Universidad de las Ciencias Informáticas
Vertex, Entornos Interactivos 3D, Facultad 5

Componente para la interconexión online del videojuego de navegador multijugador Mundo Verde

Trabajo de diploma para optar por el título de Ingeniero en Ciencias Informáticas

Autor: José Manolo Díaz Aguila

Tutores: Ing. Reinaldo Garcia Maturell

Ing. Midiala Baños Artigas

La Habana, 2016

El individuo ha luchado siempre para no ser absorbido por la tribu.
Si lo intentas, a menudo estarás solo, y a veces asustado.
Pero ningún precio es demasiado alto por el privilegio de ser uno mismo.
Friedrich Nietzsche.

Dedicatoria

Dedico esta tesis a mis padres que son la fuente de mi inspiración, por ellos hoy soy lo que soy y por ellos trataré de ser cada día mejor, en especial a mi madre que ha sido para mí la mejor del mundo.

Agradecimientos

Le agradezco a toda mi familia, a mis padres, hermanos, mi abuelita linda como siempre y mi sobrinito Yoan Carlos por estar siempre ahí para mí en todo momento, por poder confiar en ellos y pedir su ayuda incondicional. A Yani y Yessica que se comprometieron en ayudarme y así lo hicieron a mis tutores por toda la ayuda durante el proceso de desarrollo de este trabajo y muy muy especial a mi novia Midiala que sin ella hoy no estuviera escribiendo estas letras por estar ahí a mi lado siendo tutora y novia al mismo tiempo por aguantarme en los momentos más difíciles anímicamente y por guiarme tanto profesional como emocional en el desarrollo de este trabajo. También agradecer a todos mis amigos, a todos los que he conocido durante mi carrera como estudiante, muy especial a mis compañeros de cuarto que hoy me entristece saber que ya no veré algunos de ellos, pero donde quiera que estén les desearé siempre lo mejor.

Declaración de autoría

Declaramos ser autores de la presente tesis y reconocemos a la Universidad de las Ciencias Informáticas los derechos patrimoniales sobre esta, con carácter exclusivo.

Para que así conste firmamos la presente a los ____ días del mes de _____ del año _____.

José Manolo Díaz Aguila
Autor

Ing. Reinaldo Garcia Maturell
Tutor

Ing. Midiala Baños Artigas
Tutor

Resumen

El Centro de Entornos Interactivos 3D (VERTEX) de la Universidad de las Ciencias Informáticas ha desarrollado un grupo de videojuegos serios de navegador, los cuales son gestionados por una Plataforma de gestión de videojuegos de navegador. Dentro de los videojuegos que debe gestionar dicha plataforma se encuentra Mundo Verde, el cual está concebido para que interactúen 2 ó 3 jugadores al mismo tiempo en una misma estación de trabajo. A pesar de ser un videojuego de navegador multijugador que permite la interacción de más de dos jugadores, el juego posee limitaciones, pues no permite que los jugadores puedan interactuar desde diferentes estaciones de trabajo, ubicadas en distintos puntos de la red. Partiendo de esta situación, el presente trabajo se plantea como objetivo desarrollar un componente que permita la intercomunicación *online* entre jugadores para el videojuego de navegador Mundo Verde. Para el desarrollo de dicho componente se seleccionó *Node.js* para la gestión del servidor y *Websocket* como protocolo de comunicación para establecer el canal entre el servidor y cliente. Además, se seleccionó *Mysql* para la persistencia de los datos y para regir el proceso de desarrollo del proyecto en cuestión, se seleccionó la metodología ágil Programación Extrema. Al concluir el trabajo el componente desarrollado permite que los clientes puedan interactuar en tiempo real desde distintos lugares en una misma partida en el videojuego. Además, se establece la integración del videojuego con la Plataforma.

Palabras clave: componente, protocolos de comunicación, videojuego de navegador multijugador.

Índice general

Índice de figuras

Índice de tablas

Lista de códigos fuentes

Introducción

Los videojuegos son juegos electrónicos, dirigidos principalmente al entretenimiento, pero que también pueden ser aprovechados para otros fines como la educación y la medicina. En ellos pueden interactuar una o más personas por medio de un controlador, con un dispositivo electrónico dotado de imágenes de video. Este dispositivo puede ser una computadora, una máquina *arcade*, una videoconsola o un dispositivo portátil como un teléfono móvil. Los videojuegos se dividen en dos corrientes principales: videojuegos para un solo jugador y videojuegos para varios jugadores o multijugador como también se les conoce. Estos últimos son aquellos que permiten la interacción de dos o más jugadores al mismo tiempo, ya sea de manera física en una misma consola o mediante el uso de servicios de red con usuarios conectados a esta. Dicha modalidad suele ser en tiempo real o por turnos y surge con los propios videojuegos debido, sobre todo, a la complejidad de crear oponentes virtuales.

