDefensa(int numeroCasilla)
2 public void CasillasOcultas()
  public void CasillasVisibles()
4 public void MostrarCampoEquipoReceptor()
```

```
public void MostrarCampoEquipoLanzador()
```

Equipo

La clase Equipo es muy simple, almacena el nombre del equipo, una lista de sus jugadores y un booleano para saber si el entrenador del equipo fue invitado a jugar el partido (en caso de que su valor sea verdadero) o si por el contrario fue él quien invitó al otro entrenador a jugar el partido (en el caso que su valor sea falso).

```
1 public string NombreEquipo;
  public List<Jugador> Jugadores;
3 public bool Invitado;
  public Jugador Jugador(string nombre)
5 public Jugador Jugador(int casilla)
```

La clase sólo posee métodos para obtener los jugadores del equipo, ya sea por su nombre o por la casilla en la que se encuentra. Si comparamos con la clase Equipo de la parte servidor, vemos que no se recoge información alguna sobre las acciones que ha realizado el equipo, o sobre las segundas oportunidades, etc. Todo ello no necesita saberlo la parte cliente, a la parte cliente sólo le interesa saber el nombre del equipo para poder identificar a sus jugadores.

Jugador

En la clase Jugador, tal como ocurrió en la clase Tablero, se tiene tanto información referente a los datos de los jugadores como a los gráficos del jugador. Los datos que se poseen del jugador son su nombre, la casilla en la que se encuentra y si es jugador de un equipo invitado, no tenemos ni siquiera una referencia a la instancia de la clase equipo a la que pertenece tal como pasaba en la parte servidor (y bien pensado dado que la clase equipo apenas tiene información tampoco resultaría de mucha utilidad). Por último hay una variable de tipo JugadorRugby, sin entrar en detalles JugadorRugby es una clase en la que se tiene encapsulada todo el manejo de los jugadores.

```
1 public string NombreJugador;
  public string NombreNodo;
3 public int Casilla;
  public JugadorRugby nodoJugador;
5 public bool JugadorInvitado;
  public void CrearNodoJugador(Vector3 posicion)
7 public void DestruirNodoJugador()
```

Los métodos de la clase tienen que ver con la creación y destrucción del atributo nodoJugador. Nótese que para crear el nodo del jugador es necesario determinar la posición donde debe aparecer.

4.5 Diseño de las comunicaciones

Este apartado se adentrará en la forma que deben implementarse las comunicaciones del juego.

En primer lugar se hará un análisis de los agentes que intervienen en las comunicaciones del juego, y de las relaciones de comunicación entre ellos.

A continuación se detallará como se han configurado los elementos de .NET Remoting en la implementación de dichas relaciones.

También se procederá en este capítulo a detallar los servicios ofrecidos tanto por el servidor como por la parte cliente.

Por último se procederá a detallar la interacción de la parte cliente y el servidor mediante diagramas de interacción.

4.5.1 Participantes de la comunicación

A la hora de averiguar cuales son los participantes de los procesos de comunicación la respuesta resulta obvia, se tiene un proceso servidor que contiene la funcionalidad de juego, y unos procesos clientes que se conectan al servidor y que además es sabido que no se comunican entre ellos; no es muy difícil concluir por lo tanto que los participantes en un proceso de comunicación dentro del juego son el servidor y el cliente.

Como ha podido apreciarse en el apartado anterior, .NET Remoting se basa en un sistema de comunicación Cliente-Servidor, que era el modelo de comunicación elegido en el apartado 3.3.5; y dado que uno de los participantes se llama servidor y el otro cliente, uno puede llegar a la conclusión de que no hay más, el proceso servidor actúa de servidor en la comunicación y el proceso cliente actúa como solicitante del servicio; y aunque esto se ha podido por ejemplo cuando un cliente quiere loguearse en el sistema, resulta incompleto ya que no se trata de una relación de comunicación en un solo sentido, en la cual el cliente siempre es el que solicita un servicio, y el servidor el que lo resuelve, existe el caso contrario en el que el servidor realiza una petición a un cliente (por ejemplo que acepte o rechace una invitación a jugar un partido).

Llegados a este punto se puede pensar, si la comunicación es bidireccional ¿Por qué se ha elegido el modelo cliente-servidor y no peer-to-peer? Esta objeción podría tener fundamento si no fuera porque los roles de cliente y servidor son distintos, ya que un modelo peer-to-peer se basa en que todos los participantes de la comunicación tienen el mismo rol. Aquí existen dos roles diferenciados, y por lo tanto habrá dos servidores uno de servicios de servidor y otro de servicios de cliente.

4.5.2 Configuración de .NET Remoting para las comunicaciones

A continuación se detallará la configuración elegida a la hora de usar .NET Remoting.

En primer lugar se ha optado por el uso del canal TCP, ya que dado que el uso del canal HTTP con vistas a atravesar firewalls no se prevé necesario, elegimos el canal TCP que es más rápido.

La serialización que se usará para las comunicaciones será la Binaria, que es la serialización por defecto del canal TCP.

El puerto del servidor es configurable, y se ha elegido el puerto 9988. El puerto del cliente se elige en tiempo de ejecución, intentado en primer lugar hacerse con el puerto 8899, en caso de no conseguirlo genera un número aleatorio entre 2000 y 50000 e intenta hacerse con dicho puerto, si no lo consigue vuelve a producir otro número aleatorio, y así sucesivamente hasta un máximo de 100 intentos, si al cabo de esos 100 intentos no se ha conseguido ningún puerto para su uso el programa cliente finaliza.

La forma en que se registran los objetos remotos tanto en el cliente como en el servidor es por medio de Marshaling, de esta manera cada uno crea su instancia de implementación de su interfaz remota, y la asocian en el canal con una cadena de identificación. La cadena de identificación del objeto servidor es "SpaceRugbyServer" y la del objeto cliente es "SpaceRugbyClient".

Obsérvese que al ser el propio servidor y el propio cliente quien registra el objeto mediante Marshaling, ya no se produce la activación de objetos (puesto que estos ya existen), por que tendremos un único objeto que responderá todas las peticiones (que es lo que buscábamos). Pero existe un problema con los objetos registrados con Marshal que tiene que ver con el tiempo del vida del objeto, ya que .NET Remoting se encarga de destruir los objetos registrados cuando ha pasado cierta cantidad de tiempo desde la última vez que un cliente solicitó una invocación de dicho objeto; normalmente al registrar un objeto se indica su forma de activación, pero con Marshal no es asi porque

el objeto ya está activo, lo cual lleva a que cuando .NET Remoting destruye dicho objeto por inactividad, si un cliente quiere usarlo se producirá un error. La forma de evitar que .NET Remoting elimine un objeto por inactividad es que en la clase se sobrescriba el método para inicializar el tiempo de vida del objeto:

```
1 public override object InitializeLifetimeService()
  {
3     return null;
  }
```

Para terminar queda mencionar la forma en que los clientes obtienen la dirección y el puerto en que se localiza el programa servidor, la forma que se ha elegido es incluyendo esos dos parámetros en un fichero denominado SpaceRugby.ini que tiene el siguiente formato:

```
                [RED]
2               Servidor=163.117.101.160
                Puerto=9988
```

El programa servidor configura el puerto en el cual se arranca mediante su propio archivo de configuración SpaceRugby.ini que tiene el siguiente formato:

```
1               [RED]
                Puerto=9988
```

4.5.3 Servicios parte Servidora

En la parte servidora se ha desarrollado una clase remota que extiende de MarshalByRefObject, y que tiene todos sus métodos abstractos, posteriormente se creó la clase ImplementaciónServidor que hereda de IServidor e implementa todos sus métodos.

A continuación se muestra una descripción de los servicios de la parte servidora, agrupados por los casos de usos con los que están relacionados.

4.5.3.1 Servicios para los usuarios sin loguear

Loguearse en el sistema

```
        String SolicitudAutenticacion(string usuario)
```

Parámetros: usuario es el Nick del usuario que se desea autenticar.

Acción: Se verifica si el Nick de usuario existe, en caso afirmativo se genera una cadena de 50 caracteres.

Resultado: el servidor devuelve la cadena generada en el caso de que exista el Nick, en caso de no existir devuelve la cadena "0".

```
1    int Login(string usuario, string md5, string cadenaConexion)
```

Parámetros: El parámetro usuario es el Nick del usuario; md5 es como su nombre indica un resumen MD5 de la cadena obtenida en la solicitud de autenticación y la contraseña; y cadenaConexión es una cadena de caracteres con la cadena de conexión que nos permitirá obtener un Proxy del objeto remoto del cliente.

Acción: El servidor genera un resumen MD5 con la concatenación de la cadena que envió al usuario y su contraseña, si el resultado es igual que el md5 indicado por el usuario entonces se genera un número entre 20 y 1000 que será el identificador de sesión del usuario.

Resultado: si no se ha conseguido autenticar al usuario (ya sea por el usuario no existe o porque el resumen MD5 está mal) se devuelve -1, el identificador generado.

Solicitar nueva contraseña

```
1    Bool SolicitarNuevaContraseña(string usuario)
```

Parámetros: usuario es el nick del usuario que quiere obtener una nueva contraseña.

Acción: Se genera una nueva contraseña para el usuario y se le envía a la dirección de correo indicada en la cuenta del usuario.

Resultado: si se ha conseguido con éxito cambiar la contraseña y enviar el correo al usuario se devuelve true, en caso contrario se devuelve false.

Registrarse

```
1    String RegistrarNuevoUsuario(string nick, string contraseña,
         string nombre, string apellidos, string email)
```

Parámetros: el nombre del parámetro corresponde directamente con la información de la cuenta a la que representa.

Acción: El servidor intenta crear la cuenta, si falla entonces avisa al usuario que el nick está ocupado.

Resultado: Si el nick de usuario ya está ocupado devuelve la cadena " - El nick de usuario ya existe, por favor elija otro", sino devuelve "OK".

4.5.3.2 Servicios para los usuarios logueados

Ver mensajes chat

```
1    String ObtenerMensajesChat(int identificador)
```

Parámetros: identificador es el número identificador de sesión del usuario.

Acción: Si es la primera vez que se solicita los mensajes de chat, entonces devuelve los 10 últimos mensajes; en caso contrario devuelve todos los mensajes que se han producido desde la última vez que se solicitaron los mensajes de chat.

Resultado: Si el identificador de sesión no es válido devuelve "Sesión de usuario inválida", en caso contrario devuelve el texto de los mensajes de chat con el formato que se deben mostrar en la pantalla de usuario.

Escribir mensaje chat

```
1    String EnviarMenajeChat(int identificador, string mensaje)
```

Parámetros: identificador es el número identificador de sesión del usuario; mensaje es el texto del mensaje que se desea enviar.

Acción: Almacena el mensaje en el sistema registrando el usuario, el tiempo y el mensaje.

Resultado: Si el identificador de sesión no es válido devuelve "Sesión de usuario inválida", en caso contrario devuelve los mensajes de chat desde la última vez que se solicitaron (o en caso de ser la primera vez devuelve los 10 últimos mensajes).

Ver usuarios conectados

```
1        String ObtenerUsuariosConectados(int identificador)
```

Parámetros: identificador es el número identificador de sesión del usuario.

Acción: Si es la primera vez que se solicita los usuarios conectados se devuelve la lista de los nick de todos los usuarios conectados (incluido quien la solicita), en caso de haber ya solicitado los usuarios conectados solo se devuelven los cambios en la lista. El formato de la lista consiste en una repetición de carácter '+' o '-' seguidos del nick de un usuario, un carácter '+' significa que el nick se añade a la lista, y un carácter '-' que se elimina de la lista de usuarios conectados.

Resultado: Si el identificador de sesión no es válido devuelve "Sesión de usuario inválida", en caso contrario devuelve la cadena con la información de los usuarios conectados.

Ver ranking equipos

```
1        String ObtenerRanking(int identificador)
```

Parámetros: identificador es el número identificador de sesión del usuario.

Acción: Obtiene de base de datos la lista del ranking del equipo conforme se expresó en (CITA VENTANITAS). La lista está compuesta por registros del tipo $Puesto @Entrendor @Equipo @Raza @Victorias @Derrotas @Empates @Valoracion, una por cada equipo de la lista.

Resultado: Si el identificador de sesión no es válido devuelve "Sesión de usuario inválida", sino devuelve la lista de ranking.

Buscar equipo

```
1        String BuscarRankingEquipo(int identificador, string puesto,
             string entrenador, string equipo, string raza, string
             victorias, string derrotas, string empates, string valoracion
             )
```

Parámetros: identificador es el número identificador de sesión del usuario; el resto de parámetros son los criterios de búsqueda.

Acción: Obtiene de base de datos los equipos que coinciden con los criterios de búsqueda especificados en los parámetros. Para evitar filtrar por un criterio de búsqueda de los parámetros (por ejemplo empates) hay que establecerlo a nulo y entonces dicho criterio no se tendrá en cuenta. La lista de resultados de la búsqueda está compuesta por registros del tipo $Puesto @Entrendor @Equipo @Raza @Victorias @Derrotas @Empates @Valoracion, una por cada equipo resultante.

Resultado: Si el identificador de sesión no es válido devuelve "Sesión de usuario inválida", sino devuelve la lista de los resultados de la búsqueda.

Cambiar datos personales

```
1   String ObtenerDatosPersonales(int identificador)
```

Parámetros: identificador es el número identificador de sesión del usuario.

Acción: Construye una cadena con los datos personales de la cuenta. La cadena es del estilo: #Nick#Nombre#Apellidos#Email

Resultado: Si el identificador de sesión no es válido devuelve "Sesión de usuario inválida", sino devuelve los datos personales en una cadena de caracteres.

```
1   Bool CambiarDatosPersonales(int identificador, string nombre,
        string apellidos, string contraseñaAntigua, string
        contraseñaAntigua, string contraseñaNueva, string email)
```

Parámetros: identificador es el número identificador de sesión del usuario; el resto de parámetros son los nuevos datos de la cuenta a excepción de contraseñaAntigua que es la contraseña actual.

Acción: Si la contraseña antigua coincide con la contraseña válida en el momento del cambio de los datos, se procede a cambiar los datos.

Resultado: Si la contraseña antigua coincide con la contraseña válida en el momento del cambio de los datos y se ha conseguido cambiar los datos se devuelve true, en caso contrario se devuelve false.

Invitar usuario a jugar partido

```
1        InvitarAJugarPartido(int identificador, string usuario, string
             tipoPartido)
```

Parámetros: identificador es el número identificador de sesión del usuario; usuario es el nick del usuario al que se desea invitar y tipoPartido es "Oficial" o "Amistoso". La diferencia es que en los partidos de tipo Oficial, el valor de la valoración de los equipos puede verse afectada como resultado del partido, mientras que en los partidos amistosos esto no es así.

Acción: Comprueba que el usuario al que se desea invitar está conectado, y que ambos están en estado LOGUEADO. En caso afirmativo envía la invitación al usuario pasado por parámetro, en caso contrario rechaza la invitación del usuario a jugar el partido.

```
1        ResponderInvitacionAJugarPartido(int identificador, string usuario
             , bool respuesta)
```

Parámetros: identificador es el número identificador de sesión del usuario; usuario es el nick del usuario a quien se desea responder, y respuesta es la respuesta a la invitación.

Acción: Se comprueba que ambos jugadores están conectados, que el estado del usuario que responde es INVITADO_A_JUGAR_PARTIDO y que el estado del usuario al que se va a responder es ESPERANDO_RESPUESTA_INVITACION, así como que el usuario al que se desea responder está esperando una respuesta del usuario que indica el identificador; en caso afirmativo envía la respuesta.

Obtener estadísticas

```
1        String ObtenerEstadisticas(int identificador)
```

Parámetros: identificador es el número identificador de sesión del usuario.

Acción: Obtiene el equipo del usuario con el identificador pasado por parámetro y rellena en una cadena los datos del equipo. La cadena tiene el formato: NombreEquipo #Raza #Victorias #Derrotas #Empates #Valoracion #Dinero #CadenaJugadores #SegundasOportunidades #FactorDeHinchas. CadenaJugadores es una cadena con el siguiente formato: $Número @Nombre @Categoria @Posicion @MO @FU @AG @AR @CadenaHabilidades @Estado @PasesCompletos @TouchDowns @Intercepciones @HeridasCausadas @PuntosMejorJugadorEncuentro @PuntosExperiencia. CadenaHabilidades es una sucesión de las habilidades del jugador separadas por ','.

Resultado: Si no existe ningún usuario con tal identificador o si existe pero no tiene equipo devuelve la cadena vacía, en caso contrario devuelve los datos del equipo en una cadena de caracteres.

Crear nuevo equipo

```
1    String ObtenerListaRazas(int identificador)
```

Parámetros: identificador es el número identificador de sesión del usuario.

Acción: Almacena en una cadena la sucesión de las razas disponibles separadas por ','.

Resultado: Si el identificador de sesión no es válido devuelve "Sesión de usuario inválida", en caso contrario devuelve la cadena con las razas soportadas por la aplicación.

```
1    String ObtenerPrecioSegundasOportunidadesYFactorHinchas(int
         identificador)
```

Parámetros: identificador es el número identificador de sesión del usuario.

Acción: Busca el valor del factor de hinchas y las segundas oportunidades para la raza del equipo del jugador.

Resultado: Si el identificador de sesión no es válido devuelve "Sesión de usuario inválida", en caso contrario devuelve el precio de las segundas oportunidades y factor de hinchas.

```
1   Bool CrearEquipo(int identificador, string nombre, string raza)
```

Parámetros: identificador es el número identificador de sesión del usuario; nombre es el nombre que se quiere dar al equipo y raza es un valor de la lista de razas disponibles.

Acción: En primer lugar comprueba que la raza especificada existe; luego si el jugador tiene ya un equipo se elimina y por último se crear el nuevo equipo.

Resultado: Devuelve false si la raza especificada no existe, si no se pudo eliminar el equipo del jugador o si el equipo nuevo no fue creado, en caso contrario devuelve true.

```
1   String ObtenerPosicionesJugadores(int identificador)
```

Parámetros: identificador es el número identificador de sesión del usuario.

Acción: Para todas las posiciones de la raza del equipo de jugador, se concatena el valor de su cantidad máxima por equipo, el nombre de la posición, el precio, su movimiento, fuerza, agilidad, armadura, habilidades, si tiene permitidas las habilidades generales, las de agilidad, las de fuerza, las habilidades de pase y las habilidades físicas. La información de cada posición se almacena con el siguiente formato y se concatena a la descripción de las demás posiciones: $CantidadMaxima @Posicion @Precio @MO @FU@AG @AR @ListaHabilidades @HabilidadesGeneralesPermitidas @HabilidadesAgilidadPermitidas @HabilidadesFuerzaPermitidas @HabilidadesPasePermitidas @HabilidadesFisicasPermitidas La lista de habilidades es una enumeración de las habilidades separadas por ','.

Resultado: Si el identificador de sesión no es válido devuelve "Sesión de usuario inválida", en caso contrario devuelve la cadena de caracteres con la información de las posiciones disponibles para esa raza.

```
1   Bool ComprarNuevoJugador(int identificador, string nombre, string
        posicion)
```

Parámetros: identificador es el número identificador de sesión del usuario; nombre es el nombre que se quiere dar al jugador y posición es la posición en la que juega dentro del equipo.

Acción: Primero comprueba que el jugador tiene creado un equipo, y que la posición indicada existe dentro de la lista de posiciones determinadas para la raza del equipo; a continuación comprueba que el nombre del jugador no exista ya en el equipo y que no se haya alcanzado el máximo de jugadores que se pueden tener en dicha posición; y por último crea el jugador.

Resultado: False si ocurre alguna de las siguientes cosas: el jugador no tenga creado un equipo, la posición no es una de las disponibles para esa raza, el nombre del jugador ya existe en el equipo o se ha alcanzado el número máximo permitido de jugadores en dicha posición en el equipo. En caso contrario devuelve true.

```
1    Bool VenderNuevoJugador(int identificador, string nombre)
```

Parámetros: identificador es el número identificador de sesión del usuario; nombre es el nombre del jugador que se quiere eliminar.

Acción: Primero comprueba que el jugador tiene creado un equipo y que en ese equipo hay un jugador con ese nombre, a continuación lo elimina y reintegra el valor del jugador al dinero del equipo.

Resultado: False si ocurre alguna de las siguientes cosas: el jugador no tenga creado un equipo, el equipo del jugador no tiene jugadores o no tiene ningún jugador con ese nombre, o no se consiguió eliminar al jugador. En caso contrario devuelve true.

```
1    Bool ComprarSegundasOportunidades(int identificador, int cantidad)
```

Parámetros: identificador es el número identificador de sesión del usuario, cantidad es el número de unidades que se desean comprar

Acción: Primero comprueba que el usuario tiene creado un equipo y a continuación comprueba que dispone del dinero suficiente para comprar la cantidad especificada de segundas oportunidades, por último realiza la compra y descuenta el precio del dinero del equipo.

Resultado: False si el usuario no tiene creado un equipo o suficiente dinero para comprar la cantidad especificada de segundas oportunidades, en caso contrario devuelve true.

```
1    Bool VenderSegundasOportunidades(int identificador, int cantidad)
```

Parámetros: identificador es el número identificador de sesión del usuario, cantidad es el número de unidades que se desean vender.

Acción: Primero comprueba que el usuario tiene creado un equipo y a continuación comprueba que dispone de la cantidad suficiente de segundas oportunidades para realizar la venta, por último realiza la venta y suma el valor obtenido de la venta al dinero del equipo.

Resultado: False si el usuario no dispone de un equipo o la cantidad de segundas oportunidades para vender, en caso contrario devuelve true.

```
1    Bool ComprarFactorDeHinchas(int identificador, int cantidad)
```

Parámetros: identificador es el número identificador de sesión del usuario, cantidad es el número de unidades que se desean comprar

Acción: Primero comprueba que el usuario tiene creado un equipo y a continuación comprueba que dispone del dinero suficiente para comprar la cantidad especificada de factor de hinchas, por último realiza la compra y descuenta el precio del dinero del equipo.

Resultado: False si el usuario no tiene creado un equipo o suficiente dinero para comprar la cantidad especificada de factor de hinchas, en caso contrario devuelve true.

```
1    Bool VenderFactorDeHinchas(int identificador, int cantidad)
```

Parámetros: identificador es el número identificador de sesión del usuario, cantidad es el número de unidades que se desean vender

Acción: Primero comprueba que el usuario tiene creado un equipo y a continuación comprueba que dispone de la cantidad suficiente de factor de hinchas para realizar la venta, por último realiza la venta y suma el valor obtenido de la venta al dinero del equipo.

Resultado: False si el usuario no dispone de un equipo o la cantidad de factor de hinchas para vender, en caso contrario devuelve true.

```
1    EquipoTerminado(int identificador)
```

Parámetros: identificador es el número identificador de sesión del usuario.

Acción: Guarda los datos del equipo y jugadores del usuario.

Jugar partido

```
1    Bool Posiciones(int identificador, string posicionesJugadores)
```

Parámetros: identificador es el número identificador de sesión del usuario, posicionesJugadores es una cadena que contiene los valores de los nombres de los jugadores con el formato: @NombreJugador #Casilla

Acción: Se obtienen los nombres y posiciones de los jugadores, a continuación se comprueba que existan los jugadores nombrados y las casillas indicadas. Por último se comprueban que las posiciones cumplen con el reglamento.

Resultado: True si las posiciones son correctas, false en caso contrario.

```
1    Bool Patada(int identificador, int casilla, string jugador)
```

Parámetros: identificador es el número identificador de sesión del usuario; casilla indica la casilla a la que se va a lanzar el balón, jugador indica el nombre del jugador que va a realizar la patada.

Acción: Comprueba que el jugador existe y que al equipo del usuario le toca hacer la patada. A continuación comprueba que se cumplen las condiciones del reglamento para efectuar la patada.

Resultado: True si se puede efectuar la patada, false en caso contrario.

```
1    String AccionesJugador(int identificador, string nombreJugador)
```

Parámetros: identificador es el número identificador de sesión del usuario; nombreJugador es el nombre del jugador del cual se desea conocer que acciones puede hacer.

Acción: Se evalúa si el jugador puede realizar la acción de girarse, ponerse en pie, mover, pasar, placar y recoger balón. Aquellas acciones que pueda hacer se guardan en una cadena de caracteres separadas por el carácter '#'.

Resultado: Las acciones que el jugador puede hacer, en caso que no pueda hacer nada devuelve la cadena vacía.

```
1    MoverJugador(int identificador, string jugador, string casillas)
```

Parámetros: identificador es el número identificador de sesión del usuario; jugador es el nombre del jugador que se desea mover; y casillas es una sucesión de las casillas por las que se desea que el jugador se mueva, separadas por el carácter '#'.

Acción: Se comprueba que el jugador que se desea mover existe, y que es el turno del equipo del usuario, a continuación se efectúan las comprobaciones especificadas en el reglamento.

```
1    FinalizaTurno(int identificador)
```

Parámetros: identificador es el número identificador de sesión del usuario.

Acción: Si estamos en la primera parte y la suma de los turnos de los equipos no ha llegado a 16 entonces el turno pasa a ser del equipo contrario, si ha llegado a 16 entonces se ha llegado a la mitad del partido. Si estamos en la segunda parte y la suma de los turnos de los eqiupos no ha llegado a 16 entonces el turno pasa a ser del equipo contrario, si son 16 entonces el partido ha acabado.

```
1    RecogerBalon(int identificador, string jugador)
```

Parámetros: identificador es el número identificador de sesión del usuario; jugador es el nombre del jugador que se desea que recoga el balón.

Acción: Se comprueba que el jugador existe, que es el turno del equipo del usuario y que el balón y el jugador están en la misma casilla, a continuación se efectúan las comprobaciones especificadas en el reglamento.

```
1    Placar(int identificador, string nombreJugador)
```

Parámetros: identificador es el número identificador de sesión del usuario; jugador es el nombre del jugador con el que se desea realizar el placaje.

Acción: Se comprueba que el jugador existe y que es el turno del equipo del usuario, a continuación se efectúan las comprobaciones especificadas en el reglamento.

```
1    Pasar(int identificador, string nombreJugador)
```

Parámetros: identificador es el número identificador de sesión del usuario; jugador es el nombre del jugador con el que se desea realizar el pase.

Acción: Se comprueba que el jugador existe y posee el balón, y que es el turno del equipo del usuario, a continuación se efectúan las comprobaciones especificadas en el reglamento.

```
1    Girar(int identificador, string nombreJugador)
```

Parámetros: identificador es el número identificador de sesión del usuario; jugador es el nombre del jugador que se desea girar.

Acción: Se comprueba que el jugador existe, que se encuentra aturdido y que es el turno del equipo del usuario; a continuación se efectúan las comprobaciones especificadas en el reglamento.

```
1    PonerEnPie(int identificador, string nombreJugador)
```

Parámetros: identificador es el número identificador de sesión del usuario; jugador es el nombre del jugador que se desea poner en pie.

Acción: Se comprueba que el jugador existe, que se encuentra derribado y que es el turno del equipo del usuario; a continuación se efectúan las comprobaciones especificadas en el reglamento.

```
1    ListoParaEmpezarPartido(int identificador)
```

Parámetros: identificador es el número identificador de sesión del usuario.

Acción: Se registra que el equipo del usuario está preparado para empezar el partido, si el equipo adversario también ha informado que está listo para empezar, se procede a informar de los jugadores del partido en cada equipo y a designar al equipo lanzador y receptor.

```
1    String DatosJugador(int identificador, string nombreJugador)
```

Parámetros: identificador es el número identificador de sesión del usuario; jugador es el nombre del jugador del cual se desea obtener información.

Acción: Se obtienen los datos sobre el nombre, el movimiento, la fuerza, agilidad, armadura y habilidades del jugador especificado y se almacenan en una cadena de caracteres con el siguiente formato: #Nombre #MO #FU #AG #AR @[lista_habilidades] La lista de habilidades es una sucesión de las habilidades del jugador separadas por '@'.

Resultado: La cadena con los datos del jugador, o la cadena vacía si el jugador especificado no existe.

```
1    AbandonarPartido(int identificador)
```

Parámetros: identificador es el número identificador de sesión del usuario.

Acción: Se detiene el partido y se va al postpartido.

```
1    String ResultadoPartido(int identificador)
```

Parámetros: identificador es el número identificador de sesión del usuario.

Acción: De determina el vencedor del partido a razón del mayor número de touchdowns marcados.

Resultado: Se devuelven en una cadena de caracteres separados por el carácter '#' el nombre del equipo lanzador, los touchdowns marcados por el equipo lanzador, el nombre del equipo receptor, los touchdowns marcados por el equipo receptor, y por último el mensaje con el resultado del partido.

```
1    String Recaudacion(int identificador)
```

Parámetros: identificador es el número identificador de sesión del usuario.

Acción: Se calcula la recaudación del equipo para el partido que acaba de terminar.

Resultado: El valor de la recaudación en una cadena de caracteres.

```
1    String HeridasJugadores(int identificador)
```

Parámetros: identificador es el número identificador de sesión del usuario.

Acción: Se examinan aquellos jugadores del equipo que han resultado heridos o muertos durante el partido.

Resultado: Por cada jugador herido se almacena en una cadena de caracteres el nombre del jugador, el tipo de herida y el efecto de la herida con el siguiente formato: @NombreJugadorHerido #TipoHerida #EfectoHerida#

```
1    String FactorHinchasResultante(int identificador)
```

Parámetros: identificador es el número identificador de sesión del usuario.

Acción: Se calcula el factor de hinchas para el equipo, teniendo en cuenta el resultado del partido.

Resultado: Se devuelve el nuevo factor de hinchas en una cadena de caracteres.

```
1    String ValorResultanteEquipo(int identificador)
```

Parámetros: identificador es el número identificador de sesión del usuario.

Acción: En caso de que el partido sea oficial, se calcula la valoración del equipo en relación con el resultado del partido que ha realizado el equipo.

Resultado: Se devuelve la nueva valoración del equipo en una cadena de caracteres.

```
1    Bool VenderJugador(int identificador, string nombre)
```

Parámetros: identificador es el número identificador de sesión del usuario; nombre es el nombre del jugador que se desea despedir.

Acción: Primero comprueba que el jugador tiene creado un equipo y que en ese equipo hay un jugador con ese nombre, a continuación lo elimina. Importante: no se reintegra el valor del jugador al dinero del equipo.

Resultado: False si ocurre alguna de las siguientes cosas: el jugador no tenga creado un equipo, el equipo del jugador no tiene jugadores o no tiene ningún jugador con ese nombre, o no se consiguió eliminar al jugador. En caso contrario devuelve true.

```
1    PostPartidoTerminado(int identificador)
```

Parámetros: identificador es el número identificador de sesión del usuario.

Acción: Guarda los datos del equipo y jugadores del usuario. Elimina el objeto partido cuando ambos jugadores han finalizado el postpartido.

Salir

```
1    Bool CerrarSesion(int identificador)
```

Parámetros: identificador es el número identificador de sesión del usuario.

Acción: Se encarga de guardar los cambios en los jugadores, y el equipo del jugador, lo extrae de la lista de usuarios conectados y termina la sesión.

Resultado: Devuelve false si no se tiene constancia de que el usuario está conectado, y true si se guardaron los datos del jugador y se le deslogueó con éxito.

4.5.4 Servicios parte Cliente

En la parte cliente se ha desarrollado una clase remota que extiende de MarshalByRefObject, y que tiene todos sus métodos abstractos, posteriormente se creó la clase ImplementaciónCliente que hereda de ICliente e implementa todos sus métodos.

A continuación se muestra una descripción de los servicios de la parte cliente.

4.5.4.1 Servicios para los usuarios logueados

Invitar a jugar partido

```
1    InvitacionAJugarPartido(string usuario, string mensaje)
```

Parámetros: usuario es el nick del usuario que realiza la invitación a jugar un partido, mensaje es el mensaje de la invitación a mostrar al usuario.

Acción: Se abre un formulario de forma modal donde se muestra el mensaje de invitación y se pide que acepte o rechace la invitación.

```
1    RespuestaInvitacionAJugarPartido(bool respuesta)
```

Parámetros: respuesta es la respuesta a la invitación que efectuó con anterioridad el usuario.

Acción: La respuesta a la invitación se muestra en la ventana de invitación a jugar partido que el usuario tiene abierta, y se activa el botón que permite cerrar dicha ventana.

```
1    CancelarInvitacionAJugarPartido()
```

Acción: El usuario ha sido invitado a jugar un partido y todavía no ha aceptado o rechazado la invitación, este método se encarga de cerrar la ventana que solicita al usuario que acepte o decline la invitación a jugar el partido.

Jugar partido

```
1    MoverJugador(string equipo, string jugador, string casillas)
```

Parámetros: equipo es el nombre del equipo del jugador que se desea mover; jugador es el nombre del jugador que se desea mover; y casillas es el itinerario por el que se mueve el jugador.

Acción: Las casillas por las que tiene que moverse el jugador están en la forma #1ªCasilla #2ªCasilla #3ªCasilla... #NªCasilla. Se insertan las casillas del itinerario en una lista, se obtiene la instancia del jugador que hay que mover y se ordena la animación. Si el número de casillas que se tiene que mover es mayor de dos entonces el jugador correrá, si no andará.

```
1    ResultadoHeridaJugador(string equipo, string jugador, int
         resultado, string equipoCausante, string jugadorCausante)
```

Parámetros: equipo es el nombre del equipo del jugador que ha recibido la herida; jugador es el nombre del jugador herido; resultado es un valor entre 0 y 5 que indica el tipo de herida; equipoCausante es el nombre del equipo cuyo jugador ha causado la herida y jugadorCausante es el nombre del jugador que ha causado la herida.

Acción: Si resultado es 0 entonces el jugador ha sido derribado, si es 1 es que ha sido aturdido; cuando resultado es 2 significa que el jugador ha quedado inconsciente, cuando es 3 significa que ha sido herido leve y no podrá seguir jugando, cuando es

4 es que ha sido herido grave y no podrá seguir jugando, y cuando es 5 significa que el jugador ha muerto. Si el jugador ha quedado inconsciente, herido leve, herido grave o muerto se le retira inmediatamente del campo. Para cualquier resultado se informa al usuario por pantalla de lo que ha ocurrido.

```
1    esTurno()
```

Acción: Se informa al usuario por pantalla de que es su turno y se habilitan los controles necesarios para que pueda efectuar las acciones permitidas a los jugadores durante el turno del equipo.

```
1    turnoContrario()
```

Acción: Se informa al usuario por pantalla de que es su turno y se habilitan los controles necesarios para que no pueda efectuar las acciones permitidas a los jugadores durante el turno del equipo.

```
1    PatadaInicial(string equipo, string jugador, string casillas)
```

Parámetros: equipo es el nombre del equipo del jugador que va a realizar la patada inicial; jugador es el nombre de dicho jugador y casillas es una secuencia de casillas por las que rebotará el balón como resultado de la patada.

Acción: Las casillas por las que tiene que rebotar el balón en la patada inicial están en la forma #1ªCasilla #2ªCasilla #3ªCasilla... #NªCasilla. El balón rebotará por dichas casillas siguiendo el orden expresado en la secuencia.

```
1    ResultadoPlacaje(string equipoAtacante, string jugadorAtacante,
         string equipoDefensor, string jugadorDefensor, int casilla,
         bool movimientoEmpuje, int resultadoPlacaje)
```

Parámetros: equipoAtacante es el nombre del equipo cuyo jugador va a realizar el placaje; jugadorAtacante es el nombre del jugador que va a efectuar el placaje; equipoDefensor es el nombre del equipo cuyo jugador es el beneficiario del placaje; jugador defensor es el nombre del jugador que se va a intentar placar; casilla indica

la posible casilla a la que se empujaría al jugador defensor; movimiento empuje indica si se el jugador defensor es empujado a la casilla; resultadoPlacaje es un valor de 0 a 5 que indica el resultado.

Acción: Si movimiento empuje es cierto, se ordena una animación de placaje de un jugador donde el jugador defensor es empujado a la casilla indicada, sino se ordena una animación normal de placaje. Actualmente el parámetro resultadoPlacaje no se utiliza.

```
1   String ElegirJugadorLista(string mensaje, string lista)
```

Parámetros: mensaje es un texto informativo destinado al usuario y lista es una secuencia de jugadores en formato #nombre

Acción: Se muestra al usuario una ventana que contiene un textbox y un combobox, en el textbox se muestra el mensaje pasado por parámetro y en el combo se han cargado los nombres indicados en la lista.

Resultado: Devuelve la cadena vacía si el usuario no ha elegido ningún nombre o si pulsó cancelar; en caso contrario devuelve el nombre del jugador seleccionado en el momento en que pulsó hecho.

```
1   ResultadoPase(string equipoLanzador, string jugadorLanzador,
        string jugadorReceptor, string equipoInterceptor, string
        jugadorInterceptor, string casillasRebote, int resultadoPase)
```

Parámetros: equipoLanzador es el nombre del equipo que posee el balón; jugador-Lanzador es el nombre del jugador que posee el balón; jugadorReceptor es el nombre del jugador al que se destina el pase; equipoInterceptor es el nombre del equipo que no posee el balón; jugadorInterceptor es el nombre del jugador que va a intentar interceptar el pase; casillasRebotes es una secuencia de casillas y resultadoPase un entero que indica el resultado del pase.

Acción: No resulta estrictamente necesario indicar el equipo interceptor ni el jugador interceptor excepto en el caso que se haya interceptado el pase, en el caso de

especificarse en las animaciones se animará también el jugador interceptor. CasillasRebote es una sucesión de casillas separadas por '#' que indican el orden de casillas por las que el balón debe moverse en caso de rebotar. Si el resultado es 0 entonces significa que el pase se ha completado satisfactoriamente y se ordena realizar la animación del pase; si el resultado es 1 entonces es que ha fallado el lanzamiento y el balón se le escapa al jugador lanzador rebotando en el suelo; si el resultado del pase es 2 entonces el jugador interceptor ha capturado el balón durante el pase; y si el resultado del pase es 3 entonces significa que el jugador receptor ha fallado la recepción del balón y éste rebota por el suelo.

```
1    ResultadoRecogerBalon(string equipo, string jugador, string
         casillas, int resultado)
```

Parámetros: equipo es el nombre del equipo del jugador que va a recoger el balón; jugador es el nombre del jugador; casillas es una secuencia de casillas y resultado es un entero entre 0 y 1 (ambos inclusive).

Acción: Casillas es una sucesión de números separados por '#' que indican las casillas por las que debe moverse el balón en caso de fallar la recogida del balón. Si el resultado es 0 significa que se ha recogido el balón satisfactoriamente y el jugador pasa a poseer el balón; si es 1 significa que se ha fallado al recoger el balón y que éste rebota por el suelo.

```
1    JugadoresPartido(string miEquipo, string misJugadores, int
         misSegundasOportunidades, string equipoContrario, string
         jugadoresAdversarios, int segundasOportunidadesAdversario,
         bool invitado)
```

Parámetros: miEquipo es el nombre de equipo del usuario; misJugadores es una secuencia de los nombres de los jugadores del usuario separados por '#', misSegundasOportunidades es el número de segundas oportunidades que poseo, equipoContrario es el nombre del equipo de mi adversario, jugadores adversarios es una secuencia de los nombres de los jugadores adversarios separados por '#', segundasOportunidadesAdversario es el número de segundas oportunidades de las que dispone el equipo adversario e invitado indica si el usuario fue invitado a jugar el partido o si fue él quien invitó.

Acción: Se cargan los datos de los nombres de los equipos, sus jugadores y sus segundas oportunidades. El parámetro invitado sirve para determinar el color de los jugadores.

```
1    EquipoLanzador()
```

Acción: Indica al usuario que su equipo es el equipo lanzador.

```
1    EquipoReceptor()
```

Acción: Indica al usuario que su equipo es el equipo receptor.

```
1    Coloca(string equipo, string posicionesJugadores)
```

Parámetros: equipo es el nombre del equipo que coloca sus jugadores, posiciones jugadores son una secuencia de los nombres de los jugadores junto con la casilla que ocupan.

Acción: Cada elemento de posiciones jugadores tiene el formato @NombreJugador #NºCasilla. Se cargan los jugadores especificados en la casilla especificada.

```
1  AnotasTouchDown()
```

Acción: Se actualiza el marcador. Se quitan todos los jugadores y el balón del terreno de juego.

```
1    EncajasTouchDown()
```

Acción: Se actualiza el marcador. Se quitan todos los jugadores y el balón del terreno de juego.

```
1    MediaParte()
```

Acción: Se actualiza el indicador de la parte en el marcador. Se quitan todos los jugadores y el balón del terreno de juego.

```
1    FinPartido()
```

Acción: Se eliminan todos los controles y elementos relativos al partido, y se empiezan a mostrar las ventanas de postpartido.

```
1    FinalizaAccion()
```

Acción: Indica al usuario que ha terminado la acción que previamente solicitó realizar. Se activa el control de elegir acción y se activa la selección de jugadores del equipo.

```
1    UsaSegundaOportunidad(string equipo)
```

Parámetros: equipo es el nombre del equipo que usa la segunda oportunidad.

Acción: Se actualiza en el marcador el respectivo valor de las segundas oportunidades del equipo indicado.

```
1    Bool MensajeConfirmacion(string mensaje)
```

Parámetros: mensaje es un texto destinado al usuario.

Acción: Se muestra al usuario una ventana con el mensaje pasado por parámetro y con dos botones Si y No.

Resultado: True si el usuario pulsó el botón Si, false si el usuario pulsó el botón No.

```
1    Int ElegirResultadoPlacaje(int opcion1, int opcion2, int opcion3)
```

Parámetros: opcio1, opcion2 y opcion3 son valores entre -1 y 5.

Acción: Se muestra una ventana con 3 botones, uno para representar cada opción. Si el valor de la opción es -1 entonces se oculta el botón que lo representa, si no entonces se muestra el texto correspondiente con el valor de la opción: para el valor 0 "Atacante derribado", para el valor 1 "Defensor derribado", para el valor 2 "Ambos derribados", para el 3 "Defensor empujado", para el 4 "Defensor empujado y derribado" y para el 5 "Ninguno derribado".

Resultado: Se devuelve el valor de la opción elegida por el usuario.

```
1        RebotеBalon(int casillaBalon, string rebote, string equipo, string
             jugador)
```

Parámetros: casillaBalon es la casilla donde actualmente se encuentra el balón, rebote es una secuencia de casillas por donde debe rebotar el balón, equipo es el nombre del equipo que posee el balón, jugador es el nombre del jugador que posee el balón.

Acción: En caso de que se indique el equipo y el jugador que poseen el balón, se ordenará una animación en la cual el jugador pierde el balón y este rebota por las casillas; sino se ordenará una animación en la que el balón rebota en el suelo por las casillas.

```
1        GirarJugador(string nombreEquipo, string nombreJugador)
```

Parámetros: nombreEquipo es el nombre del equipo del jugador que se va a girar, nombreJugador es el nombre del jugador que se va a girar.

Acción: Se rota al jugador 180 grados, de forma que anteriormente estaba boca arriba en el campo y a continuación pasa a estar boca abajo.