Cuba en los últimos años ha estado inmersa en el desarrollo de software para informatizar la sociedad. En aras de apoyar este proceso el Centro de Entornos Interactivos 3D (VERTEX) de la Universidad de las Ciencias Informáticas (UCI), desarrolla un grupo de videojuegos serios de navegador. Los videojuegos serios son aquellos que no son solo para el entretenimiento, sino que además, tienen propósitos educacionales. Dichos videojuegos son controlados por una Plataforma de Gestión de Videojuegos también desarrollada en el centro. La plataforma permite que los usuarios puedan conectarse desde cualquier lugar del país y acceder a sus servicios.

Entre los videojuegos que debe gestionar dicha plataforma se encuentra el videojuego de tablero Mundo Verde, en el cual un usuario a medida que avanza en el juego adquiere conocimiento de las especies invasoras que existen en Cuba. El juego está concebido para que interactúen dos o tres jugadores en la misma estación de trabajo y de ellos gana aquel que llegue primero a la meta; sin embargo, no permite que los jugadores se puedan conectar desde diferentes estaciones de trabajo. Debido a esto se hace necesario establecer una vía que permita la comunicación entre varios jugadores del videojuego desde varios puntos de red, lo que permite ampliar y compartir su experiencia de juego.

Teniendo en cuenta la situación antes descrita se plantea como **problema de investigación**: ¿Cómo lograr la interconexión entre los jugadores del videojuego de navegador Mundo Verde?

Se define como **objeto de estudio** los videojuegos de navegador multijugador y como **campo de acción** los protocolos de comunicación para videojuegos de navegador multijugador.

Para dar solución al problema de investigación de la presente investigación se plantea el siguiente **obje-**

tivo: Desarrollar un componente para la interconexión *online* del videojuego de navegador Mundo Verde.

Las **tareas a desarrollar** para asegurar el cumplimiento del objetivo trazado son:

- Identificación de términos relacionados con la investigación.
- Caracterización de la Plataforma de Gestión de Videojuegos de Navegador.
- Caracterización de los videojuegos multijugador masivos en red, haciendo énfasis en los videojuegos de navegador.
- Caracterización del videojuego de navegador Mundo Verde.
- Investigación y análisis de los protocolos de comunicación para videojuegos de navegador multijugador.
- Selección de las tecnologías que serán empleadas en el proceso de desarrollo del componente.
- Definición de requisitos funcionales.
- Diseño e implementación del componente de intercomunicación *online* del videojuego de navegador Mundo Verde.
- Integración del videojuego con la plataforma.
- Realización de las pruebas de rendimiento y calidad a las principales funcionalidades del software.

Para el correcto desarrollo de la investigación, se emplean los siguientes **métodos científicos**.

Métodos teóricos:

- Método analítico-sintético: para analizar desde diferentes aristas los conceptos asociados a la investigación y sintetizar la información recopilada, permitiendo describir las características generales y las relaciones esenciales entre estos.
- Método de modelación: para crear abstracciones con el objetivo de explicar la realidad, se utiliza para la modelación de los diversos diagramas necesarios en cada uno de los flujos de trabajo según la metodología seleccionada.

Métodos empíricos:

- Observación científica: se emplea con el objetivo de observar el comportamiento de diferentes tecnologías que puedan ser seleccionadas para el desarrollo de la solución del problema de investigación.

CAPÍTULO 1

Fundamentación Teórica

Introducción

En este capítulo se abordarán algunos conceptos relacionados con los videojuegos, en especial los videojuegos de navegador multijugador que se desarrollan de manera *online*. Al mismo tiempo se realizará una caracterización del videojuego Mundo Verde que brinde información acerca de su estructura, con el fin de tener una mayor interpretación del juego en cuanto a funcionamiento lógico. Además, se realizará un estudio acerca de las principales tecnologías para el desarrollo de una posible solución al problema en cuestión, así como la metodología que regirá todo el proceso de desarrollo del proyecto.

1.1. Términos relacionados con la investigación

Para entender mejor los temas que serán abordados en la investigación, con el objetivo de elaborar una propuesta de solución al problema planteado, se hace necesario mencionar algunos conceptos asociados al dominio del problema.

Componente

El término componente se ha utilizado en repetidas ocasiones, a pesar de que es difícil de efectuar una descripción definitiva del término. Sin embargo, cuando se menciona en el ámbito de la informática, se refiere a una parte reemplazable, casi independiente y no trivial de un sistema, que cumple una función clara en el contexto de una arquitectura bien definida (PRESSMAN, 2005).

Un componente de software puede ser desplegado sin que dependa de otras partes del sistema, así como participar en sus composiciones. De igual forma debe ofrecer un servicio predefinido y debe ser capaz de comunicarse con otros componentes. Su capacidad de ser reutilizado, es una característica importante de los componentes de software de alta calidad. Un componente debe ser diseñado e implementado de tal forma que pueda ser reutilizado en otros programas, aun cuando estos sean diferentes para el cual fue diseñado. Es necesario que se documente todo lo referente al diseño e implementación, además de que una vez desarrollado

debe ser probado varias veces para que no existan errores en su integración o su reutilización, comprobando así que el componente es robusto.