```
1        EnPieJugador(string nombreEquipo, string nombreJugador)
```

Parámetros: nombreEquipo es el nombre del equipo cuyo jugador va a ponerse en pie, nombreJugador es el nombre del jugador.

Acción: Se ordena la animación de poner en pie al jugador indicado.

4.5.5 Diagramas de interacción de los casos de uso

En esta sección se utilizarán diagramas de interacción para ilustrar la relación entre las comunicaciones y los casos de uso. Los valores que se han elegido para emplear como parámetros y como respuestas se han elegido con idea de facilitar la comprensión de los diagramas de interacción, por lo cual dichos valores son fieles a la esencia de la interacción pero puede que no sean valores que realmente pudieran darse en el sistema.

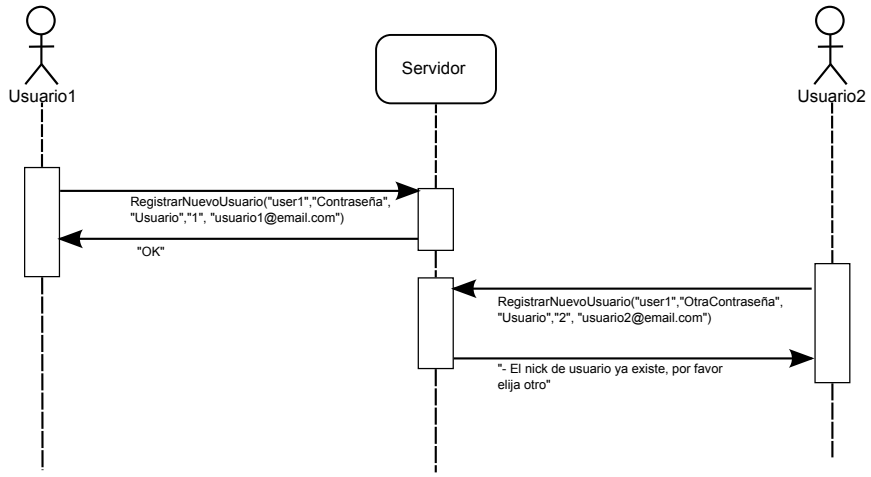


Figura 4.35: Diagrama de interacción del registro

4.5.5.1 Diagramas de interacción en el caso que el usuario esté sin loguear

Registrarse En el diagrama de interacción 4.35 puede verse como el Usuario1 crea correctamente su cuenta, a continuación el Usuario2 intenta crear también una cuenta pero falla al existir el nick "user1" ya en el sistema.

Solicitar nueva contraseña

El diagrama de interacción 4.36 es un ejemplo de éxito en la solicitud de nueva contraseña.

Loguearse

En el diagrama de interacción de la figura 4.37 el Ususario1 solicita en primer lugar la autenticación para posteriormente loguearse satisfactoriamente; a continuación Usuario2 solicita la autenticación pero el proceso de autenticación falla al estar mal calculado el MD5.

4.5.5.2 Diagramas de interacción en el caso que el usuario esté logueado

Ver mensajes chat y escribir mensaje chat

En el diagrama de interacción de la figura 4.38 el Usuario1 envía un mensaje de chat, obteniendo como respuesta los últimos mensajes de chat (en este caso lo que

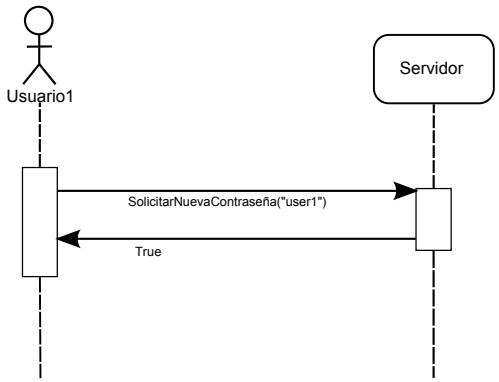


Figura 4.36: Diagrama de interacción de una solicitud de contraseña

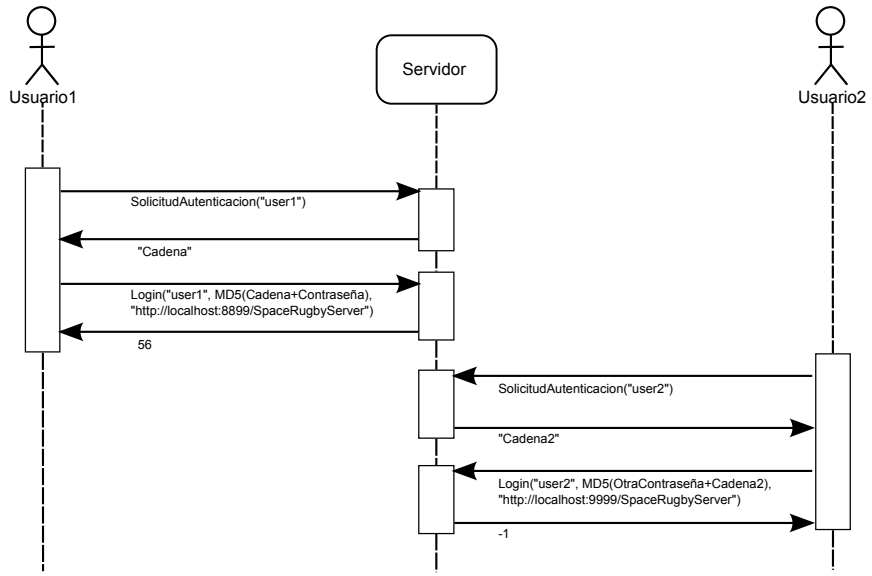


Figura 4.37: Diagrama de interacción de un proceso de logueado

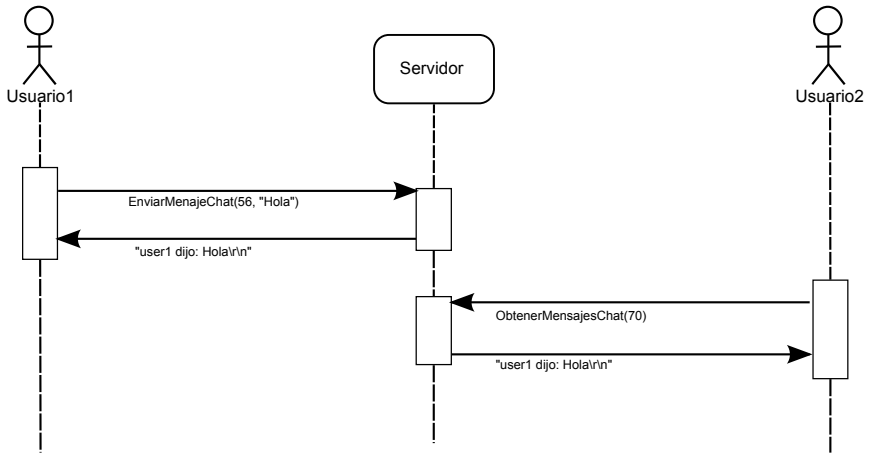


Figura 4.38: Diagrama de interacción mensajes chat

él ha escrito); después el Usuario2 solicita los últimos mensajes de chat al servidor, obteniendo el mensaje de User1.

Ver usuarios conectados

El diagrama de la figura 4.39 muestra como el Usuario1 solicita la lista de los usuarios conectados, y como respuesta recibe que solo están conectados user1 y user2.

Ver datos y cambiar datos personales

En este diagrama de interacción de la figura 4.40 puede verse como en primer lugar Usuario1 solicita sus datos personales, a continuación los cambia, y por último al solicitarlos de nuevo ve como son distintos a cuando los solicitó al principio. A continuación el Usuario2 obtiene sus datos personales e intenta cambiarlos, pero no se efectúa el cambio porque la antigua contraseña que especifica es incorrecta.

Crear nuevo equipo

En el diagrama de interacción de la figura 4.41 es un ejemplo completo de creación de un nuevo equipo. Primero se solicita la lista de las razas disponible, a continuación se crea el nuevo equipo; una vez existe nuestro nuevo equipo, podemos comprar los jugadores, en el ejemplo se compra un jugador "J1" y a continuación se intenta comprar un jugador con el mismo nombre, lo cual hace que el servidor devuelva false y no se efectúe la compra; también se vende un jugador de manera satisfactoria; podemos ver también varias transacciones de compra y venta de segundas oportunidades y fac-

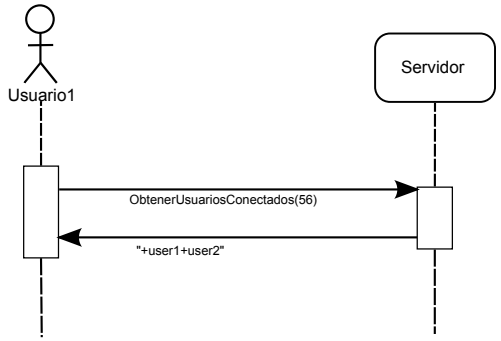


Figura 4.39: Diagrama de interacción usuarios conectados

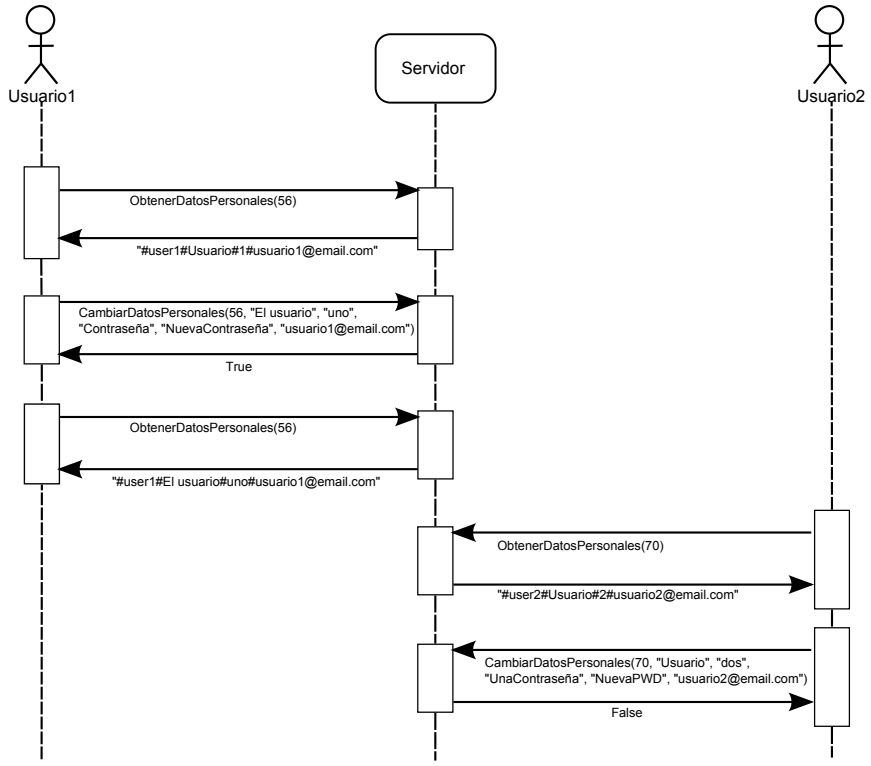


Figura 4.40: Diagrama de interacción datos personales

tor de hinchas, y cuando intentamos vender más segundas oportunidades de las que disponemos entonces vemos que el servidor devuelve false y no efectúa la venta. Por último la creación de equipo se ratifica y finaliza cuando Usuario1 envía EquipoTerminado.

Ver estadísticas equipo

En el diagrama 4.42, el Usuario1 solicita los datos de su equipo, que como podemos ver es un equipo de humanos llamado "Guerreros usuario 1", con dos jugadores: un línea llamado "J1" y un blitzer llamado "J2".

Ver ranking

El el diagrama de interacción de la figura 4.43 muestra como el Usuario1 solicita el ranking de su equipo.

Buscar equipo

En el diagrama de la figura 4.44, el Usuario1 busca un equipo cuyo entrenador sea "user1" (él mismo) y obtiene el resultado.

Invitar usuario a jugar partido, aceptar invitación a jugar partido y rechazar invitación a jugar partido

El diagrama de la figura 4.45 contiene dos ejemplos, un primer ejemplo en el que el Usuario1 invita al Usuario2 a un partido amistoso y éste acepta, y un segundo ejemplo en el cual el Usuario1 invita al Usuario2 a un partido amistoso y ante la falta de contestación del Usuario2 el servidor expira la validez de la invitación.

Jugar Partido

Los detalles sobre la interacción entre el servidor y los clientes es compleja y se detallará en la siguiente sección.

Salir

El diagrama de la figura 4.46 muestra como el Usuario1 solicita desconectarse y lo consigue con éxito.

4.5.6 Diagramas de interacción de las acciones del partido

A continuación se muestra como es la comunicación en las acciones del partido.

Colocar jugadores

En el diagrama de la figura 4.47 puede verse como el Usuario1 envía al Servidor las posiciones de sus jugadores, J1 está en la casilla 59, J2 en la 68 y J3 en la 44; dado que las posiciones son correctas el Servidor pide al Usuario2 que coloque los jugadores en

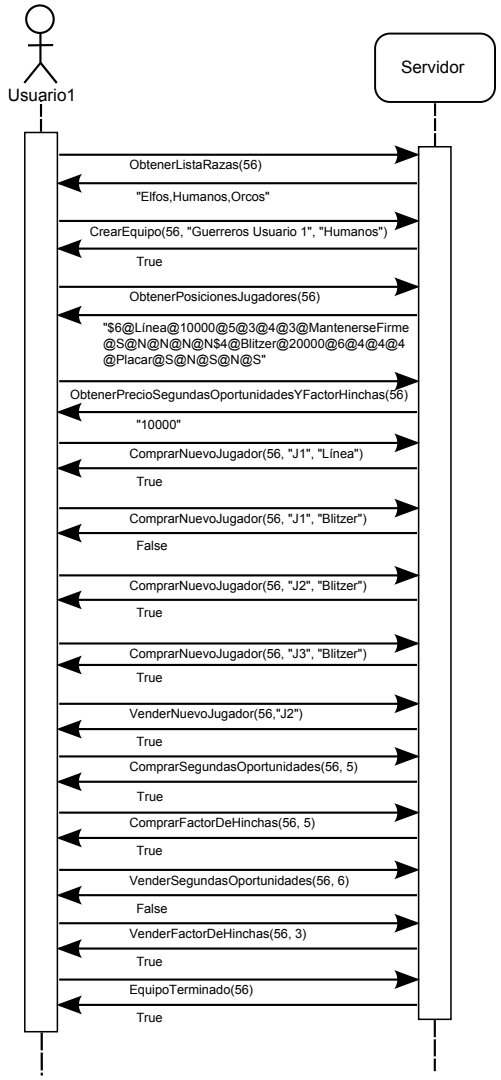


Figura 4.41: Diagrama de interacción de creación de un nuevo equipo

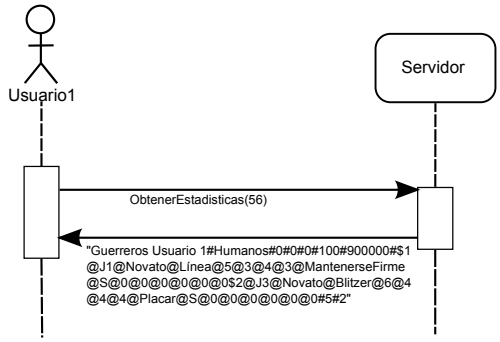


Figura 4.42: Diagrama de interacción de estadísticas de equipo

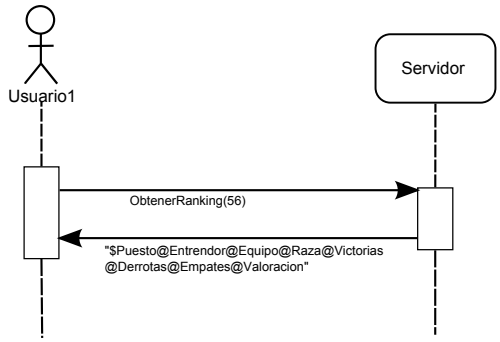


Figura 4.43: Diagrama de interacción ranking equipo

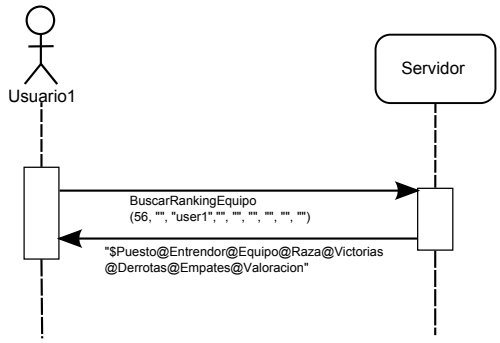


Figura 4.44: Diagrama de interacción búsqueda de equipo

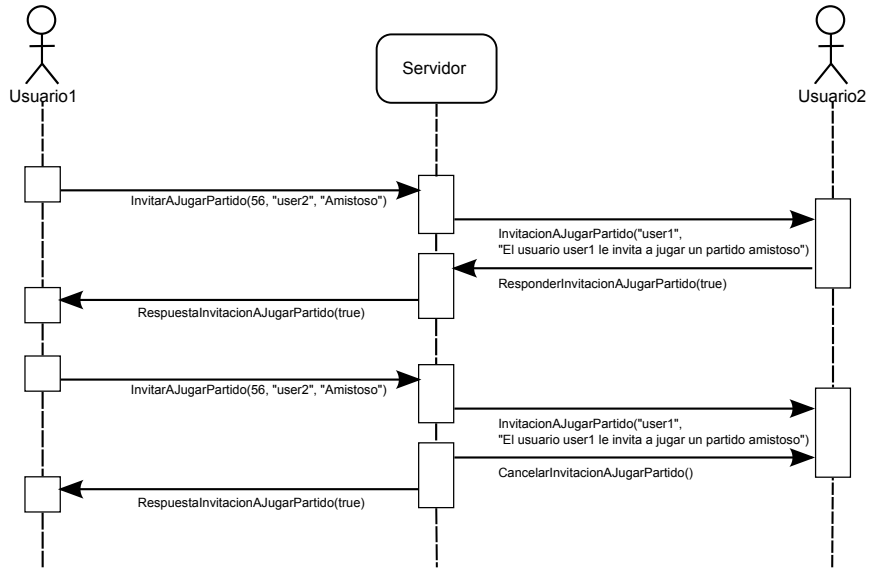


Figura 4.45: Diagrama de interacción de una invitación a jugar una partido

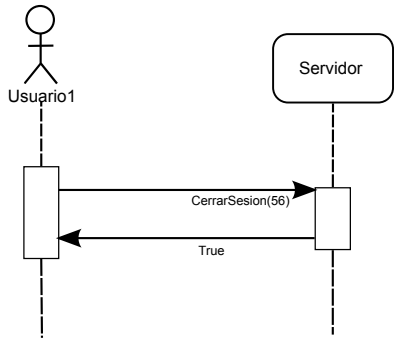


Figura 4.46: Diagrama de interacción de un proceso de logueado

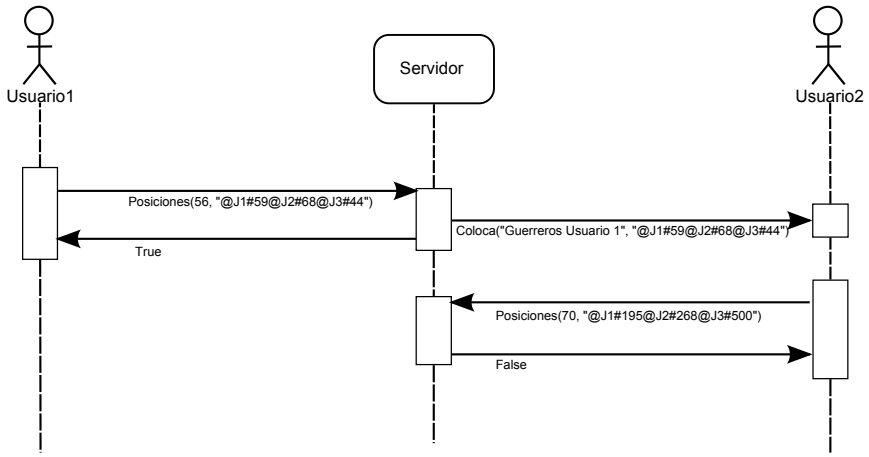


Figura 4.47: Diagrama de interacción colocar jugadores

las posiciones y devuelve al Usuario1 el valor verdadero. A continuación el Usuario2 envía al Servidor las posiciones de sus jugadores, J1 en la casilla 195, J2 en la 268 y J3 en la 500; dado que no existe ninguna casilla 500 el Servidor devuelve falso al Usuario2.

Patada inicial

En el diagrama 4.48 el Usuario1 envía al servidor la casilla del campo contrario hacia la cual se va a patear el balón, en este caso la casilla 233, y el nombre del jugador que realizará la patada, que es J1. A continuación el Servidor envía a ambos usuarios el resultado de la patada: el nombre del equipo que realiza la patada, el nombre del jugador que patea, y la sucesión de casillas por las que rebota el balón, por lo cual el balón cae en la casilla 233 y rebota por la 248, 262 y por último llega a la 246. Una vez informados los jugadores del resultado de la patada, el Servidor le indica al Usuario1 que es su turno, y al Usuario2 que es el turno del equipo contrario.

Recoger balón

En el diagrama 4.49 puede verse como el Usuario1 solicita al servidor que resuelva el movimiento de recogida de balón por parte del jugador J1. El Servidor comunica a ambos usuarios que el jugador J1 del equipo Guerreros Usuario 1 ha recogido el balón (resultado 0). A continuación vemos un ejemplo de como el Usuario2 pide al servidor que resuelva el movimiento de recogida del balón por parte del jugador J2, el resultado en este caso es 1, lo cual significa que el jugador J1 del Equipo Usuario 2 ha fallado al

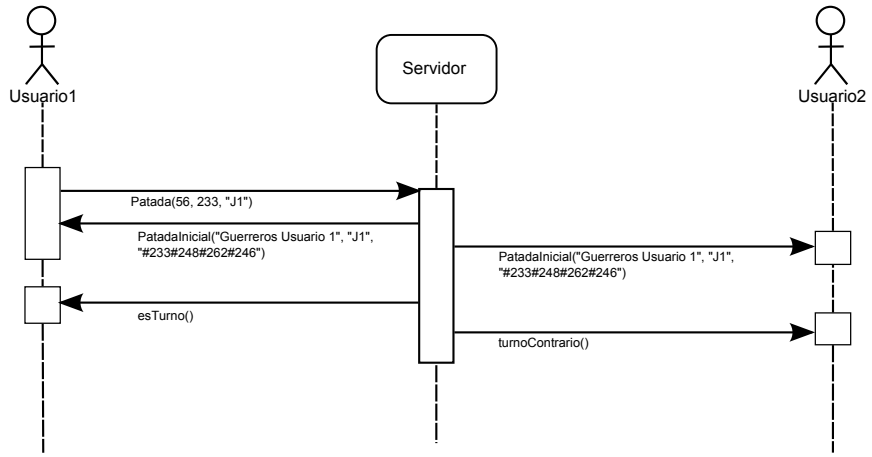


Figura 4.48: Diagrama de interacción patada inicial

intentar recoger el balón, como consecuencia se produce un cambio de turno, que pasa a poseer el Usuario1.

Placar jugador

Puede apreciarse en el diagrama 4.50 como se envía la petición de que el jugador J1 del equipo del Usuario1 plaque al jugador J2 del equipo del Usuario2, el resultado (3) es que el jugador J2 es empujado a la casilla 200, y se comunica a ambos usuario, una vez comunicada el resultado del placaje el Servidor comunica al Usuario1 que ha finalizado la acción y por lo tanto puede solicitar otra. Los resultados de un placaje pueden ser: Atacante derribado (0), defensor derribado (1), defensor y atacantes derribados (2), defensor empujado (3), defensor empujado y derribado (4) y ninguno derribado (5). A continuación se muestra un ejemplo más complejo de placaje en el cual el Usuario1 envía la petición de que el jugador J1 de su equipo plaque al jugador J3 del equipo del Usuario2, se producen tres resultados entre los que el Usuario1 tiene que elegir (0, 2 y 5), el usuario elige el 2 por el cual ambos acabarían derribados, entonces el servidor le pregunta al Usuario1 si quiere utilizar una ficha de segunda oportunidad, a lo que el Usuario1 contesta que si, se vuelve a resolver el placaje y ahora el Usuario1 debe elegir entre los resultados 0, 1 y 3, y elige 1; el resultado se comunica a ambos usuarios y se le indica al Usuario1 que la acción de placaje ha terminado.

Pasar balón

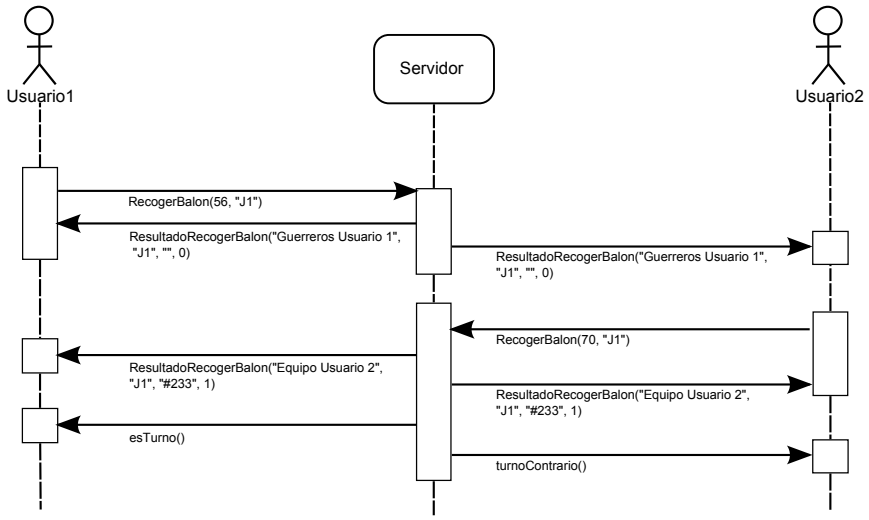


Figura 4.49: Diagrama de interacción de un proceso de logueado

En el diagrama 4.51 se ilustran dos ejemplos de un pase de balón. Los resultados de un pase pueden ser: pase completo (0), fallo en el lanzamiento (1), pase interceptado (2) y fallo en la recepción (3).

En el primer pase se produce una intercepción del pase. El Usuario1 solicita que el jugador J1 pase el balón, a continuación el servidor le pregunta a cual jugador (J2 o J3) quiere destinar e pase, y el Usuario1 elige a J2, se le pide al Usuario1 una confirmación para realizar el pase; a continuación (y dado que existen jugadores en la trayectoria de intercepción del pase) se solicita al Usuario2 que elija con qué jugador intentará interceptar el pase, y responde que lo intentará con J3; por último se comunica el resultado del pase a ambos usuarios, y dado que se ha producido una intercepción, se produce un cambio de turno.

En el segundo pase el Usuario2 solicita que el jugador J3 pase el balón, como en la anterior ocasión se pregunta al Usuario2 por el jugador que recibirá el pase, en este caso J2, y una confirmación sobre si realmente quiere realizarlo; a continuación se le pregunta al Usuario1 que elija el jugador con que intentará interceptar el pase, en este caso J3; por último se resuelve el pase, cuyo resultado es que falla el lanzamiento (1), y se produce un cambio de turno.

Mover jugador

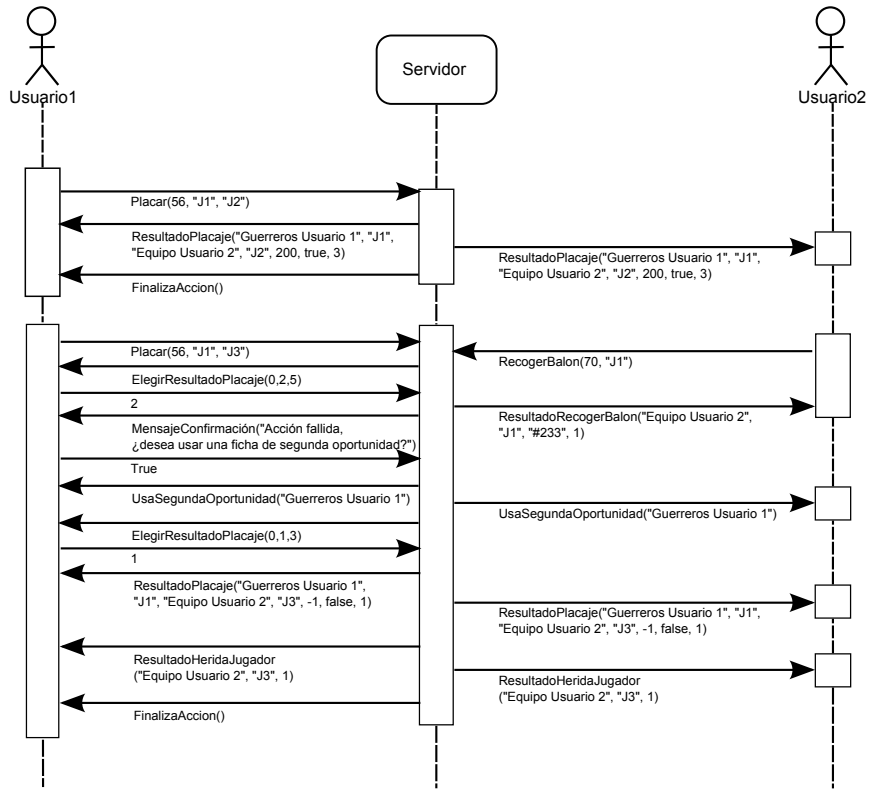


Figura 4.50: Diagrama de interacción placar jugador

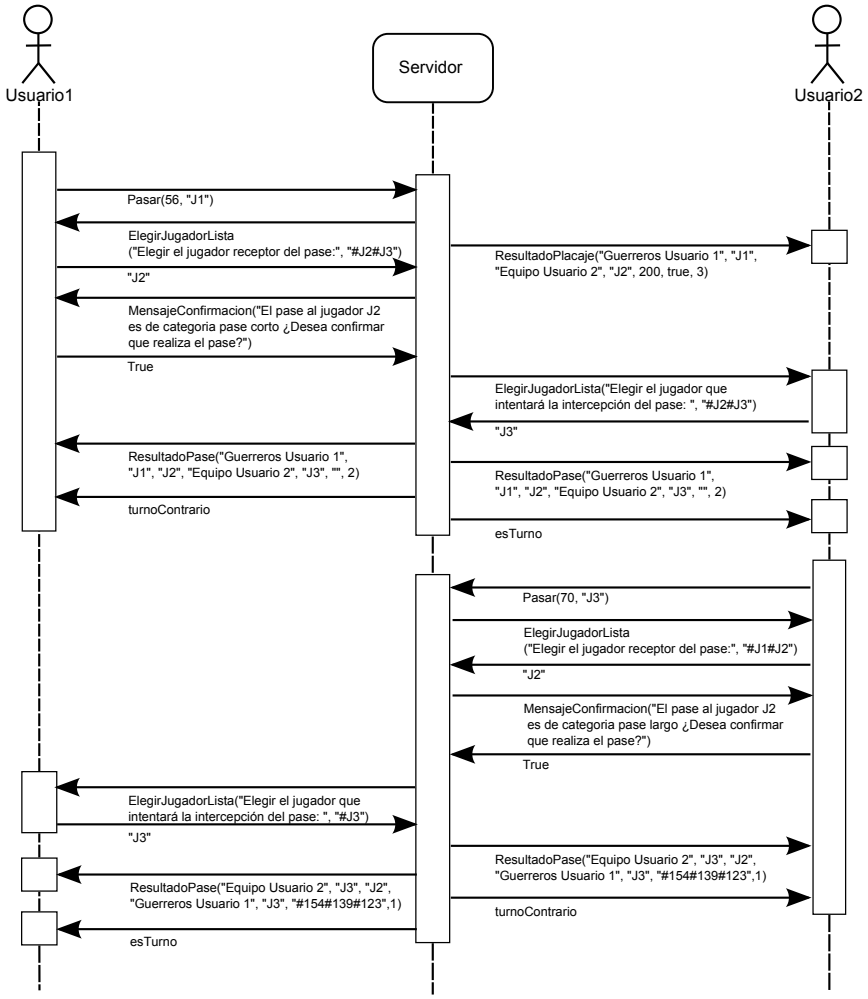


Figura 4.51: Diagrama de interacción pasar balón

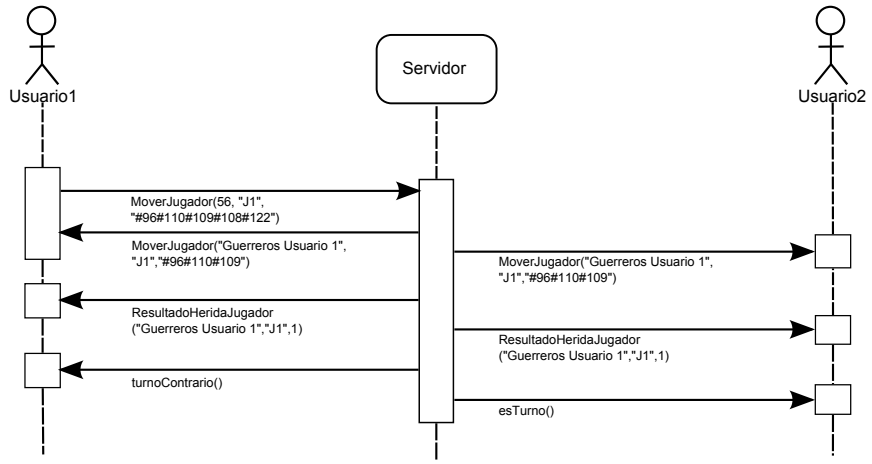


Figura 4.52: Diagrama de interacción mover jugador

En el diagrama 4.52 se muestra un ejemplo de movimiento de jugador. El Usuario1 solicita mover el jugador J1 por las casillas 96, 110, 109, 108, 122 en dicho orden. El servidor comunica a ambos usuarios que el jugador J1 se mueve por las casillas 96, 110 y 109; pero al llegar a dicha casilla el jugador falla al intentar moverse y se produce una herida que se comunica a continuación. Los tipos de heridas que puede sufrir un jugador son: derribado (0), aturdido (1), inconsciente (2), herido leve (3), herido grave (4) y muerto (5).

Elegir acción, girar y poner en pie

Se tiene un ejemplo de solicitud de las acciones permitidas a un jugador en el diagrama 4.53. El Usuario1 solicita la lista de acciones que puede realizar el jugador J1 de su equipo, el servidor contesta que el jugador J1 puede Mover, Placar y Pasar. A continuación el Usuario1 pregunta por las acciones del jugador J2, que se encuentra aturdido, y cuya acción posible es Girar; el Usuario1 solicita al servidor que gire al usuario y el Servidor comunica a ambos usuarios que se gire al jugador J2 y posteriormente indica al Usuario1 que ha finalizado la acción, por lo cual el Usuario1 puede solicitar de nuevo las acciones de J2 que ahora solo tiene permitido ponerse en pie; el Usuario1 solicita poner en pie al jugador J2, y el servidor comunica a ambos usuarios que se ponga de pie dicho jugador, seguidamente comunica al Usuario1 que ha finalizado la acción.

Finalizar Turno

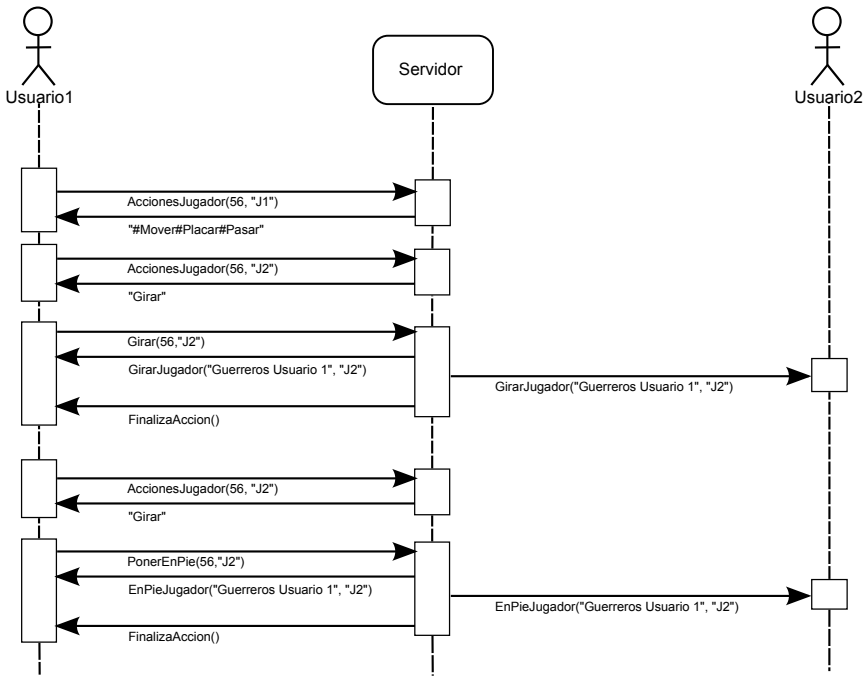


Figura 4.53: Diagrama de interacción elegir acción

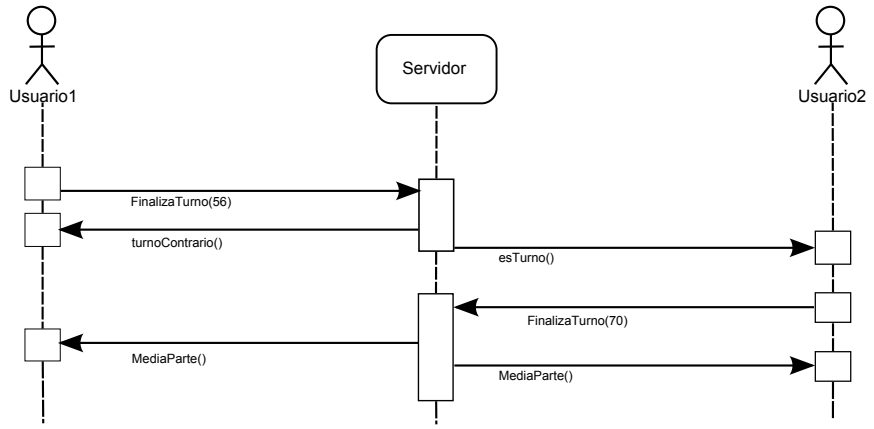


Figura 4.54: Diagrama de interacción finalizar turno

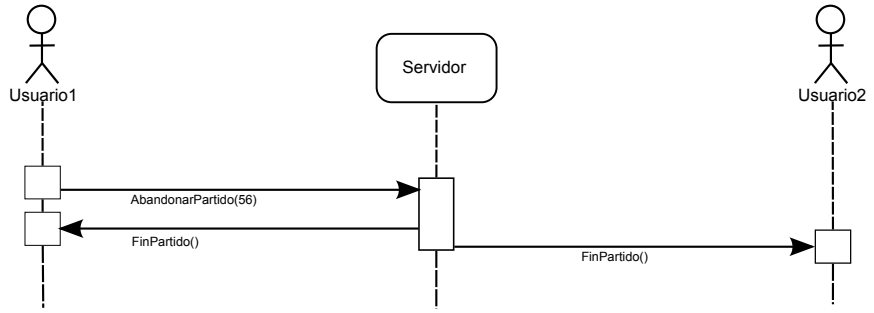


Figura 4.55: Diagrama de interacción abandonar partido

En el diagrama 4.54 se encuentra un ejemplo de como se efectúa un cambio de turno, el Usuario1 indica al servidor que quiere terminar su turno, a continuación el servidor indica al Usuario1 que es el turno contrario y al Usuario2 que es su turno. Posteriormente el Usuario2 indica al servidor que quiere terminar su turno, y en este caso al finalizar el Usuario2 su turno se ha llegado a la mitad del partido, situación que el servidor se encarga de comunicar a ambos usuarios.

Abandonar partido

En el diagrama 4.55 muestra un ejemplo de petición de finalización del partido por parte del Usuario, en él el Usuario1 solicita al servidor abandonar el partido, el servidor comunica la finalización del partido a ambos usuarios.

4.6 Diseño de los elementos 3D

Este capítulo se encarga de describir todo el proceso de creación de los elementos 3D del juego, mostrando todos los aspectos de su desarrollo y las peculiaridades o problemas que se han encontrado.

Sin embargo, este capítulo no pretende ser una guía de modelado 3D, lo cual precisaría de varios capítulos para tan solo llevar a cabo una explicación sencilla de lo que es el modelado 3D usando Blender.

4.6.1 Herramientas de modelado

Blender es una herramienta creada por Stichting Blender Foundation y puede descargarse de www.blender.org. Blender se distribuye bajo licencia GPL, que significa que puede obtenerse de forma gratuita y utilizarse tanto con propósito comercial como no comercial.

La versión de Blender empleada para el desarrollo del proyecto ha sido la 2.47. En el descargable, aparte del código del programa vienen una serie de scripts, entre los cuales se incluyen los scripts para importar y exportar otros formatos de objetos 3D. Junto con el programa, vienen incluidos algunos módulos de Python, ya que los scripts de Blender están escritos en ese lenguaje; es por esta razón que algunos scripts para Blender precisen módulos de Python que no vienen incluidos con el código del programa, motivo por el cual se recomienda que antes de instalar Blender se tenga instalada una distribución completa de Python. Python puede obtenerse de forma gratuita en www.python.org. La versión de Python que he utilizado para el proyecto ha sido la 2.52.

Al arrancar Blender, aparecen dos ventanas. Una tiene la forma de una ventana de símbolo de sistema y se trata la ventana de salida Python de los scripts de Blender. En dicha ventana es donde se muestran los resultados de las operaciones que Blender realizan con Python.

La otra ventana es la ventana de la aplicación. A primera vista resulta compleja por la cantidad de botones y opciones que aparecen, esto es así para optimizar la velocidad en el trabajo diario; este es el motivo que hace que Blender resulte complejo para los novatos. En la imagen 4.56 puede apreciarse como es la ventana de la aplicación.

A continuación se describirá la pantalla y sus elementos más importantes:

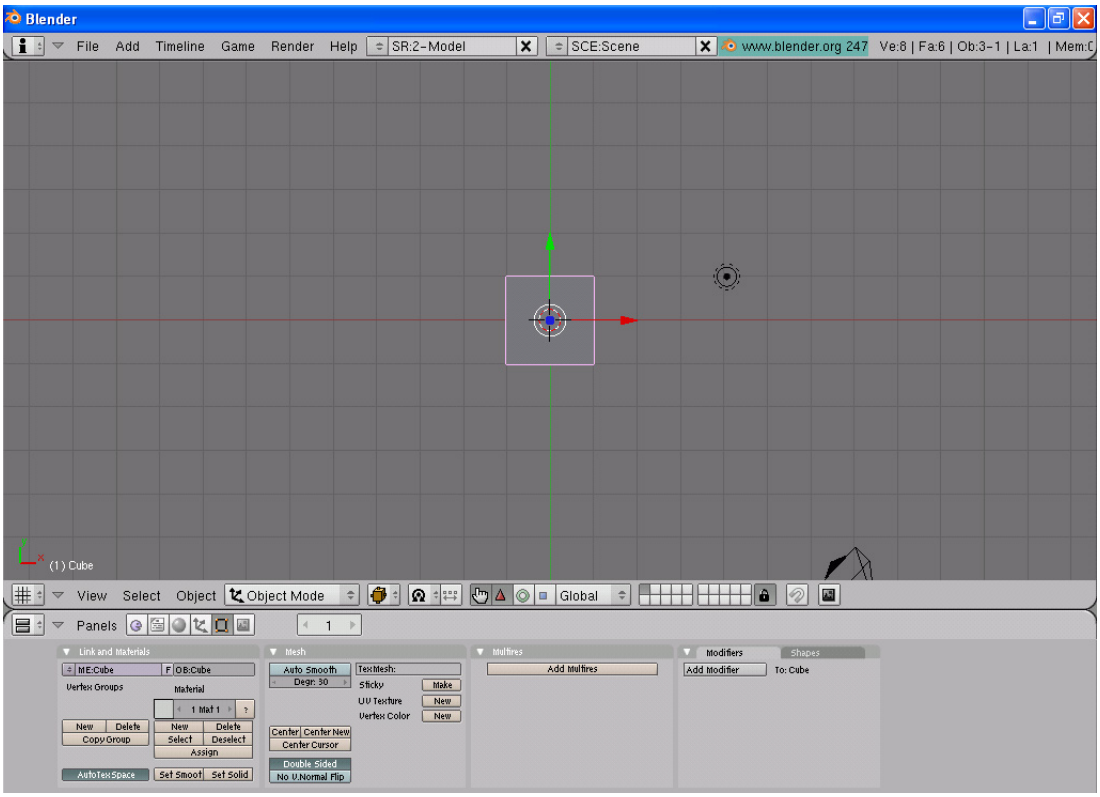


Figura 4.56: Ventana principal de Blender

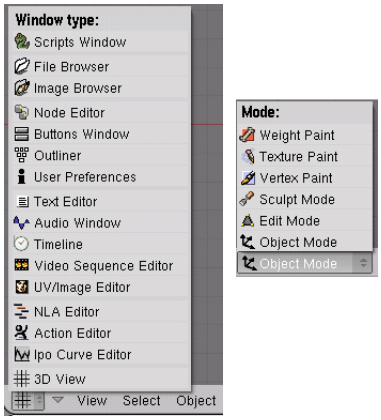


Figura 4.57: a)Desplegable con los tipos de ventana b) Desplegable con los tipo de operación

En la parte superior de la pantalla se encuentran los menús del programa, seguidos de dos combos: uno para crear nuevos elementos de la escena, y otro para crear nuevas escena. De estos elementos cabe destacar el menú File, desde el cual podemos crear, abrir, guardar, importar y exportar modelos.

Debajo de los menús está la ventana de trabajo. Esta ventana tiene la peculiaridad que puede dividirse tanto horizontal como verticalmente en varias ventanas de trabajo, lo cual permite tener varias visualizaciones o ventanas de distintos tipos a la vez en pantalla. En la parte inferior de la ventana de trabajo se encuentran los botones y menús del tipo de ventana de trabajo, los tipos de ventana de trabajo son: 3D view, Ipo Curve Editor, Action Editor, NLA Editor, UV/Image Editor, Video Secuence Editor, Timeline, Audio Window, etc. Las ventanas en principio más importantes son las de 3D view, la cual muestra el modelo 3D para su edición; Action Editor, que permite ver y editar los keyframes de las animaciones; y Script Window, que permite la ejecución de scripts de Python.

Se puede seleccionar el tipo de ventana de trabajo pulsando sobre el botón desplegable del extremo izquierdo del menú de la ventana de trabajo, como se muestra a continuación en la figura 4.57 a).

Del menú de la ventana 3D view interesa resaltar el desplegable situado al lado de los menús View Select y Object. Dicho desplegable permite elegir el modo de operación con el objeto 3D tal como se muestra en la figura 4.57 b).

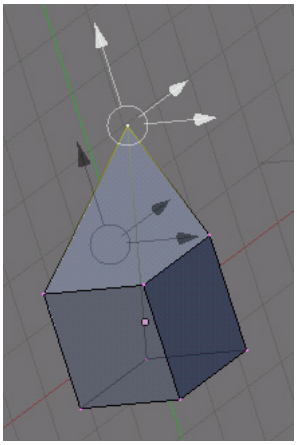


Figura 4.58: Ejemplo de edición de un objeto 3D

Si se elige trabajar en "Object Mode", se trabajará con los objetos 3D como un todo, mientras que si se elige el modo "Edit Mode" se podrán manipular los vértices, aristas y caras del polígono que forma el objeto. En la figura 4.58 se muestra un ejemplo en el que estamos desplazando un vértice del objeto en modo edición y recordar que para seleccionar un elemento de un objeto debe pincharse sobre dicho elemento con el botón derecho del ratón.

Para un mejor conocimiento de las interfaces de la forma de trabajar con Blender acudir al manual de la aplicación (5).

4.6.2 Elementos a desarrollar

Tras esta breve explicación de la interfaz, puede comenzarse la creación los elementos del juego. El primer paso en la producción de los elementos 3D, es determinar cuales son los elementos que necesitamos desarrollar y cuales no necesitamos desarrollar.

En el partido de rugby se tiene un tablero, las casillas, el balón y los jugadores. Podría pensarse en un primer momento que el tablero ya tiene las casillas, pero esto no es así si se desea (como es el caso) que las casillas se muestre y oculten.

En principio como tablero sirve un plano sobre el que colocar los jugadores, para lo cual no es preciso crear un objeto 3D ya que el mismo motor gráfico proporciona herramientas para imprimir en pantalla planos líneas, puntos y planos.

Las casillas consistirán en cajas de igual ancho que largo y serán de dos tipos: unas

macizas y otras serán solo el borde de la casilla. Para diseñar el balón se tomará como referencia los balones reales utilizados.

Tal como se expresa en (2) existen dentro del juego distintas posiciones de los jugadores y también distintas razas; lo normal sería que los modelos de los jugadores fueran distinguibles según su raza y posición, pero esto resulta inviable para este proyecto porque es trabajo para un equipo de personas con experiencia previa, mientras que se dispone solo de un diseñador 3D sin experiencia previa; por este motivo se diseñará un jugador, y dado que el partido tiene lugar en el espacio deberá recordar tanto a un jugador de rugby como a un astronauta. Dicho modelo será el utilizado para todos los jugadores independientemente de su raza y posición, pero manteniendo colores distintos para los jugadores de distintos equipos.

Por motivos que mas adelante se detallará en el apartado 4.6.6, el diseño del jugador de rugby estará dividido en seis partes: casco, brazo derecho, brazo izquierdo, pecho, pierna derecha y pierna izquierda. Posteriormente mediante el motor gráfico se unirán las distintas partes en una sola a ojos del usuario.

4.6.3 Técnica utilizada para el modelado de los objetos 3D

A continuación se abordará el proceso de creación de los objetos mediante mayas poligonales.

No es de carácter obligatorio el uso de ninguna técnica para crear objetos 3D mediante mayas poligonales, ya que un diseñador experimentado o simplemente hábil, podría crear objetos valiéndose simplemente de su pulso y su buena observación. Y es cierto que para objetos simples como las casillas o el balón de rugby no se ha precisado el uso de ninguna técnica en el caso de las casillas. Las casillas al tratarse de cajas no supusieron ningún problema. El balón simplemente hubo que retocarlo unas cuantas veces en su largo y su ancho hasta que tuvo apariencia de balón de rugby.

Pero en el caso de diseñadores novatos, es posible que el diseño de formas complejas (como por ejemplo la de nuestro jugador), se convierta en un infierno de reajuste de tamaños y proporciones. Ya que aunque hagamos la mano, puede que los dedos resulten demasiado grandes o largos, que la longitud del brazo sea excesiva respecto a la de la mano o la del tronco, o aunque todo parezca del tamaño y proporción correctos se percibe un efecto raro al observarlos.

Todos estos problemas que los aficionados al dibujo reconocerán como propios cuando empezaron a dibujar, hay que imaginárselos observando el objeto de creación no ya en dos dimensiones sino en tres. Esto lleva a los diseñadores novatos a utilizar cilindros, cajas, etc. para representar las partes del cuerpo y posteriormente irlas adaptando (o esculpiendo) hasta conformar la silueta de un individuo de forma humana.

La técnica que se va a explicar no consiste en aproximarse mediante formas geométricas tradicionales (cubos, cilindros, pirámides, etc.), sino otra cuya principal idea es que partiendo de una imagen como plantilla resulta más sencillo obtener objetos 3D armoniosos. Dicha técnica se ha denominado alzado sobre imagen.

La técnica consiste en que a partir de la silueta del objeto, se crea un conglomerado de planos que completen dicha silueta, y que posteriormente se le de profundidad a dichos planos. El empleo de la técnica de alzado a partir de imagen no implica que al diseñador se le ahorre toda dificultad, ya que puede comprobarse que la técnica es muy simple. El uso de esta técnica ayudará para alcanzar el tamaño y la proporción correctos del objeto, pero es el diseñador el que debe hallar la manera en la que debe construirse el objeto.

Para crear los planos que cubran la silueta, el diseñador debe fijarse primero en los distintos elementos significativos que componen el objeto; especialmente aquellos elementos que tienen profundidades distintas. Deberá cumplirse que cada profundidad esté cubierta por un panto. Véase como ejemplo el brazo del jugador de rugby. En él se aprecia en un principio tres partes: la hombrera, la parte del brazo desde el hombro hasta la muñeca, y la mano. Entonces resulta interesante construir en principio la silueta con tres planos; pero tras un poco de reflexión se cae en la cuenta de que interesa separar el brazo en dos y diferenciar así donde se encuentra el codo. De igual manera necesitamos distinguir los dedos de la palma de la mano. Pero además hay un detalle más sutil, ya que es preciso que la hombrera esté subdividida con varios planos para poder luego darle profundidad y curvar la superficie.

Para que sea más fácil comprender a donde se quiere llegar se ilustrará mediante imágenes los pasos que llevamos hasta la fecha. La figura 4.59 a) es la imagen de la silueta de un brazo que utilizamos como plantilla, la figura b) es la silueta completada con planos y la figura c) es un detalle de los planos que componen la silueta.

Lo primero que cabe resaltar es que en la plantilla están marcadas todas las partes que deseamos diferenciar, para poder así extender los planos sobre dichas partes. Lo

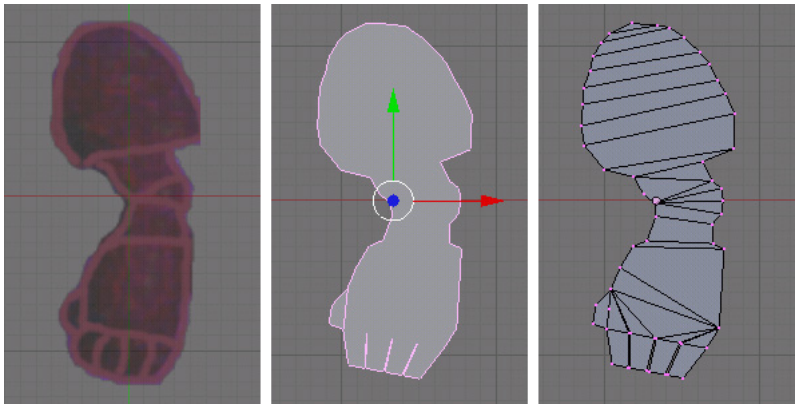


Figura 4.59: a)Plantilla b)Silueta c)Silueta en detalle

siguiente que puede apreciarse es que en la figura 4.59 c), para rellenar la silueta se han utilizado bastantes planos, diez para la hombrera, siete para la palma de la mano, cinco para los dedos y ocho para el brazo y antebrazo, pero además necesitaremos más planos ya veremos por qué.

El siguiente paso es darle profundidad a la silueta, ello lo conseguimos mediante la acción Extrude. El resultado es mostrado en 4.60.

En la figura 4.60 a) puede verse que el cambio respecto a la figura 4.59 c) consiste básicamente en darle grosor a la silueta, y al mirar el detalle de los planos en la figura 4.60 b), puede verse que hay cosas que corregir, por ejemplo los dedos tienen el ancho del brazo y habrá que reducírselo. También se han añadido más planos a la hombrera para poder darle forma curva a la superficie.

Llegados a este punto puede apreciarse el trabajo que el diseñador se ha ahorrado. Aunque debe retocar algunos aspectos del brazo, se tienen una serie de polígonos que se parecen bastante a la estructura que se desea obtener, y además se ha realizado el proceso (y aquí es donde está la clave de la técnica) de manera sencilla sin tener que preocuparnos de si se está guardando bien la proporción del tamaño de las distintas partes del brazo, porque dicha proporción ya estaba en la plantilla que hemos utilizado.

Hasta ahora se ha realizado la parte más sencilla y automática del proceso, aunque esta parte no ha estado carente de la necesidad de previsión por parte del diseñador (por ejemplo al separar los dedos en planos distintos, lo formar la hombrera con distintos planos para poder darle forma curva). La parte que queda ahora es la de modificar el

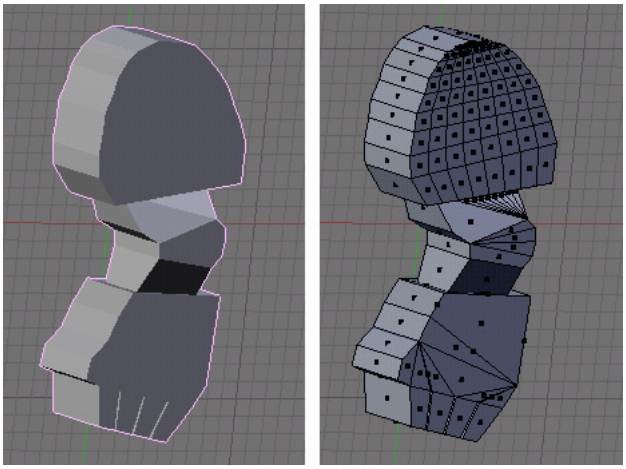


Figura 4.60: a)Brazo en perspectiva b)Detalle de los planos del brazo

objeto que ya se tienen para darle realismo (por ejemplo en los dedos) y el acabado que se desea.

Las imágenes a) y b) de la figura 4.61 son el mismo brazo visto de forma sólida y en detalle de sus planos. Podemos apreciar grandes cambios respecto a la última imagen del brazo vista en la figura 4.60, como es por ejemplo que la hombrera está curvada y que los dedos ahora son menos gruesos y más realistas. Pero hay otros cambios más sutiles y que están orientados a proporcionar realismo al modelo, y es que se han suavizado los bordes del guante, y también se ha dado volumen a lo que sería el bíceps del brazo suavizando de esa manera el borde que había resultado al realizar la extrusión de la silueta.

El paso realizado de las imágenes a) y b) de la figura 4.61 a la imagen c) es sutil pero significativo. En primer lugar se ha dotado a los dedos de falanges con lo que se consigue el extremo del dedo sea más fino que la base, y que además ahora puedan doblarse; la segunda diferencia es que la parte donde se unen el brazo y la hombrera ha sido hundida hacia la hombrera de forma que la sensación de que el brazo se introduce en la hombrera es mayor.

Ya puede darse por terminado el diseño del brazo, y con él la explicación de esta técnica de diseño. A continuación se resumen los pasos que se han llevado a cabo:

- Seleccionar imagen que utilizaremos como plantilla (debe tener delimitadas las distintas partes del objeto que queremos diferenciar)

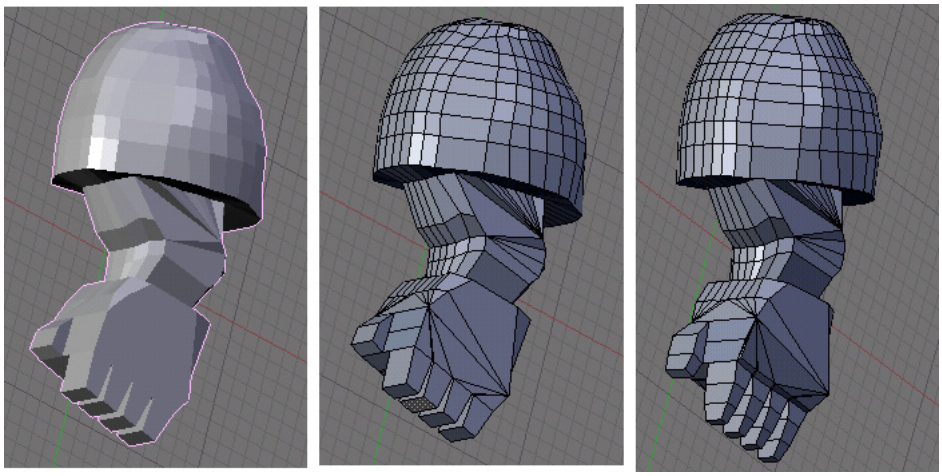


Figura 4.61: a)Brazo con dedos b)Detalle de brazo con dedos c)Detalle de brazo con dedos y falanges

- Rellenado de la plantilla mediante planos hasta conformar la silueta. Recordar que las zonas a distintas profundidades deben estar en planos distintos.
- Dar profundidad a la silueta mediante extrusión.
- Modificar las profundidades y alturas de los planos donde sea necesario.
- Suavizar los bordes producidos al realizar la extrusión.
- Observar que detalles podrían dar mayor realismo al modelo y añadirlos.

4.6.4 Generación de las animaciones de los elementos

El objetivo de este apartado es detallar como se han realizado las animaciones de los objetos 3D. Los únicos objetos animados son los jugadores, el balón podría haberse animado pero finalmente no se vio la necesidad puesto que podía suplirse mediante control por parte del motor gráfico. Dado que los jugadores están divididos en seis partes, se tomará como ejemplo el proceso de animación de una de esas partes.

Las animaciones que van a realizarse son del tipo skeletal, que son las que normalmente se usan cuando la animación supera los 100 frames. No ha sido necesario el uso de animaciones Morph en el proyecto.

En primer lugar es útil (aunque no necesario) determinar cuantas animaciones necesitan hacerse. Saber de antemano qué animaciones se necesitan hace más cómodo para el diseñador la tarea, ya que si tiene que hacer una animación de un jugador andando, otra del jugador corriendo y por último una del jugador realizando un pase, entonces dado que la animación de andar y correr son parecidas al diseñador le resulta más fácil hacerlas seguidas una de la otra, y la animación del pase hacerla la primera o la última.

A continuación se enumeran las animaciones que han sido necesarias:

- Animación del jugador corriendo. En este caso deben diferenciarse dos casos distintos, que el jugador posea el balón y que no lo posea, ya que un jugador con el balón debe protegerlo en todo momento y un jugador sin balón podrá mover los brazos libremente; por lo cual la animación será diferente en cada caso.
- Animación del jugador andando. Habrá una animación para cuando el jugador ande con el balón en la mano y otra para cuando ande sin él.
- Animación para cuando el jugador realice un placaje. Habrá una animación para cuando el jugador plaque con el balón en la mano y otra para cuando tenga las manos libres.
- Animación del jugador defendiéndose. Como en los casos anteriores, habrá dos animaciones según el jugador posea o no el balón.
- Animación del jugador realizando un pase largo.
- Animación del jugador realizando la recepción de un pase largo.
- Animación del jugador realizando de un pase corto.
- Animación del jugador realizando la recepción de un pase corto.
- Animación de un jugador realizando la patada inicial del balón.
- Animación de un jugador recogiendo el balón.
- Animación de un jugador cuando cae derribado.
- Animación de un jugador que está derribado.
- Animación de un jugador que se pone en pie.

- Animación de un jugador que se encuentra de pie y quieto. Para este caso habrá una animación para los jugadores con balón y otra para los jugadores sin balón.

Para crear una animación no existe una técnica que te sirva de base para generar los distintos momentos de la animación. Existen algunos factores importantes como por ejemplo la forma en que estén dispuestos los bones dentro del objeto 3D, ya que el giro y traslación de los bones afecta a los vértices del objeto. Como recomendación, siempre que se trate de una representación de un objeto vertebrado, debe intentarse que los bones representen de forma general el esqueleto del objeto.

Para llevar a cabo los giros y tiempos de la animación el diseñador deberá guiarse de su instinto, y sobre todo de la observación del movimiento en el mundo real para poder así intentar reproducirlo en nuestro objeto animado.

Después de crear la animación es muy recomendable observar reiteradamente la animación (y también desde distintos ángulos) para comprobar en primer lugar que la animación es correcta, ya que podría darse el caso de que algunos vértices se movieran de forma rara, y en segundo lugar para asegurarse que la animación resultante es lo suficientemente realista.

Cuando se trata de una animación en bucle como por ejemplo andar, la animación debe acabar y empezar con sus elementos en la misma posición. De otra forma no se conseguiría la sensación de continuidad y el usuario se percataría de que existe un salto en la animación.

A continuación se van a ilustrar los pasos en que se compone el proceso de creación de una animación.

Como primer paso, debe incorporarse el esqueleto al modelo 3D (en caso de no haberlo hecho ya), por lo tanto deberán primero crearse los bones teniendo cuidado de que su tamaño es adecuado (se recomienda que un bone nunca exceda el tamaño del objeto del cual forma parte), y que se colocan adecuadamente, consiguiendo que empiecen y acaben en zonas clave. Una vez que se ha creado el esqueleto, deberá enlazarse el objeto al esqueleto.

En la imagen a) de la figura 4.62 se muestra el esqueleto, que está compuesto de cinco bones.

En la imagen b) de la figura 4.62 puede verse como están enlazados el esqueleto y el objeto. Obsérvese como hay un bone que parte desde donde estaría el hombro y

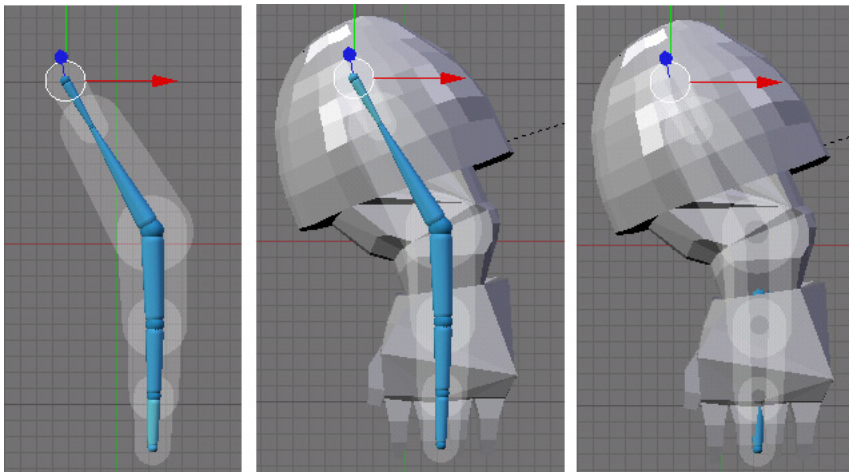


Figura 4.62: a)Esqueleto b)Brazo con esqueleto enlazado c)Brazo con esqueleto oculto

que termina en el codo, otro que va desde el codo a la muñeca, otro desde la muñeca a los nudillos, y otro desde los nudillos a la punta del dedo. Esto sirve para ilustrar lo anteriormente escrito sobre empezar y acabar los bones en zonas clave, ya que dada la disposición que le hemos dado podremos hacer movimientos de flexión de codo, muñeca y dedos.

En la imagen c) de la figura 4.62, el esqueleto está dentro del objeto, de forma que lo atraviesa por el interior; en la imagen b) puede verse porque estaba activada la opción rayos-x.

Una vez que el objeto y el esqueleto están enlazados, el siguiente paso es asignar a que bone pertenece cada vértice. Previamente el diseñador debe asegurarse que al enlazar el objeto con el esqueleto Blender no haya asignado los vértices a los bones; en ese caso deberemos primero eliminar los vértices asignados a los bones. La forma en se asignarán los vértices a los bones es por medio de la herramienta "*Weight Paint*" de Blender, que consiste en asignarle un peso de pertenencia del vértice que pintamos respecto del bone seleccionado. Un grado de pertenencia de 0% se representará con el color azul, mientras que un grado de 100% de pertenencia se marca con el color rojo.

Un vértice puede estar asignado a tantos bones como se quiera, teniendo en cuenta que al asignar un vértice a muchos bones se consigue un comportamiento del vértice poco previsible. Por ello que se recomienda que un vértice se asigne como mucho a cuatro vértices a la vez.

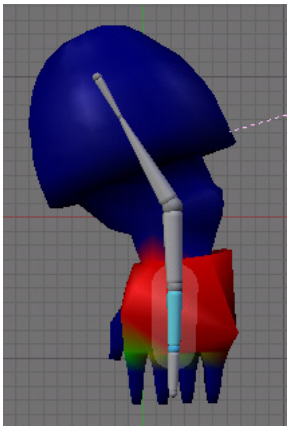


Figura 4.63: Ejemplo de Weight Paint

En esta la figura 4.63 puede verse un ejemplo de lo anteriormente dicho. En color celeste se muestra el bone seleccionado, que va desde la muñeca a los nudillos; en color azul oscuro vemos los vértices que tienen una influencia de 0% del bone; y en color rojo vemos los vértices que tienen una influencia del 100% del bone. Por la zona de los nudillos pueden verse en color verde los vértices que tienen una influencia aproximada del 50%.

Llegado este momento, si se quisiera utilizar el bone para desplazar los vértices, se obtendría el efecto deseado, y se desplazarían los vértices (y tan solo los vértices) que actualmente tienen peso, los que aparecen en la imagen 4.64 a).

Para que se moviera toda la mano, primero tendríamos que asignar al bone que va desde los nudillos a la punta de los dedos, los vértices de los dedos, y posteriormente ya podríamos hacer movimientos completos como si se estuviera girando la muñeca, ya que los bones guardan una jerarquía y su estado (posición y giro) depende a su vez del estado del padre.

En la imagen a) de la figura 4.64 se aprecia como al girar el bone el giro solo afecta a los vértices que se habían asignado a dicho bone (el resto de vértices del objeto no estaban asignados a ningún bone). Tras asignar los vértices de los dedos en la imagen b), puede apreciarse en la imagen c) como al girar el bone de la muñeca, el giro se transmite al bone de los dedos por lo cual gira toda la mano.

Una vez asignados todos los vértices del modelo a al menos un bone, se puede proceder al registro de la animación. Hasta aquí podría decirse que lo que se estaba

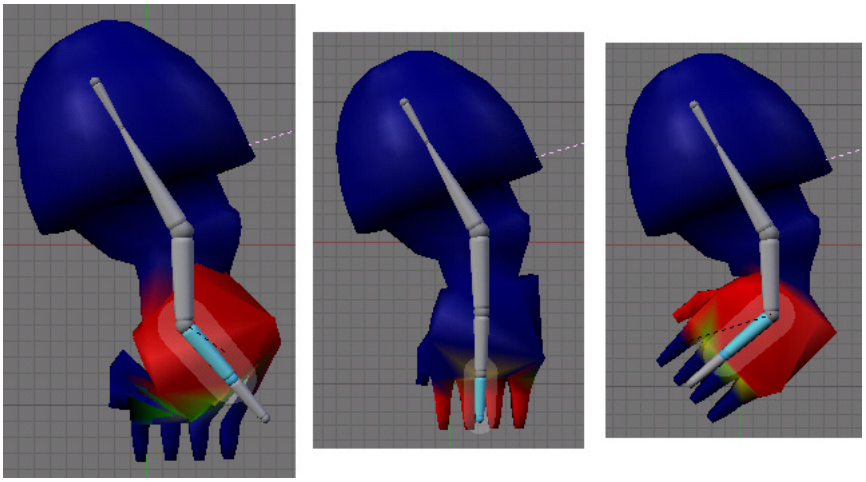


Figura 4.64: a)Giro de muñeca sin afectar a los vértices de los dedos b)Asignación de peso a los vértices de los dedos c)Giro satisfactorio de la muñeca

haciendo era preparar el objeto para crear la animación.

Crear una animación consiste en establecer el keyframe, es decir, la localización y rotación de un bone (establecer la localización y rotación de un bone es equivalente a decir establecer la localización y rotación de los vértices asignados a un bone) para un frame X determinado, y después establecer una localización o rotación para dicho bone para un frame posterior X+t y así sucesivamente hasta lograr la animación deseada.

En este caso, la posición de los vértices en los t-1 frames intermedios entre X y X+t se interpolarán según sus posiciones en X y en X+t. También existe la posibilidad de que se mantengan en la posición que tenían en el frame X y cuando llegue el frame X+t cambien bruscamente de posición, lo que se conoce como salto, pero este último no es el efecto buscado.

Dado que las posiciones de los t-1 frames intermedios se interpolan matemáticamente, no basta simplemente con especificar la posición inicial y final del movimiento, ya que si se quiere que los vértices pasen por una zona que no se encuentra en la interpolación de dichos puntos obviamente los vértices no pasarán.

Un ejemplo que ilustra esto es el movimiento del brazo, imaginemos que un personaje tiene que recoger un objeto que se encuentra debajo de una mesa y que inicialmente tiene la mano apoyada en la mesa; si solo se insertaran los keyframes de la posición inicial y final, entonces la mano atravesaría la mesa para coger el objeto, por lo tanto

sería necesario insertar más keyframes para hacer el movimiento más natural.

Cuantos más keyframes se inserten tanto más realista será el movimiento, pero también será más larga y difícil la creación de la animación. Debe llegarse por lo tanto a un equilibrio entre realismo de la animación y tiempo necesitado para ello.

4.6.5 Exportación de los modelos 3D

Los modelos 3D generados utilizan el formato de Blender (con extensión .blend) pero este no es un tipo de modelo reconocido por el motor gráfico OGRE, lo cual hace necesaria la exportación de modelos tipo Blender a modelos tipo OGRE (con extensión .mesh).

Con tal objetivo se ha empleado una herramienta que nos permite exportar archivos de Blender a formato .mesh. La herramienta convierte por medio de scripts en Python el modelo en de blender en un fichero XML, y posteriormente por medio de una aplicación de consola convierte el fichero XML en un archivo mesh.

Para integrar los scripts de Python en Blender se tienen dos opciones, o guardar el contenido del archivo Zip en la carpeta en la que se tenga configurada la ruta para scripts Python de Blender, o al revés, configurar la ruta de los scripts a la carpeta donde se descomprima el contenido del archivo Zip.

Una vez instalada la herramienta de conversión a archivos .mesh en el sistema, puede realizarse la exportación de los objetos 3D.

Para exportar un objeto primero debe seleccionarse dicho objeto en modo objeto. Posteriormente debe abrirse la ventana de scripts, y en el menú **Scrips** seleccionamos **Update Menus** (pero previamente se han debido colocar los scripts de exportación en la ruta de scripts de la aplicación); gracias a ello en el submenú **Export** ahora aparecerá la opción **OGRE Meshes**. Al pulsar aparecerá una ventana como la que se muestra en la figura 4.65.

En la ventana aparece el objeto 3D que tenemos seleccionado (en este caso "*brazo*"), el archivo .material que se va a generar, y diversas opciones de exportación como si se desea que el color sea el del ambiente, si se quiere que se copien las texturas, etc.

Pueden además añadirse las animaciones que se quiere que el mesh posea, indicando el frame de inicio de la animación, el frame de fin de la animación y un nombre para la animación.

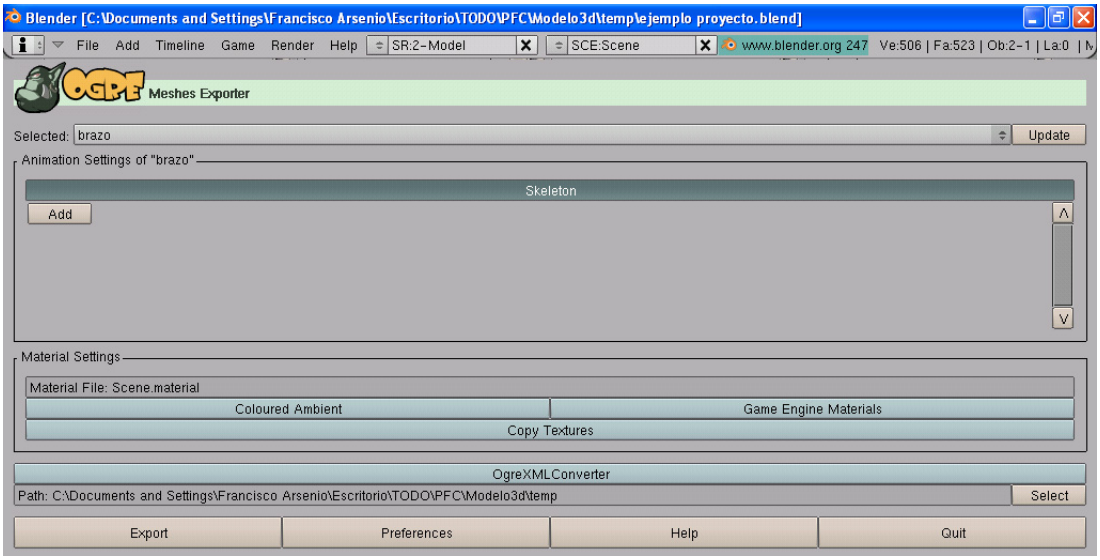


Figura 4.65: Ventana de exportación

Puede especificarse el directorio en el que se quiere guardar los .mesh generados. Pero antes de exportar el objeto debe pulsar el botón de preferencias, y aparecerá siguiente ventana de la figura 4.66. En dicha ventana deberá seleccionarse el programa que convierte los archivos XML en .mesh, y debe seleccionarse también la opción *manual*.

De vuelta en la ventana de exportación, al pulsar el botón de exportar aparecerá una ventana de log (ver figura 4.67) que nos informa del resultado de la exportación.

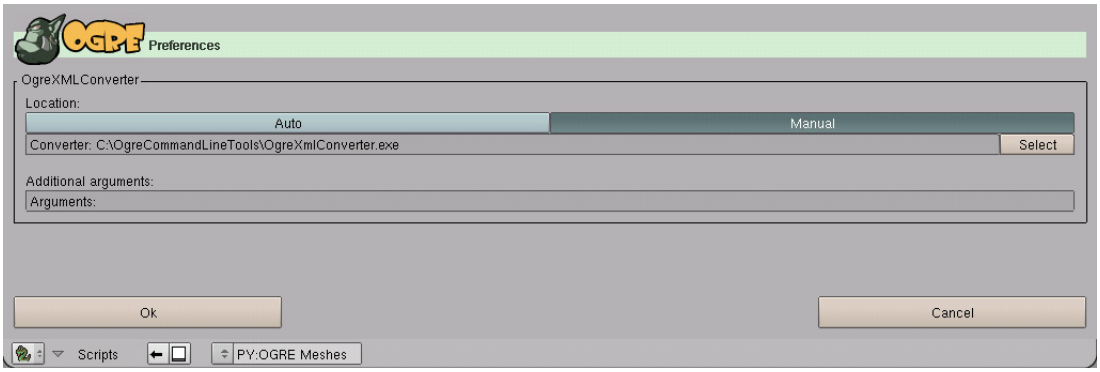


Figura 4.66: Ventana de exportación

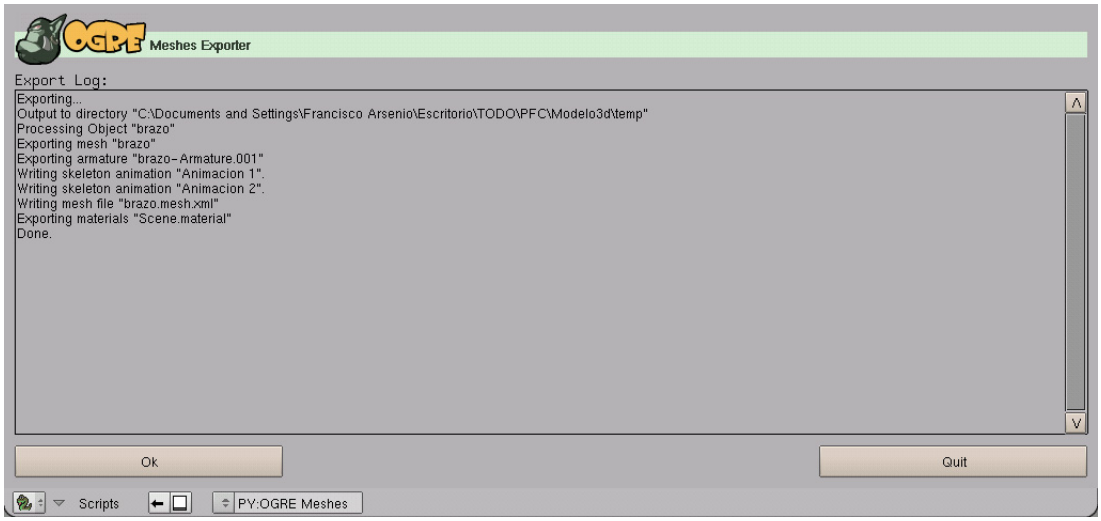


Figura 4.67: Ventana de exportación

4.6.6 Testing

Este apartado trata sobre las pruebas que deben hacerse a los modelos. Es de radical importancia probar los modelos para poder detectar de forma precoz los problemas que puedan surgir y así ahorrar la replicación de los mismos errores en el resto de modelos.

Principalmente las pruebas que deben hacerse a los modelos están relacionadas con la manera en la que se visualizan los modelos y sus diferentes características (color, animación) en el motor gráfico. Debe prepararse un banco de pruebas (no es necesario que sea el entorno final en el cual se utilizará el modelo) donde pueda comprobarse en primer lugar que el modelo se visualiza correctamente desde todos sus ángulos, y en segundo lugar que las animaciones se ejecutan correctamente. Por último hay que comprobar que el modelo adopta correctamente las texturas y los materiales que se le han establecido.

Es muy importante que cuando realizamos un modelo se pruebe que se visualiza correctamente en el programa de prueba antes de proceder a realizar las animaciones; ya que en caso de visualizarse mal, habrá que rehacer las animaciones. De igual manera no conviene hacer los modelos de forma industrial sin comprobar que tras hacer un modelo todo es correcto.

Ejemplos de esto son dos problemas encontrados al desarrollar los modelos 3D. Ambos son problemas con características del modelo que al exportarlas a formato .mesh

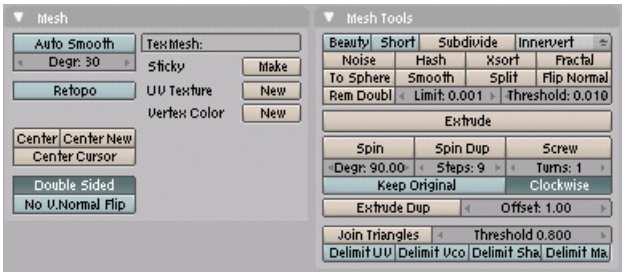


Figura 4.68: Opciones de la maya, a la izquierda (seleccionada) el botón de *Double Sided*, a la derecha el botón *flip normal*

hacían que el modelo actuara de forma extraña o se mostrase de forma incorrecta en el motor gráfico.

La primera característica es la de *Double sided plane*, que hace que ambas caras de un plano sean visibles al renderizarse. Normalmente cuando se define un plano, aparte de los puntos que lo delimitan se especifica también una normal. Para el motor gráfico la normal de un plano determina el lado visible de dicho plano, y lo que sucede con la característica *double sided plane* es que ambas caras del plano son visibles. Pero al exportar el modelo se elimina esta característica y al cargarlo en el programa de prueba algunos planos han desaparecido haciendo que se vea a través del modelo. La forma de arreglarlo es redireccionar las normales de los planos que desaparecen en sentido contrario mediante el botón *flip normal* que puede verse en la figura 4.68.

El otro problema que se encontró durante el modelado de los objetos 3D tiene que ver con las animaciones. Existe un problema al exportar las animaciones de un modelo, si en su esqueleto existen bones que comparten padre. La imagen 4.69muestra un ejemplo de bone con padre compartido.

Al realizar la exportación del modelo, la animación era un sinsentido, y no existe manera de arreglarlo por lo cual se tuvo que eliminar el esqueleto del modelo e insertar nuevos esqueletos en los cuales cada bone tenga un solo padre que no compartan con otro bone. También se tuvieron que rehacer las animaciones. Es por ello que se tiene por separado las extremidades, tronco y cabeza de los jugadores, porque sino no sería posible realizar las animaciones.

Estos dos problemas ocasionaron que se retrasara la producción de los modelos 3D, siendo necesario volver a versiones anteriores y en el caso de las animaciones a rehacer

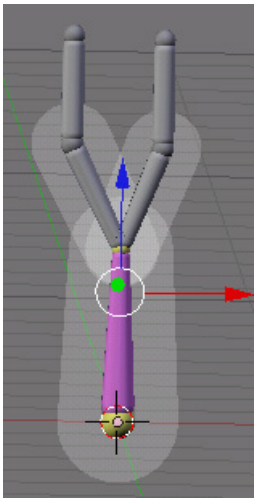


Figura 4.69: Ejemplo de dos bones que tienen un mismo padre

animaciones para los seis modelos. Es por ello de vital importancia para evitar en la manera de lo posible contratiempos lo expuesto en esta sección.

5

Manual de instalación

El proceso de instalación de instalación del juego consta de los siguientes pasos:

1. Instalación de DirectX 9c
2. Instalación de .Net Framework 2.0
3. Instalación de Mogre 1.4.6
4. Almacenamiento del código ejecutable del juego.
5. Almacenamiento de los modelos del juego.
6. Almacenamiento de los archivos de configuración.

Se recomienda que previamente al proceso de instalación se cierren todos los programas abiertos en el sistema.

5.1 Instalación de DirectX 9c

A continuación se detallará los pasos a llevar a cabo para la instalación de DirectX 9c. La instalación de este componente no es precisa si en el computador ya está instalada la versión especificada o una versión posterior.

1. Ir a la carpeta "SpaceRugby/ProgramasInstalacion" del CD de instalación y ejecutar el archivo directx_9c_redist.exe. Se abrirá una ventana de aceptación de licencia como la que podemos ver en la figura 5.1, pulsar Yes.

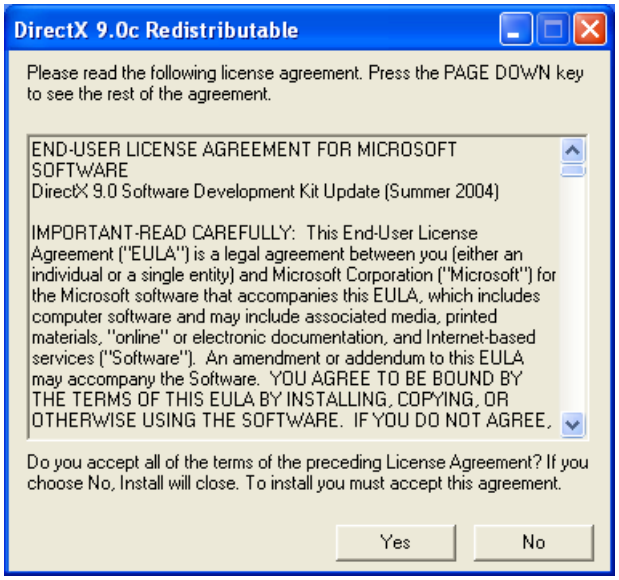


Figura 5.1: Ventana de aceptación de licencia

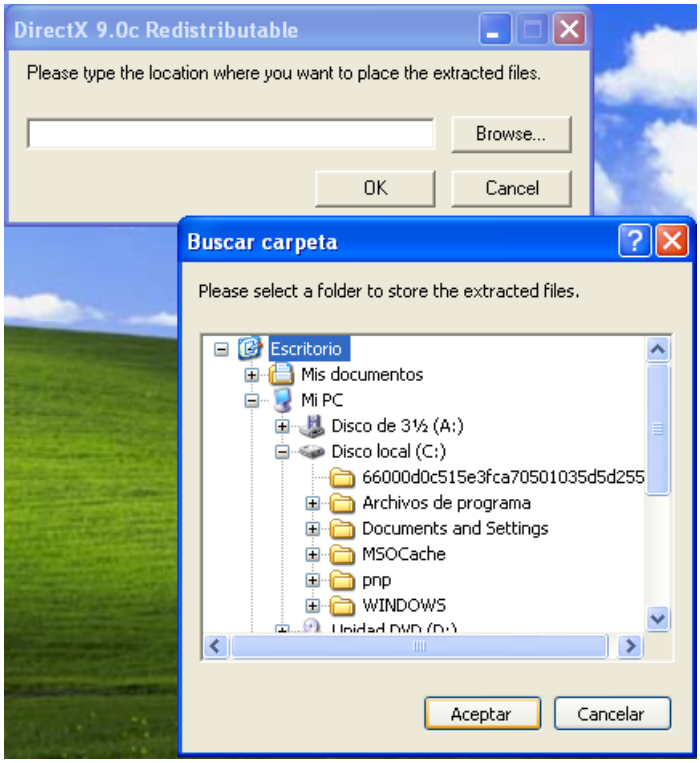


Figura 5.2: Ventana de extracción de archivos

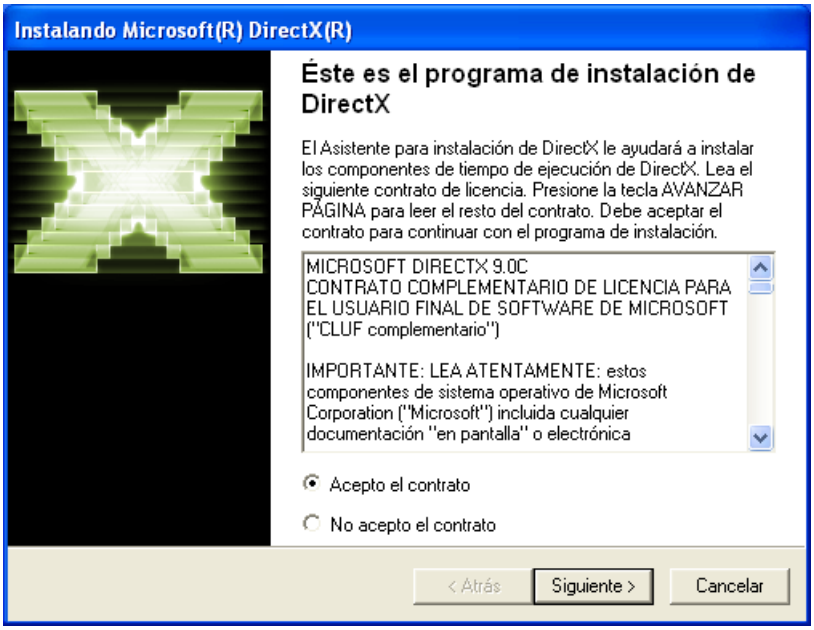


Figura 5.3: Ventana

2. Elegir la carpeta en la cual extraer los archivos y pulsar OK (figura 5.2).

3. Desde la carpeta en la que se descomprimieron los archivos, ejecutar el archivo dxsetup.exe; se abrirá una ventana como la mostrada en 5.3, seleccionar *Acepto el contrato* y pulsar *Siguiente*.

4. Se le informará que la instalación buscará componentes e el equipo, pulsar *Siguiente*. Cuando termine la instalación aparecerá una ventana como en la figura 5.4.

5.2 Instalación de .Net Framework 3.5

A continuación se detallará los pasos a llevar a cabo para la instalación de .Net Framework. La instalación de este componente no es precisa si en el computador ya está instalada la versión especificada o una versión posterior.

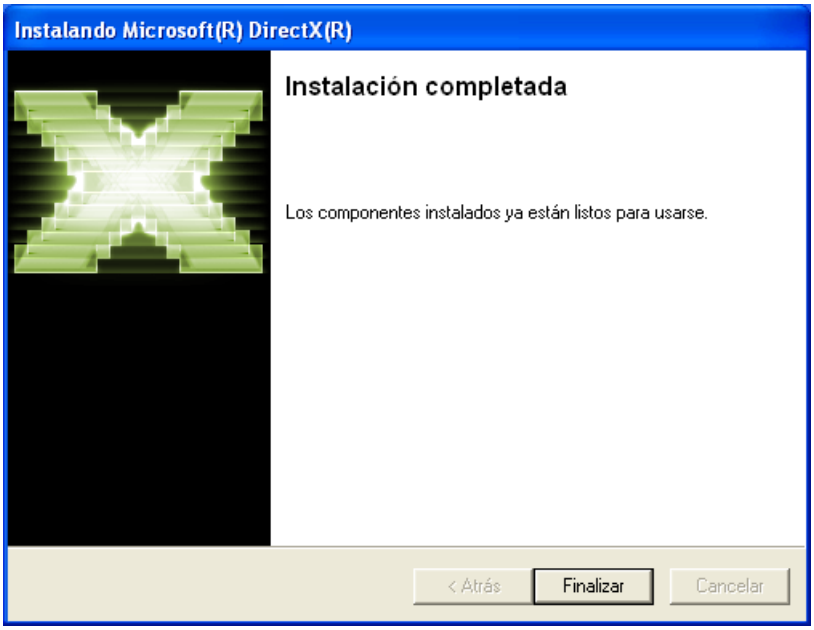


Figura 5.4: Ventana de fin de instalación

1. Ir a la carpeta "*SpaceRugby/Programas Instalación*" del CD de instalación y ejecutar el archivo dotNetFx3.5.exe. Se abrirá una ventana de aceptación de licencia, pulsar *Yes*.

2. Aparecerá una ventana con los términos de la licencia de uso de .Net Framework 3.5 (fig. 5.5). Seleccionar "*He leído los términos del Contrato de licencia y los ACEPTO*" y pulsar *Instalar*.

3. El siguiente paso es, como puede verse en la figura 5.6, la descarga e instalación del framework, es importante que durante este paso el equipo no se desconecte de internet.

4. Una vez se haya terminado la instalación, se mostrará la ventana que podemos ver en la figura 5.7. Al pulsar el botón *Salir* se sale del programa de instalación.

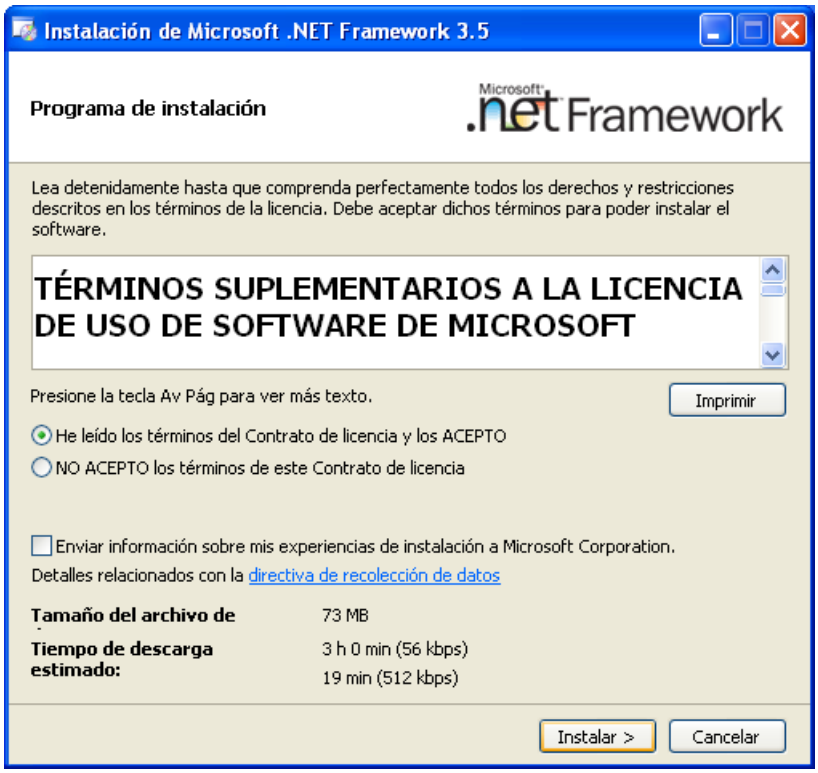


Figura 5.5: Ventana de aceptación de licencia

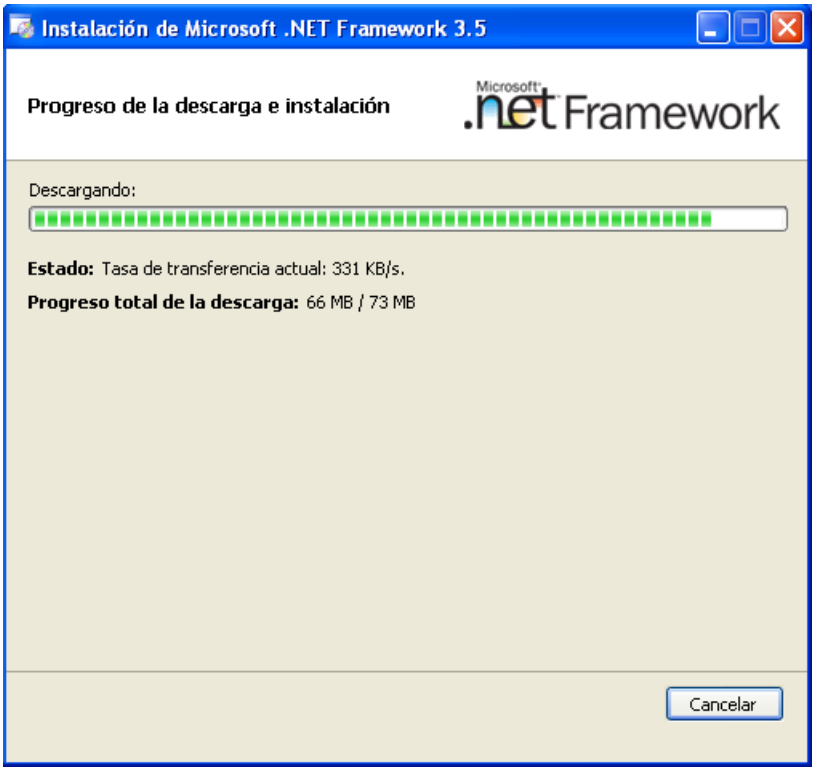


Figura 5.6: Ventana del proceso de instalación

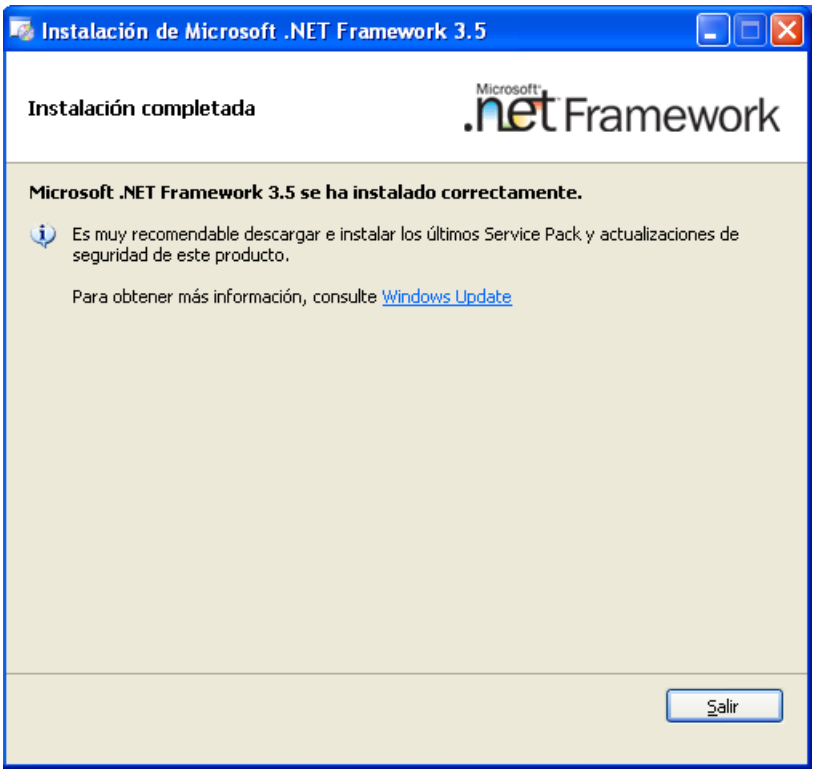


Figura 5.7: Ventana de finalización de la instalación

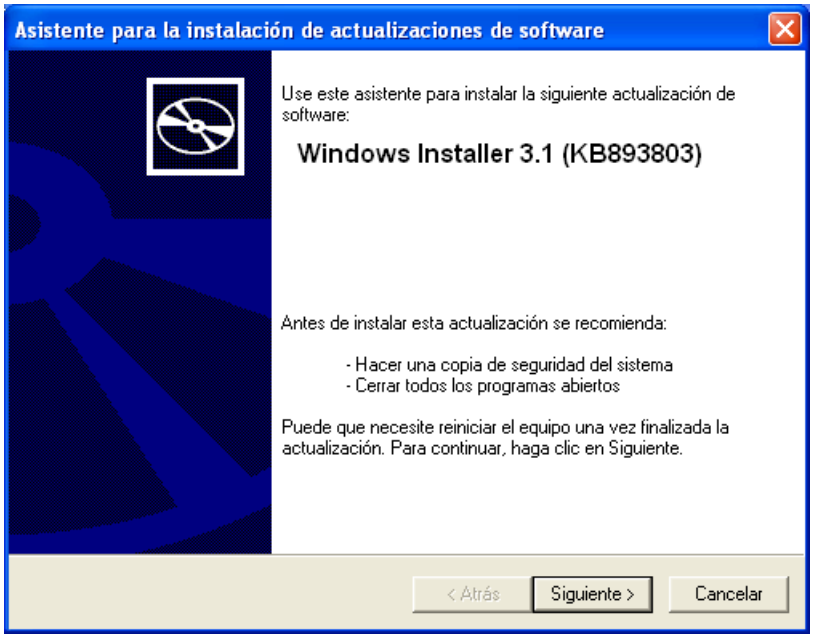


Figura 5.8: Ventana de instalación de Windows Installer

5.3 Instalación de Windows Intaller 3.1

Puede que durante la instalación del Framework 3.5 se notifique al usuario que es necesaria la instalación del Windows Installer 3.1. En ese caso han de seguirse los pasos descritos a continuación.

1. 1. Ejecutar el archivo WindowsInstaller.exe de la carpeta "*SpaceRugby/ProgramasInstalacion*". Aparecerá una ventana como la mostrada en 5.8, pulsar *Siguiente*.

2. En la pantalla de contrato de licencia elegir la opción *Acepto* y pulsar *Siguiente*.

3. Cuando finalice la instalación aparecerá una ventana como la mostrada en 5.9 y dejar que se reinicie el equipo.

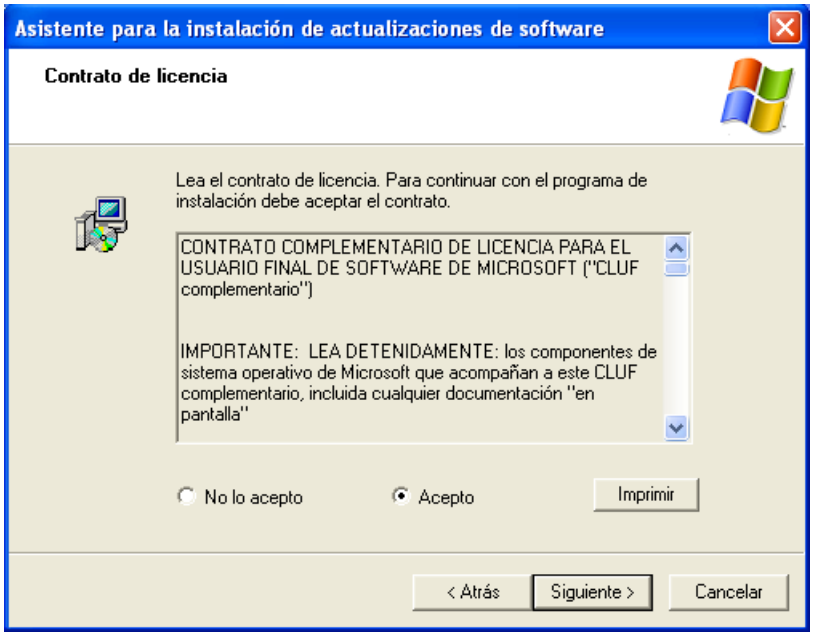


Figura 5.9: Ventana de aceptación de licencia

5.4 Instalación de Mogre 1.4.6

A continuación se detallarán los pasos a llevar a cabo para la instalación de Mogre. Previamente ha de asegurarse que no está instalado en el sistema ninguna versión anterior o posterior de estas librerías, ya que resultan incompatibles.

1. Ir a la carpeta "*SpaceRugby/ProgramasInstalacion*" del CD de instalación y ejecutar el archivo MOgreSDKSetup1.4.6_VC80.exe.

2. Cuando se muestre en pantalla la ventana de la figura 5.10, pulsar *Next*, en el siguiente paso de la aplicación de instalación pulsar el check "*I accept the terms in the License Agreement*" y posteriormente pulsar *Next*.

3. El siguiente paso de la instalación es elegir el directorio de instalación, tal como se muestra en la figura 5.11. Una vez hemos elegido el directorio destino pulsar *Next*, y a continuación *Install*.

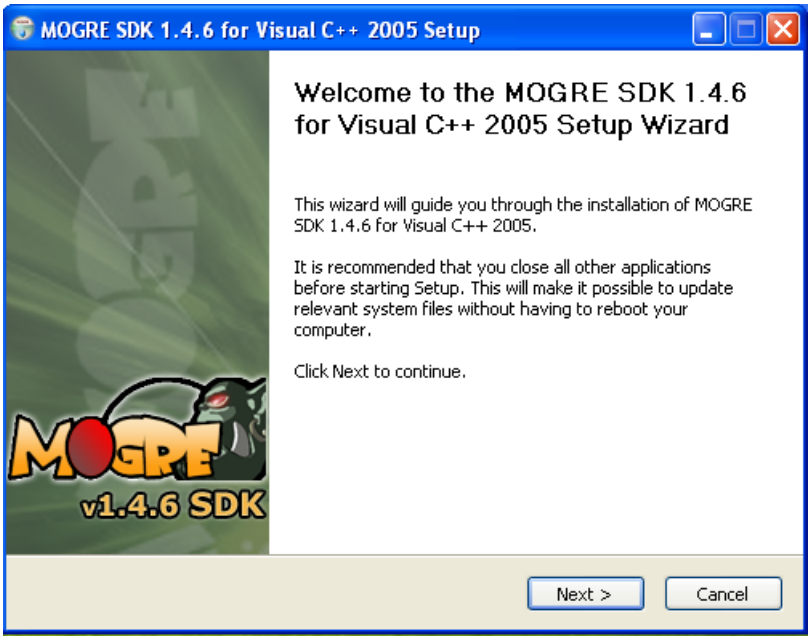


Figura 5.10: Ventana de instalación de Mogre

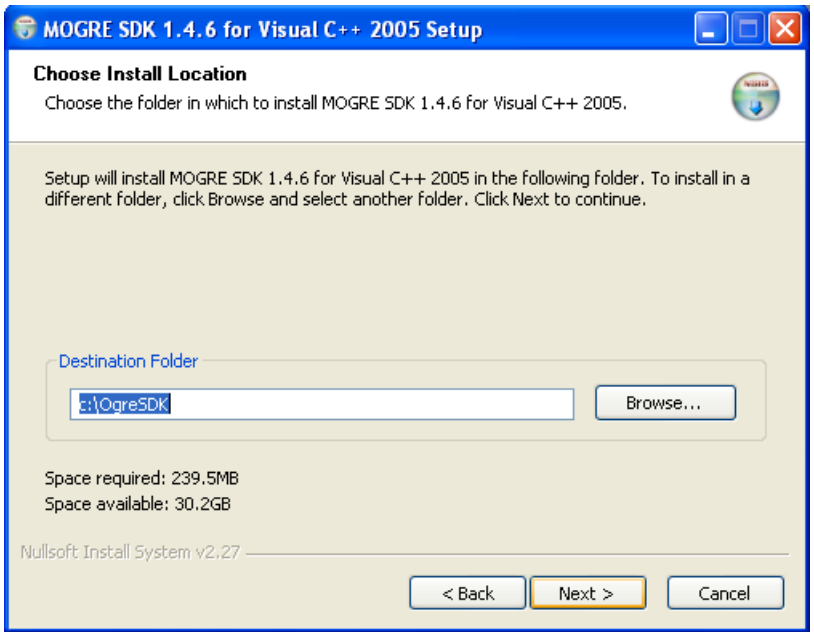


Figura 5.11: Ventana de elección de directorio de instalación

Figura 5.12: Ventana de finalización de instalación

4. Una vez terminada la instalación aparece la ventana mostrada en la figura 5.12, para salir del programa de instalación pulsar *Finish*.

5.5 Almacenamiento del código ejecutable del juego

El siguiente paso es copiar el código ejecutable del programa en el ordenador. Existen dos formas de hacerlo:

La primera forma es la más sencilla, consiste en copiar el contenido de la carpeta "*SpaceRugby/CodigoFuente/SpaceRugbyClient/SpaceRugbyClient/bin/Release*" dentro de la carpeta "*/bin/release*" del directorio de instalación de Mogre, no importa que se cree una carpeta dentro de dicho directorio para almacenar los datos del programa.

La segunda fórmula es más compleja, consiste en copiar el contenido de la carpeta "*SpaceRugby/CodigoFuente/SpaceRugbyClient/SpaceRugbyClient/bin/Release*" dentro de cualquier carpeta del sistema, y posteriormente configurar las variables de entorno (véase el apartado 5.5.1).

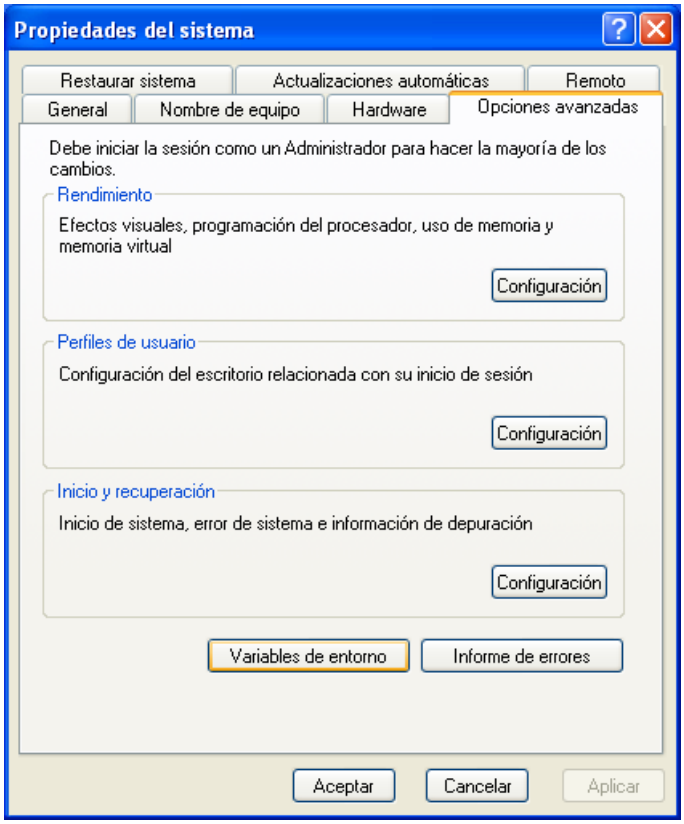


Figura 5.13: Ventana de propiedades del sistema

5.5.1 Configurar las variables de entorno

Para configurar las variables de entorno debemos en primer lugar pulsar el botón derecho sobre el icono de *Mi PC* y elegir la opción *Propiedades*. En la ventana de propiedades debe seleccionarse la pestaña *Opciones avanzadas*, y en dicha pestaña seleccionar el botón *Variables de entorno* (ver figura 5.13).

Puede que durante la instalación de Mogre ya se haya añadido una variable de entorno OGRE_HOME, de ser así no es preciso realizar lo descrito a continuación: pulsar el botón Nueva donde pone Variables de usuario, como nombre de variable escribir OGRE_HOME y como valor de la variable escribir la ruta completa de instalación de Mogre. El resultado debería como el que se muestra en la figura 5.14.

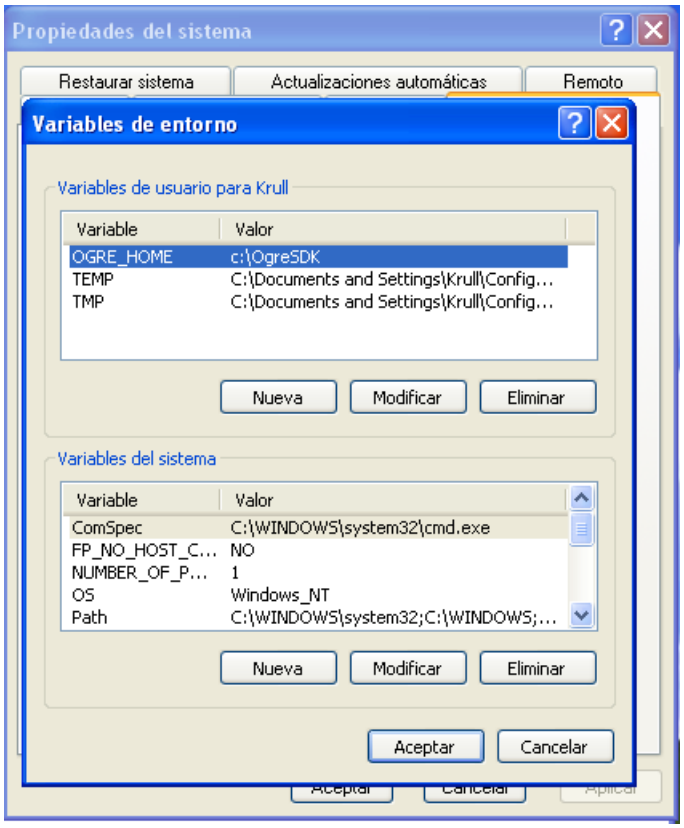


Figura 5.14: Ventana de variables de entorno

5.6 Almacenamiento de los modelos del juego

En la carpeta "*SpaceRugby/Modelo3D*" del CD de instalación se encuentra la información sobre el modelo de los jugadores, las texturas del programa y los scripts de los materiales y efectos especiales. A continuación se detallarán las carpetas cuyo contenido debe copiarse en las carpetas destino detalladas, si es necesario se sobrescribirán elementos en dichas carpetas.

El contenido de la carpeta "*SpaceRugby/Modelo3D/mesh*" debe copiarse en la carpeta "*OGRE_HOME/Media/models*". El contenido de la carpeta "*SpaceRugby/Modelo3D/scripts*" debe copiarse en la carpeta "*OGRE_HOME/media/materials/scripts*". El contenido de la carpeta "*SpaceRugby/Modelo3D/textures*" debe copiarse en la carpeta "*OGRE_HOME/media/materials/textures*".

En la explicación anterior debe entenderse que OGRE_HOME es la ruta de instalación de Mogre.

5.7 Almacenamiento de los archivos de configuración

En la carpeta "*SpaceRugby/Archivos configuración*" del CD de instalación hay cuatro archivos: media.cfg, ogr.cfg, plugins.cfg y resources.cfg. Son los archivos de configuración, indican al programa donde se encuentran los recursos, y deben copiarse dentro de la carpeta en la que se encuentra el ejecutable del programa.

Una vez copiados a la carpeta del programa, debe editarse el contenido tal como se indica a continuación:

En el archivo plugin.cfg es necesario sustituir la línea "*PluginFolder = C:/OgreSDK/bin/release*" por "*PluginFolder=OGRE_HOME/bin/release*".

En el archivo resources.cfg es necesario sustituir la ruta "C:/OgreSDK" por la ruta OGRE_HOME cada vez que aparezca en el archivo.

Debe entenderse que en la explicación anterior, OGRE_HOME es la ruta de instalación de Mogre.

6

Manual de usuario

Al iniciar el juego se muestra el nombre del juego durante cinco segundos, a continuación la pantalla de bienvenida (figura 6.1).

Las acciones que pueden realizarse son detalladas en los siguientes apartados.

6.1 Crear nueva cuenta

Desde la pantalla de bienvenida pulsar la etiqueta "**¿Aún no tiene cuenta?**". Se mostrará la pantalla de la figura 6.2. Rellenar todos los campos y pulsar el botón **Enviar**. Si todo ha ido bien se mostrará la ventana de la figura 6.3.

6.2 Solicitar una nueva contraseña

Para solicitar una nueva contraseña se deben pulsar en la etiqueta "**¿Olvidó su contraseña?**" desde la pantalla de bienvenida, se mostrará la pantalla de la figura 6.4. Rellenar el campo **Nick** y pulsar el botón **Enviar**. Si todo ha ido bien se volverá a la pantalla de bienvenida.

6.3 Entrar en el sistema

En usuario debe introducir su Nick y Contraseña en la pantalla de bienvenida y pulsar en botón **Entrar**. Si ha fallado la autenticación se mostrará el mensaje "**Falló la autenticación. Por favor revise que la contraseña y el Nick son correctos**".

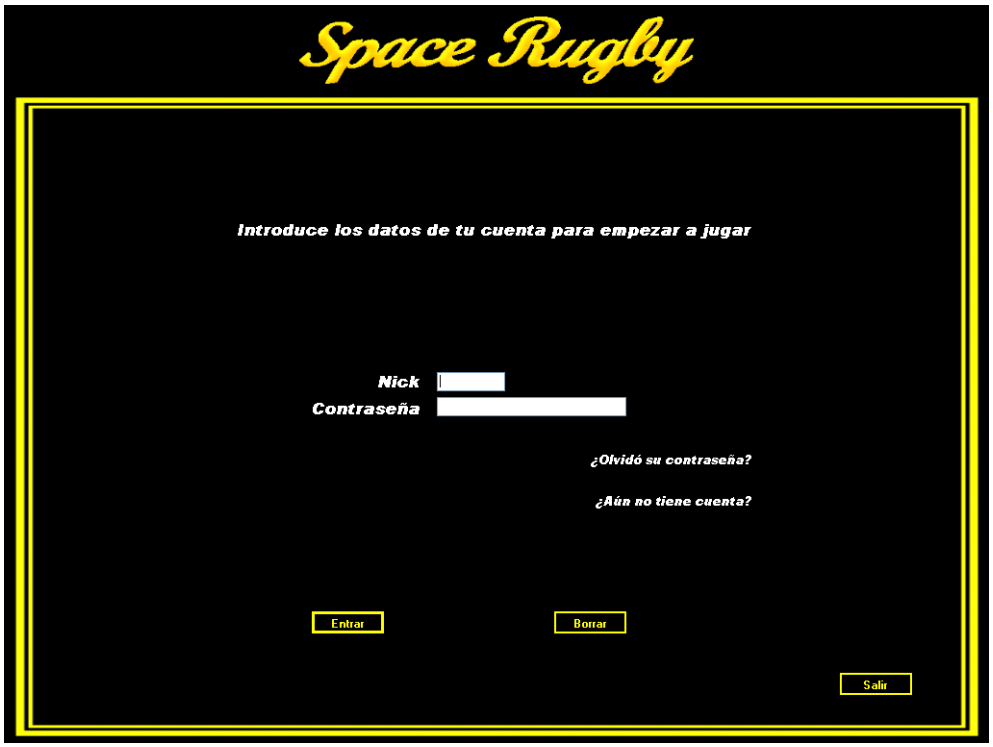


Figura 6.1: Pantalla de bienvenida

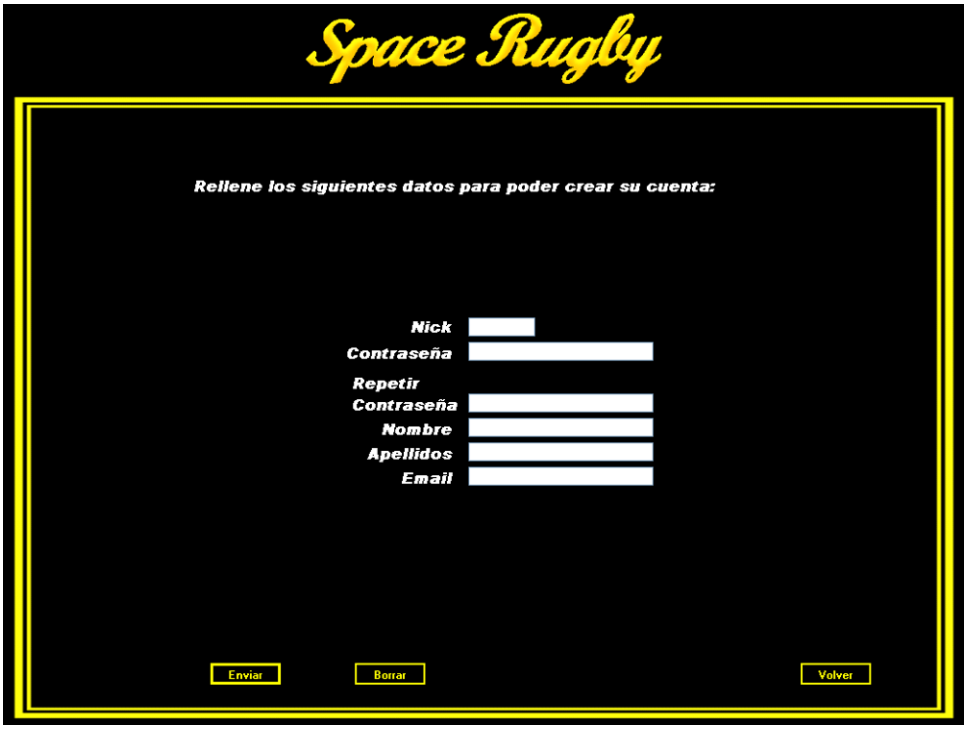


Figura 6.2: Pantalla de crear cuenta

Figura 6.3: Mensaje de cuenta creada

Figura 6.4: Pantalla de solicitud de nueva contraseña

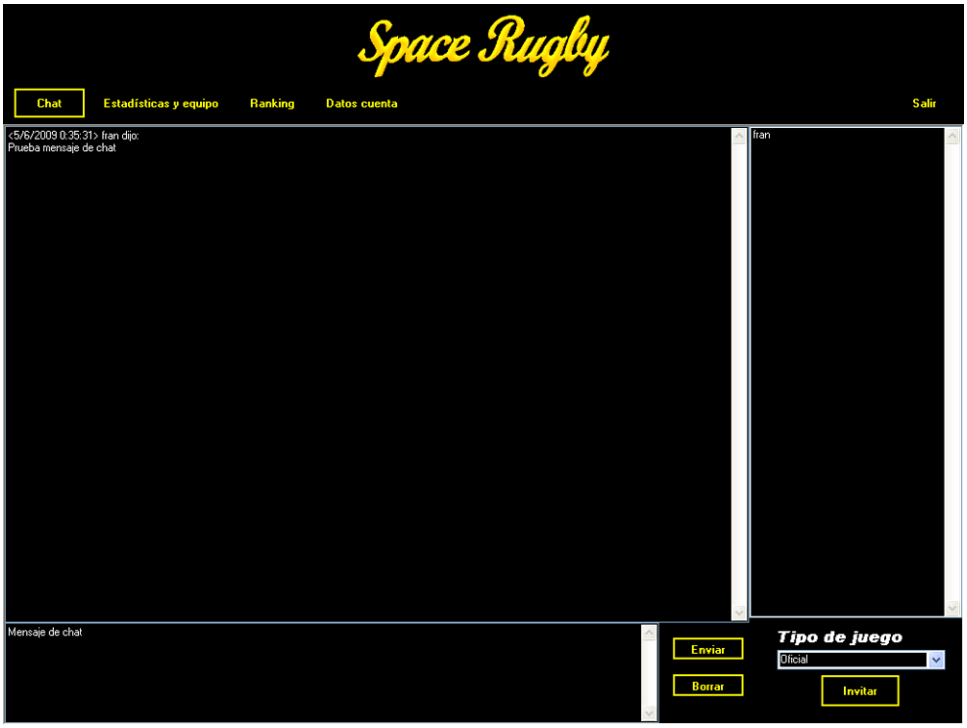


Figura 6.5: Pantalla de chat

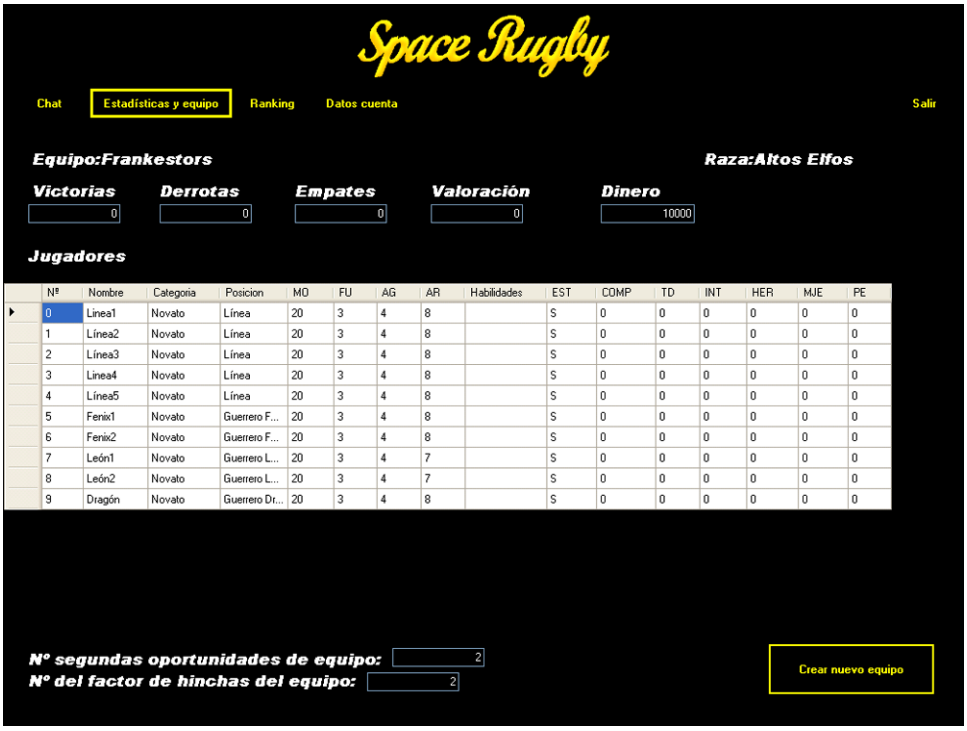
Si la autenticación del usuario ha sido correcta se mostrará la pantalla de chat (figura 6.5).

6.4 Escribir un mensaje de Chat

En la pantalla de chat, escribir en el área de texto situada en la parte inferior el mensaje que se desea enviar. Una vez se ha escrito pulsar el botón **Enviar** para mandar el mensaje de chat al servidor. Los mensajes de chat se muestran en el área de texto situada en la parte superior izquierda de la pantalla (ver figura 6.5).

6.5 Ver las estadísticas del equipo

La pantalla de estadísticas (figura 6.6) se visualiza al pulsar el botón **Estadísticas y equipo**. En dicha pantalla aparecen tanto los datos del equipo como de sus jugadores.



Nº	Nombre	Categoria	Posicion	MO	FU	AG	AR	Habilidades	EST	COMP	TD	INT	HER	MJE	PE
0	Linea1	Novato	Línea	20	3	4	8		S	0	0	0	0	0	0
1	Línea2	Novato	Línea	20	3	4	8		S	0	0	0	0	0	0
2	Línea3	Novato	Línea	20	3	4	8		S	0	0	0	0	0	0
3	Linea4	Novato	Línea	20	3	4	8		S	0	0	0	0	0	0
4	Línea5	Novato	Línea	20	3	4	8		S	0	0	0	0	0	0
5	Fenix1	Novato	Guerrero F...	20	3	4	8		S	0	0	0	0	0	0
6	Fenix2	Novato	Guerrero F...	20	3	4	8		S	0	0	0	0	0	0
7	León1	Novato	Guerrero L...	20	3	4	7		S	0	0	0	0	0	0
8	León2	Novato	Guerrero L...	20	3	4	7		S	0	0	0	0	0	0
9	Dragón	Novato	Guerrero Dr...	20	3	4	8		S	0	0	0	0	0	0

Figura 6.6: Pantalla de estadísticas y equipo

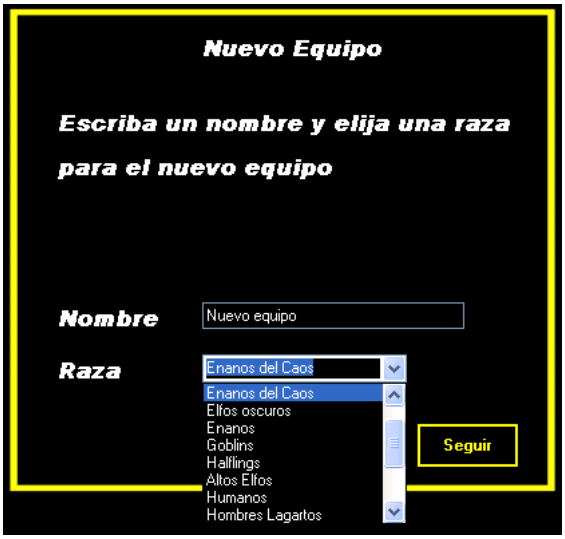


Figura 6.7: Pantalla de creación de nuevo equipo

6.6 Crear un nuevo equipo

Para crear un nuevo equipo el usuario debe pulsar el botón **Crear nuevo equipo** en la pantalla de estadísticas y equipo, se mostrará un mensaje de aviso que deberá aceptar, y a continuación se visualizará la ventana de la figura 6.7.

Deberá especificar el nombre del equipo y elegir una raza en el desplegable, a continuación pulsar el botón **Seguir**.

En la pantalla de compra de los jugadores (figura 6.8), para comprar un jugador debe elegirse una posición en el desplegable y especificar el nombre del nuevo jugador; a continuación debe pulsarse el botón **Comprar jugador**.

Los jugadores comprados aparecerán en un *grid* en la parte inferior de la pantalla de compra de nuevos jugadores. Las segundas oportunidades pueden comprarse y venderse especificando la cantidad y pulsando el botón **Comprar** o **Vender** respectivamente.

Una vez se ha terminado la compra de los jugadores pulsar el botón **Hecho** para terminar la creación del equipo y volver a la pantalla de estadísticas y equipo.

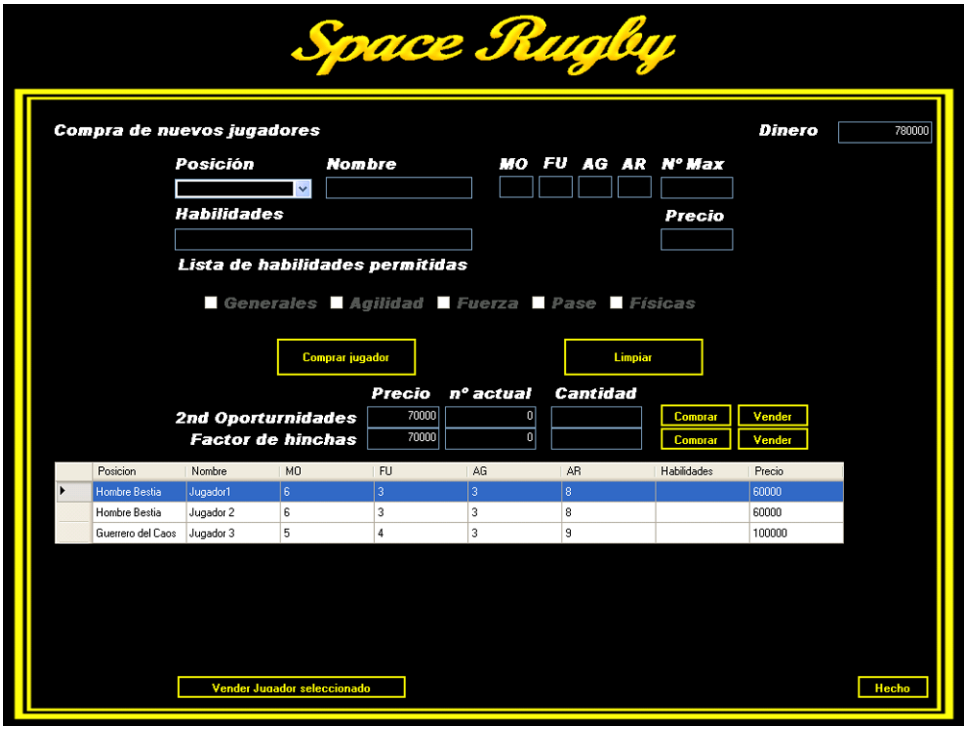


Figura 6.8: Pantalla de compra de nuevos jugadores

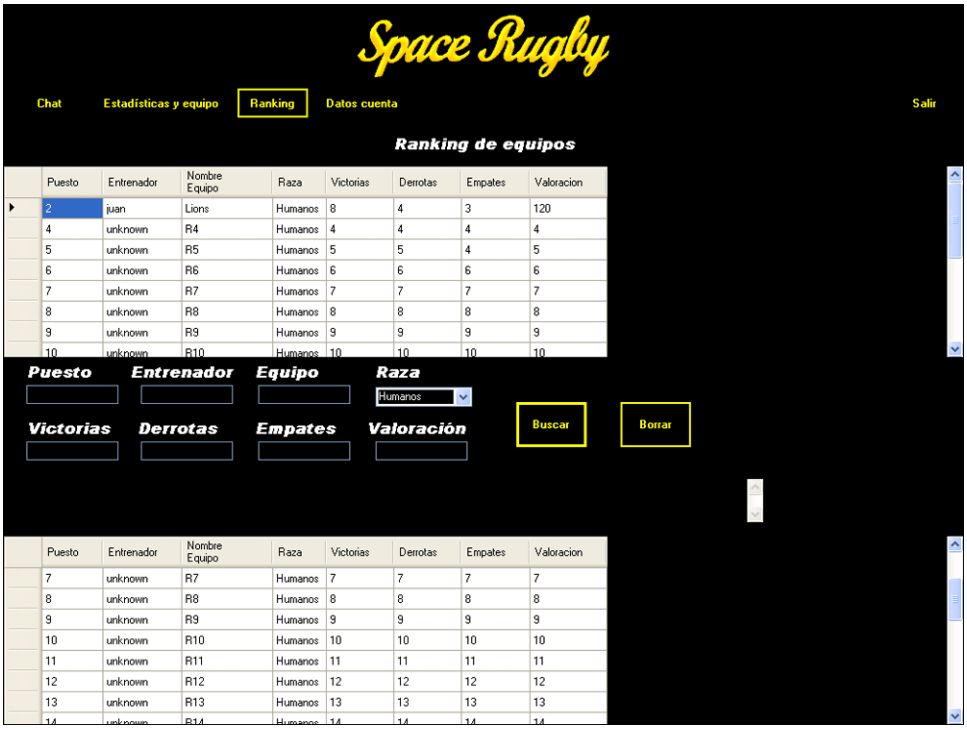


Figura 6.9: Pantalla de ranking

6.7 Ver ranking de equipos

La pantalla de ranking (figura 6.9) se visualiza al pulsar el botón **Ranking**. La pantalla muestra en la parte superior un *grid* con la lista de los mejores equipos del sistema.

6.8 Buscar un equipo en el ranking

Para buscar un equipo debe especificarse por lo menos uno de los siguientes elementos: puesto, entrenador, equipo, raza, victorias, derrotas, empates o valoración. Después de especificar el criterio pulsar el botón **Buscar**, el resultado de la búsqueda se carga en el *grid* de la parte inferior de la pantalla (ver figura 6.9).

6.9 Ver datos personales

Al pulsar el botón **Datos cuenta** se muestra la pantalla de datos personales (figura 6.10). Donde el usuario puede ver su Nick, nombre, apellidos y correo electrónico.

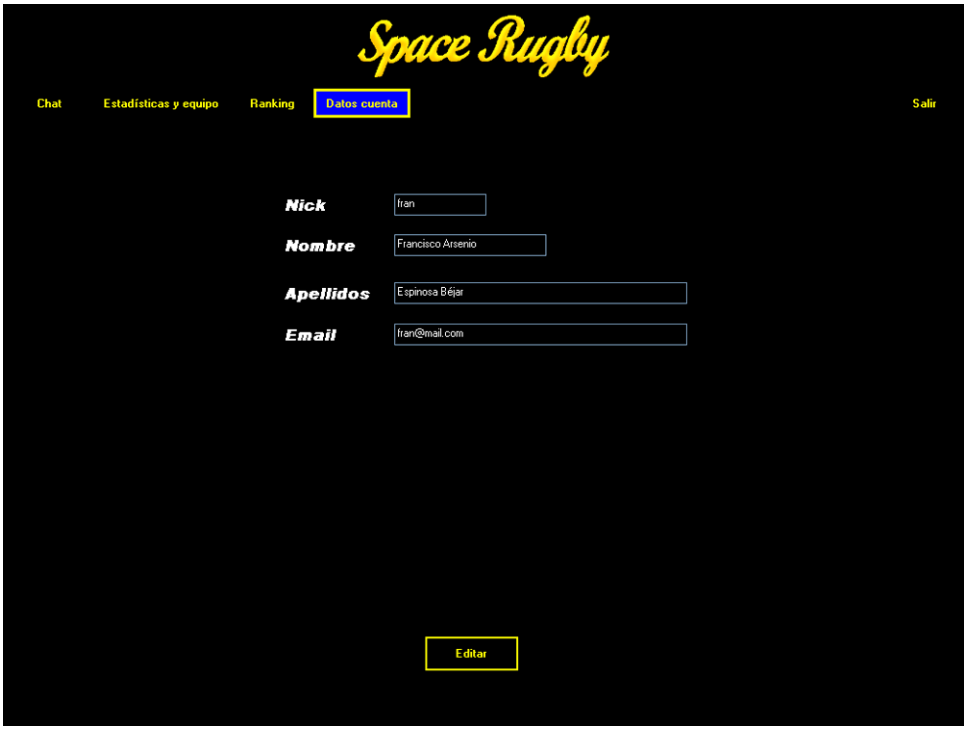


Figura 6.10: Pantalla de datos personales

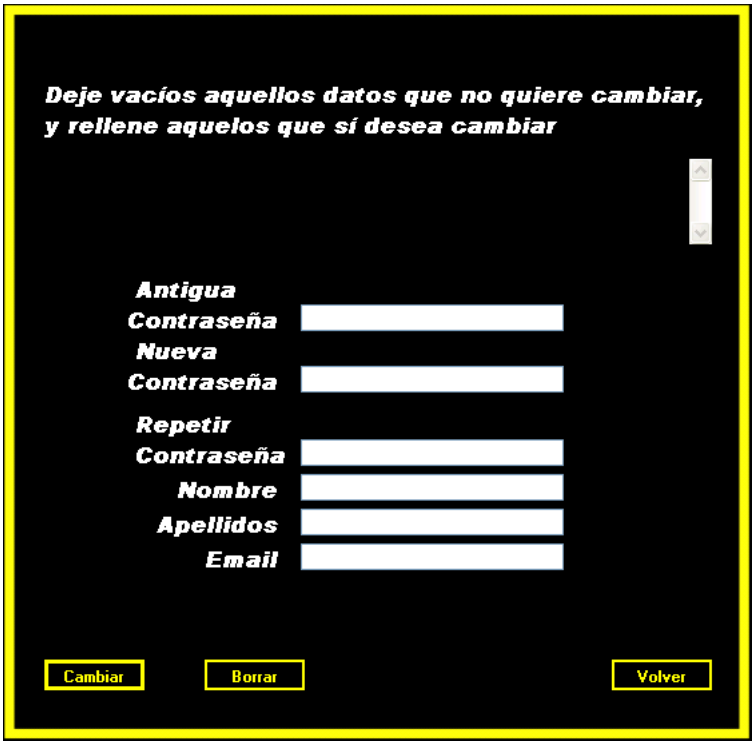


Figura 6.11: Pantalla de edición de los datos de la cuenta

6.10 Cambiar datos personales

Al pulsar el botón **Editar** en la pantalla de datos personales, se abre la ventana de edición de datos de la cuenta (figura 6.11).

El usuario debe rellenar los datos que quiere cambiar y dejar vacíos aquellos que no desea que se modifiquen. Siempre debe rellenarse el campo "**Antigua contraseña**", que debe coincidir con la contraseña actual del usuario para que la solicitud de cambio de datos sea aceptada. Pulsar el botón **Cambiar** para hacer efectivo el cambio de datos.

6.11 Invitar un jugador a jugar un partido

En la parte derecha dela pantalla de chat hay un listado de todos los usuarios conectados. Para invitar a un jugador a jugar un partido hay que en primer lugar seleccionar su Nick de la lista de usuario conectados, a continuación seleccionar el tipo de partido

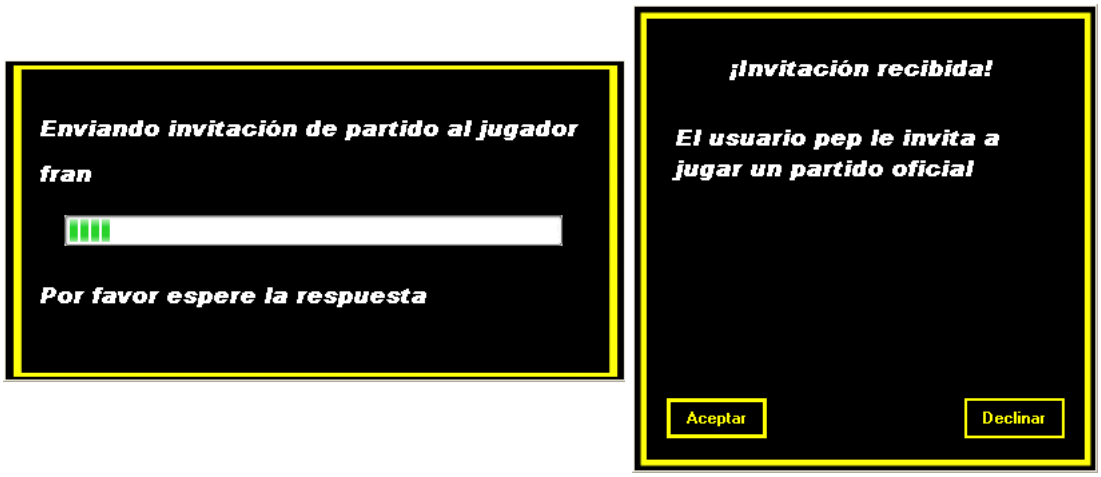


Figura 6.12: a)Pantalla de espera b)Pantalla de recepción de invitación

Figura 6.13: a)Pantalla de resultado de la invitación b)Pantalla de confirmación

del desplegable situado en la parte inferior derecha de la pantalla, y por último pulsar el botón **Invitar**. Mientras se espera la respuesta del jugador el usuario verá la pantalla de la figura 6.12 a).

6.12 Recibir invitación a jugar un partido

Cuando un jugador es invitado a jugar un partido, se le muestra la pantalla de la figura 6.12 b.

Una vez que el usuario acepta o rechaza la invitación, se le notifica la respuesta al usuario que realizó la invitación (figura 6.13 a)).

6.13 Salir del juego

Para salir del juego debe pulsarse el botón **Salir** desde la pantalla de chat, estadísticas de equipo, ranking o datos de usuario. Se solicitará al usuario que confirme si desea salir del juego (figura 6.13 b)).

6.14 Pantalla del partido

En figura 6.14 podemos ver la pantalla de juego.

En la parte superior se encuentra la información de estado del partido, en ella podemos ver los nombres de los equipos, el turno en el que se encuentran, los touchdowns marcados y las fichas de segunda oportunidad que todavía pueden usar.

La parte inferior de la pantalla de juego está destinada a mostrar al usuario los controles específicos de la acción en curso.

6.15 Colocar los jugadores

Cuando es el turno del equipo para colocar sus jugadores, se muestran las casillas del campo del equipo (ver figura 6.14).

Para colocar un jugador en el campo de juego primero debe seleccionarse su nombre en el desplegable de la parte inferior izquierda, y a continuación pulsar con el botón izquierdo del ratón sobre la casilla en la que quiere colocarse el jugador.

Si se desea quitar un jugador de una posición hay que pulsar con el botón izquierdo del ratón la casilla en la cual está colocado el jugador.

Una vez se ha terminado de colocar los jugadores pulsar Terminado.

6.16 Patada Inicial

Para realizar la patada inicial el usuario debe elegir el jugador que realizará la patada, y la casilla a donde se pateará el balón. La casilla destino de la patada inicial debe estar en el campo del equipo contrario, cuyas casillas se encuentran visible durante la preparación de la patada inicial.

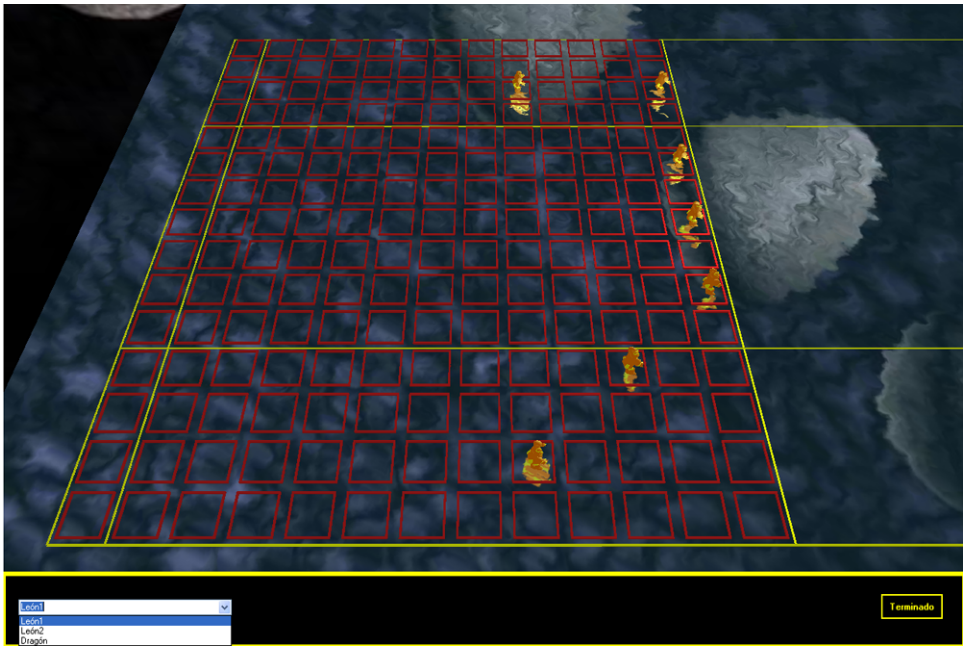


Figura 6.14: Ejemplo de colocación de jugadores

Al seleccionar el jugador, se activa un círculo de luces alrededor de dicho jugador; de igual manera cuando se selecciona la casilla destino, se marca con un haz de luces (ver figura 6.15).

Una vez se ha elegido el jugador y la casilla a la que lanzar el balón, pulsar el botón **Patear** para realizar la patada inicial.

6.17 Realizar una acción de turno

Para realizar una acción de turno, en primer lugar debe seleccionarse el jugador pulsando sobre él con el botón izquierdo del ratón; cuando un jugador se selecciona se marca con un círculo de luces alrededor suyo.

Al seleccionar un jugador, se muestran sus características en la parte inferior de la pantalla. En el combo de la parte inferior izquierda se cargan las posibles acciones del jugador tal como se muestra en la figura 6.16.

Para realizar una acción con el jugador seleccionado debe elegirse la acción en el desplegable y pulsarse el botón **Ejecutar**.

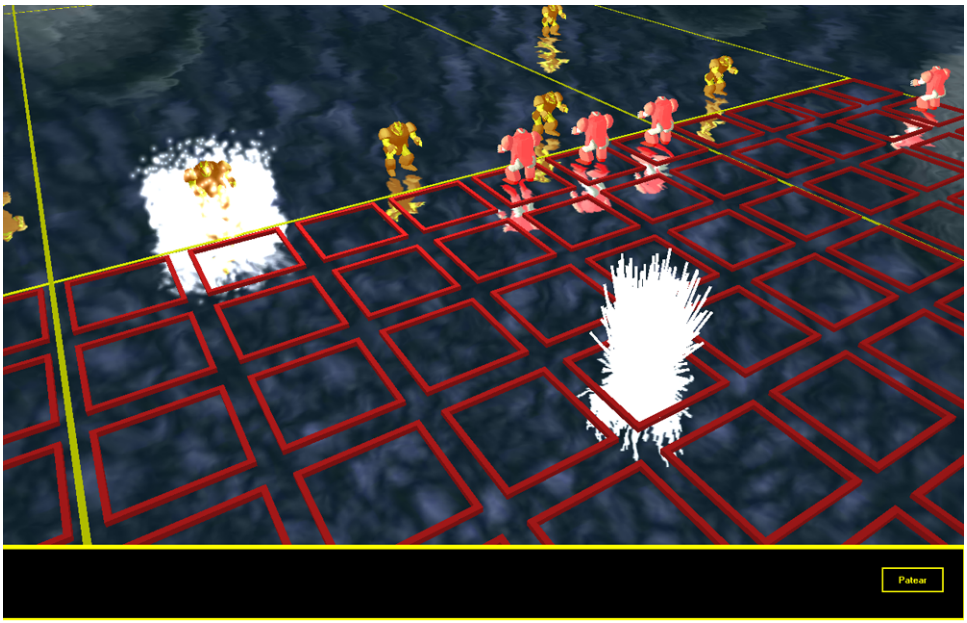


Figura 6.15: Captura de patada inicial

Figura 6.16: Ejemplo de elección de acción

Figura 6.17: Captura de movimiento de un jugador

6.18 Mover un jugador

Una vez se ha seleccionado la acción de mover un jugador, se muestran en pantalla todas las casillas del tablero.

El usuario debe seleccionar una a una las casillas por las que desea que se mueva el jugador, que adquirirán la forma de una baldosa amarilla como puede verse en la figura 6.17.

Por último debe pulsarse el botón **Mover** para iniciar la acción.

6.19 Realizar un placaje

Cuando se elige la acción de placar un jugador se muestran al usuario las casillas situadas en la zona de defensa del jugador atacante (marcado con un círculo de luces verdes) como puede verse en la figura 6.18.

El usuario debe elegir el jugador al que desea placar pulsando sobre él con el botón izquierdo del ratón, dicho jugador se marcarán con un círculo de luces violetas.

Tras determinar el jugador que se desea placar, pulsar el botón **Placar** para efectuar el placaje.

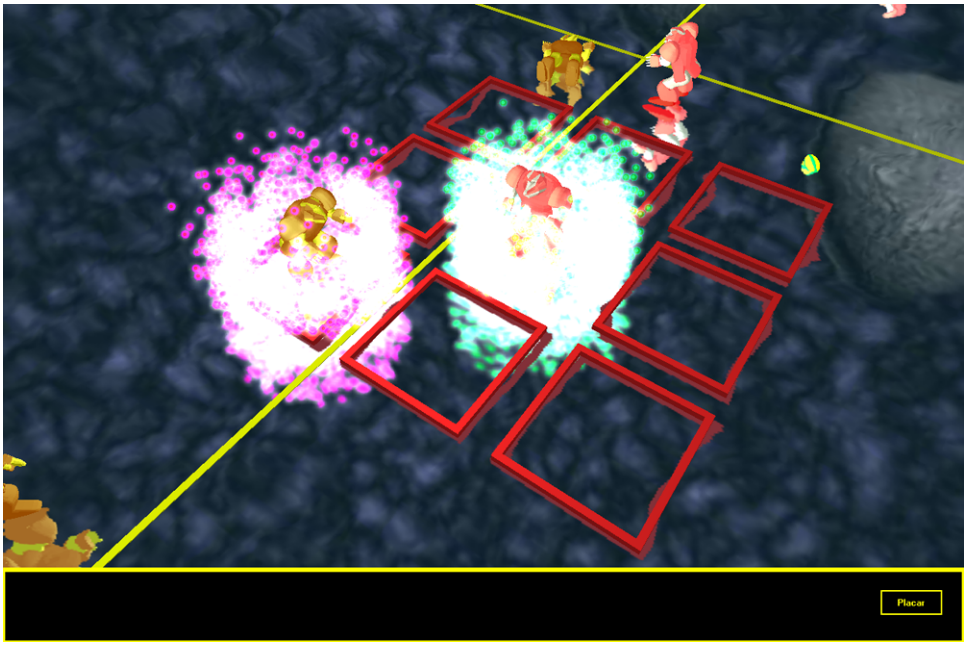


Figura 6.18: Captura de un placaje

6.20 Realizar un pase

Cuando un jugador posee el balón, puede elegir pasarlo a otro jugador de su equipo. El usuario debe elegir el nombre del jugador que recibirá el pase dentro del desplegable que se muestra en la pantalla a de la figura 6.19.

A continuación se informará al usuario la categoría del pase y si desea continuar con la ejecución del pase (ver figura 6.19 b). Para continuar con el pase pulsar **Si**, si se desea elegir otro jugador pulsar **No**.

Si hay algún jugador del equipo contrario en la trayectoria del pase, el entrenador del equipo contrario tendrá la oportunidad de intentar la intercepción del pase. Para ello debe seleccionar el jugador en el desplegable de la ventana de la figura 6.20 y pulsar **Hecho**. Si en lugar de **Hecho** pulsa **Cancelar**, no se intentará la intercepción.

6.21 Recoger el balón

Cuando un jugador se encuentra en la misma casilla que el balón, éste puede intentar recoger el balón como se muestra en la figura 6.21

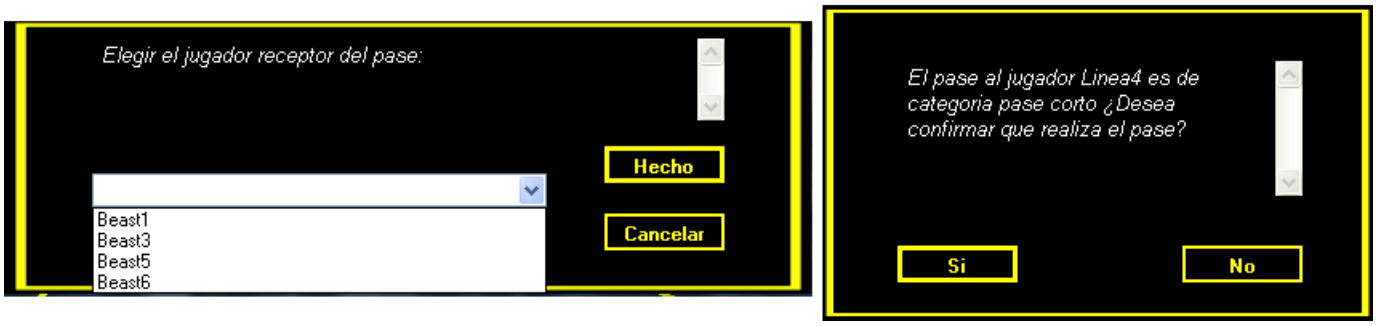


Figura 6.19: a) elegir el jugador receptor del pase b) confirmar pase

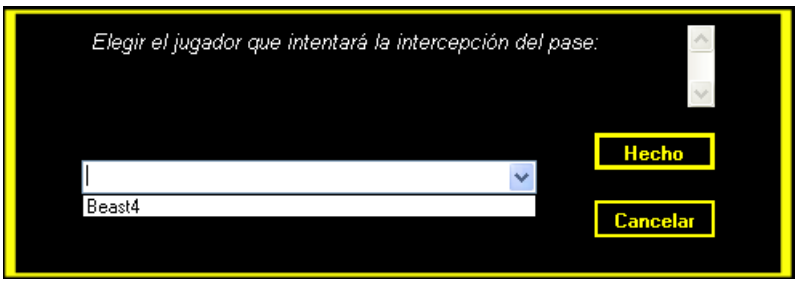


Figura 6.20: Ventana de selección de jugador interceptor

Figura 6.21: Captura de un jugador recogiendo el balón

Figura 6.22: Captura de un jugador derribado

6.22 Poner en pie un jugador derribado

Un jugador que está aturdido se encuentra boca abajo en el terreno de juego, al seleccionar la acción de poner en pie permitimos que el jugador se levante y pueda efectuar los movimientos de mover, placar, etc.

6.23 Girar un jugador aturdido

Cuando un jugador es aturdido se coloca boca arriba en el terreno de juego como puede verse en la figura 6.23.

Antes de ponerlo en pie debe girarse para posicionarse boca abajo.

6.24 Abandonar el partido

Para abandonar el partido el usuario debe pulsar el botón **Abandonar partido** situado en la parte superior derecha de la pantalla. Se mostrará una ventana de confirmación (figura 6.24), al pulsar Si se termina el partido.

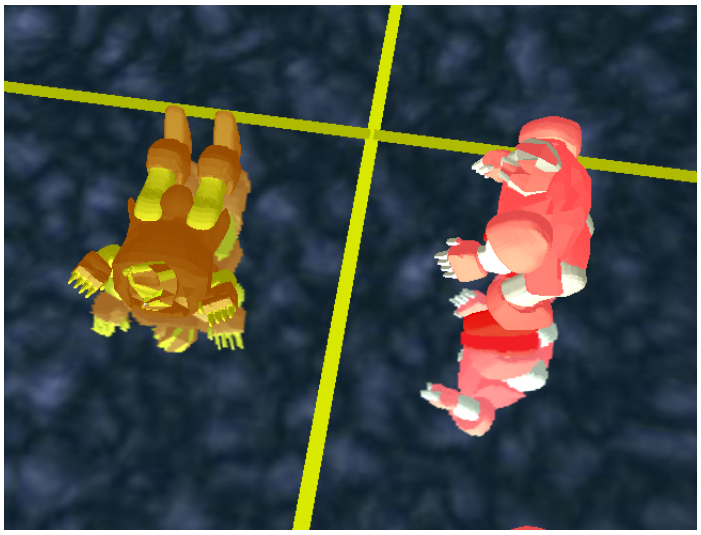


Figura 6.23: Captura de un jugador aturdido

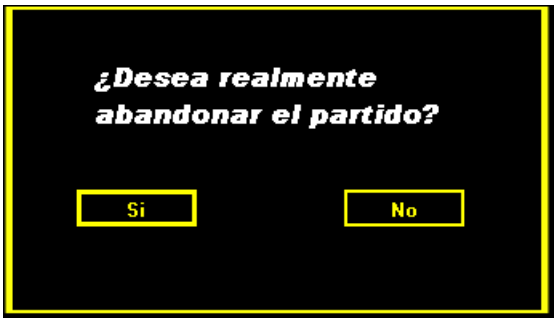


Figura 6.24: Ventana confirmación de abandono del partido

7

Presupuesto

En este apartado se va a detallar el presupuesto requerido para el proyecto, incluyendo todos los gastos precisos para su desarrollo y puesta en marcha.

El tiempo empleado en la realización del proyecto ha sido de 8 meses y una semana (210 días). A continuación se detalla el tiempo empleado en cada fase que forma el proyecto.

7.1 Desglose por fases del proyecto

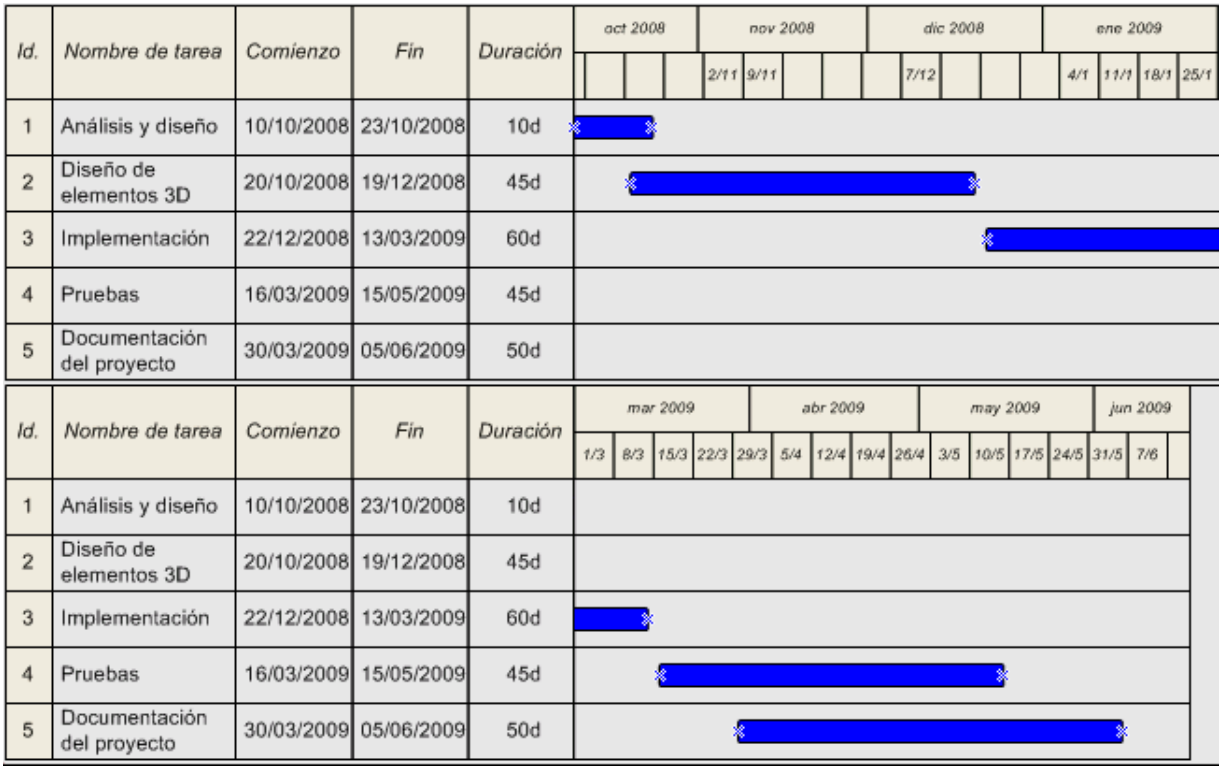
En la tabla 7.1 se especifican las actividades que se han llevado a cabo para la realización del proyecto, incluyendo las horas que han sido necesarias para llevar a cabo cada una de ellas:

En el diagrama de Gantt de la figura 7.1 puede apreciarse el comienzo y final de las distintas fases.

Fase	**Horas**
Análisis y diseño	40 h.
Documentación del Proyecto	200 h.
Diseño de los elementos 3D	180 h.
Implementación	300 h.
Pruebas	135 h.
Total	**855 h.**

Tabla 7.1: Fases del proyecto

7. PRESUPUESTO



Id.	Nombre de tarea	Comienzo	Fin	Duración
1	Análisis y diseño	10/10/2008	23/10/2008	10d
2	Diseño de elementos 3D	20/10/2008	19/12/2008	45d
3	Implementación	22/12/2008	13/03/2009	60d
4	Pruebas	16/03/2009	15/05/2009	45d
5	Documentación del proyecto	30/03/2009	05/06/2009	50d

Id.	Nombre de tarea	Comienzo	Fin	Duración
1	Análisis y diseño	10/10/2008	23/10/2008	10d
2	Diseño de elementos 3D	20/10/2008	19/12/2008	45d
3	Implementación	22/12/2008	13/03/2009	60d
4	Pruebas	16/03/2009	15/05/2009	45d
5	Documentación del proyecto	30/03/2009	05/06/2009	50d

Figura 7.1: Diagrama de gant

Cargo	Sueldo neto	Sueldo bruto		Coste/Hora
Analista	1.552,00 €/mes	2000 €/mes	28.000 €/año	25 €
Responsable de documentacion	993,28 €/mes	1280 €/mes	17.920 €/año	16 €
Diseñador 3D	1.055,36 €/mes	1360 €/mes	19.040 €/año	17 €
Programador	1.164,00 €/mes	1500 €/mes	21.000 €/año	15 €
Responsable de pruebas	698,40 €/mes	900 €/mes	12.600 €/año	15 €

Tabla 7.2: Fases del proyecto

7.2 Salarios por categoría

En la tabla 7.2 se detalla el coste de cada uno de los roles precisados para la elaboración del proyecto. Cada rol debe ser adoptado por personal informático cualificado.

- Coste/Hora indica el sueldo bruto en una hora de trabajo.
- Sueldo bruto indica el sueldo bruto anual, con 14 pagas mensuales.
- Sueldo neto indica el sueldo neto mensual. Descontamos el Impuesto sobre la Renta de Personas Físicas (20%) y la Seguridad Social (2,4%)

En el cálculo del salario hay que tener en cuenta que:

- Los días laborables al mes son 20.
- El analista, responsable de documentación y diseñador 3D trabajan 4 horas al día. El programador trabaja 5 horas al día y el responsable de pruebas trabaja 3 horas al día.

7.3 Gastos de personal imputables al proyecto

Este proyecto ha sido realizado por un informático, el cual adoptó roles distintos para desarrollar cada unas de las actividades que forman el proyecto. La tabla 7.3 muestra el coste total de cada rol.

Cargo	Horas	Coste/hora	Total
Analista	40 h.	25 €	1.000 €
Responsable de documentación	200 h.	16 €	3.200 €
Diseñador 3D	180 h.	17 €	3.060 €
Programador	300 h.	15 €	4.500 €
Responsable de pruebas	135 h.	15 €	2.025 €
Total	**855 h.**		**13.785 €**

Tabla 7.3: Gastos de personal

Recurso	Cantidad	Coste Total
Ordenador personal	5	3.640,00 €
Router	1	70,00 €
Cable de red	7	14,98 €
Visual Studio 2005	1	679,00 €
Microsoft Office 2007	1	541,49 €
Total		**4.945,47 €**

Tabla 7.4: Recursos materiales

7.4 Recursos materiales empleados

En la tabla 7.4 se detallan los costes de los recursos necesarios para la elaboración del proyecto. Los costes incluyen el Impuesto al Valor Añadido (IVA incluido).

7.5 Gastos indirectos

La tabla 7.5 muestra los gastos indirectos en el desarrollo del proyecto.

7.6 Resumen del presupuesto

La tabla 7.6 muestra un resumen de todos los costes que ha requerido el proyecto así como la suma total de los mismos.

Al coste calculado hay que añadir un margen de imprevistos del 10%: **2.010,9 €**

Coste total + margen de imprevistos: 20.108,97 + 2.010,9 = **22.119,87 €**

Recurso	Coste Total
Servicio de limpieza	Incluido en los costes indirectos
Electricidad	Incluido en los costes indirectos
Agua	Incluido en los costes indirectos
Calefacción	Incluido en los costes indirectos
Línea telefónica e internet	Incluido en los costes indirectos
Alquiler del local	Incluido en los costes indirectos
Amortización inmobiliario	Incluido en los costes indirectos
Gastos de la comunidad	Incluido en los costes indirectos
Costes indirectos (10%)	**13785 * 0,10 = 1378,5€**

Tabla 7.5: Gastos indirectos

Recurso	Coste
Personal con cargo al proyecto	13.785 €
Recursos materiales empleados	4.945,47 €
Gastos indirectos	1.378,5 €
Total	**20.108,97 €**

Tabla 7.6: Resumen del presupuesto

Finalmente se calculan los beneficios a obtener con el proyecto, un 15% sobre la suma del coste total y el margen de beneficios: **3.317,98 €**

Coste total + margen de imprevistos: 20.108,97 + 2.010,9 + 3.317,98 = **25.437,85 €**

El presupuesto total del proyecto realizado es de **25.437,85 €(veinticinco mil cuatrocientos treintaisiete con ochentaicinco euros)**, Impuesto sobre el Valor Añadido incluido.

8

Conclusiones y trabajo futuro

8.1 Conclusiones

Se han abordado todas las fases que generalmente constituyen el desarrollo e implementación de un juego.

Sobre el desarrollo de los elementos 3D puede decirse que es una tarea extensa, que requiere de un proceso continuo de prueba para comprobar la calidad del material producido. Los elementos 3D constituyen la principal atracción del juego y por ello debe cuidarse su desarrollo.

El desarrollo de la lógica ha resultado interesante, ya que la forma de controlar las reglas desde la parte del servidor ha sido distinta al control realizado desde la parte cliente. La parte del servidor se encarga de controlar las reglas del partido tal como el número de turno, las acciones que han realizado los jugadores, la resolución de tiradas de dados, etc. La parte cliente se encarga de controlar que los elementos gráficos se muestran adecuadamente, que las animaciones se muestran en el orden que son notificadas, de la activación y desactivación de elementos de la interfaz, etc.

El uso de .NET Remoting en las comunicaciones ha facilitado en gran medida la implementación de las mismas. El que existieran dos roles de comunicación bien definidos junto con la existencia de turnos para realizar las acciones, también han sido factores que han ayudado a simplificar la implementación de las comunicaciones. Existen otros tipos de juego en el cual el desarrollo de las comunicaciones es un proceso laborioso y complejo, motivo por el cual se ha querido hacer hincapié que este juego no ha sido el caso.

8.2 Trabajo futuro

Un aspecto muy importante para el futuro sería la mejora de la seguridad en las comunicaciones. Actualmente un usuario logueado se identifica mediante un número de sesión; ese número se transmite en claro en las comunicaciones entre el cliente y el servidor. Para un atacante sería fácil copiar dicho número y utilizarlo como número de sesión propio, suplantando al usuario.

Otro aspecto interesante sería dotar al servidor de un mecanismo de autenticación, de esta manera los clientes podrían comprobar que se están conectando realmente al servidor de la aplicación.

También puede aumentarse las características del partido. Existen reglas y habilidades del reglamento que no se han aplicado al juego, por lo tanto una línea futura sería incluir nueva funcionalidad al partido.

Por último puede ampliarse la funcionalidad del chat de la aplicación. Posibles nuevas características de la aplicación podrían ser la posibilidad de realizar conversaciones privadas entre usuarios logueados, el uso de imágenes para identificar al usuario y la posibilidad de utilizar colores e iconos en la ventana de chat.

Referencias y bibliografía

[1] Junker, Gregory *Pro OGRE 3D Programming*, Apress.

77, 98, 101, 171

[2] Reglamento BloodBowl, *http://www.games-workshop.com/ MEDIA_CustomProductCatalog/ m2520131_Reglamento_Blood_Bowl.zip*, GamesWorkshop.

170

[3] Tutorial sobre diseño 3D con Blender en Blender 3D Club (Inglés), *http://www.blender3dclub.com/*

[4] Tutorial sobre diseño 3D con Blender (Inglés), *http://en.wikibooks.org/wiki/ Blender_3D:_Noob_to_Pro*

[5] Manual de Blender, *http://wiki.blender.org/index.php/Doc:ES/Manual*, Blender Foundation.