Interconexión

En telecomunicaciones, la interconexión es la vinculación de recursos físicos y soportes lógicos, incluidas las instalaciones esenciales necesarias, para permitir el inter-funcionamiento de las redes y la interoperabilidad de servicios de telecomunicaciones. Sin embargo, cuando se habla de seguridad informática, una interconexión es una comunicación efectuada entre dos o más puntos, con el objetivo de crear una unión entre ambos, sea temporal o no, para efectuar una transmisión puntual o fija de forma *online*, comunicando dos máquinas de forma permanente (ALEGSA, 2008).

1.2. Videojuegos multijugador masivos en red

Los videojuegos multijugador masivos en red o *Massively Multiplayer Online Games* (MMOG, por sus siglas en inglés) es un estilo de videojuegos, que brinda a los usuarios un enfoque diferente a los videojuegos para un solo jugador. Debido principalmente a que permiten la interacción con otros jugadores y de este modo pueden medir su experiencia contra personajes reales. Se basan de forma general en la arquitectura cliente-servidor, por lo que existe un constante flujo de datos entre ambos. Esto conlleva a que los usuarios exijan un alto grado de rendimiento enfocado en conseguir tiempos de respuesta suficientemente cortos. Los videojuegos de tipo MMOG se caracterizan por ofrecer al jugador la posibilidad de participar e interactuar con un gran número de jugadores simultáneamente, conectados a través de la red. Se encuentran en su mayoría desarrollados para videojuegos de consola, que permiten calidad de visualización de los gráficos del juego y además una fácil interacción del usuario con el videojuego. Convirtiéndolos de este modo en una de las principales elecciones de los amantes a los videojuegos (VILARDELL, 2012).

Los MMOG se pueden definir como "Videojuegos que se juegan en línea y que permiten a los usuarios crear su propio personaje o *avatar*, para interactuar no solo con el entorno diseñado y los jugadores controlados por el propio videojuego, sino además que permiten la interacción con otros personajes controlados por otros usuarios desde distintas partes del mundo"(STEINKUEHLER, 2004). En otras palabras, no son más que un videojuego que permite la comunicación en tiempo real y el intercambio de experiencia de juego entre varios usuarios mediante el uso de la red.

Aun teniendo una definición de los MMOG, existen características diferenciadoras que hacen que, dentro de ellos, se distingan tres tipos por encima de los demás.

- Juegos de disparos multijugador masivos en línea en primera persona o *Massively Multiplayer Online First Person Shooter Games* (MMOFPSG, por sus siglas en inglés): En este tipo de videojuego se representan escenas donde se lleva a cabo combates entre los usuarios utilizando un arma de fuego. El personaje que representa al usuario es una silueta o un arma. Esta representación da la sensación o

idea al usuario, de que está dentro del mundo virtual en un combate armado (CAMACHO, 2015). Los jugadores de este tipo de videojuegos buscan una distracción momentánea, de modo que no suelen ser jugadores fieles a un juego en concreto. Ejemplo de estos juegos son el *Call of Duty*, *Counter Strike*, *Urban Terror*, entre otros (VILARDELL, 2012).

- Juegos multijugador masivos en línea de interpretación de roles o *Massively Multiplayer Online Role Playing Games* (MMORPG, por sus siglas en inglés): Son juegos que permiten a miles de jugadores conectarse a un mismo mundo virtual de forma simultánea a través de Internet e interactuar entre ellos. Consisten en la creación de un personaje, del que el jugador puede elegir raza, profesión, posesiones, entre otras opciones. Una vez creado el personaje, el jugador puede ir subiendo su nivel mediante batallas contra otros personajes (jugadores controlados por otros usuarios o por el propio juego) o realizando diversas aventuras o misiones (CAMACHO, 2015). La mayoría de los MMORPG comparten varias características: juego de tipo rol, que incluye misiones y sistemas de recompensa en forma de tesoros; estadísticas de cada jugador; un mundo situado con ambientación medieval con personajes; misiones e historia principal acorde al estilo medieval y organización de los jugadores en clanes (VILARDELL, 2012).
- Juegos de estrategia multijugador masivos en línea en tiempo real o *Massively Multiplayer Online Real Time Estrategy Games* (MMORTEG, por sus siglas en inglés): Se trata de videojuegos en los que el jugador debe desempeñar el papel de un líder como un rey o un caudillo, en una ambientación fantástica o en una ambientación de ciencia ficción. Esto supone de ocuparse de asuntos como la gestión de recursos económicos, la diplomacia o la formación de un ejército. El juego se desarrolla en un mundo persistente que evoluciona independientemente de que los usuarios estén o no conectados (ibíd.).

El estudio de los MMOG permite apreciar el avance que estos aportan en el contexto de los videojuegos en cuanto a experiencia de juego, debido a que permiten el intercambio y la interacción en tiempo real entre jugadores situados en diferentes partes del mundo, a través del uso de la red. Generalmente estos videojuegos se encuentran desarrollados para videoconsolas, lo que hace que se encuentren en un mismo videojuego dos aspectos muy importantes: la calidad de visualización de los gráficos y la fácil interacción con el juego, sumado a esto la competitividad con personas reales, lo que conlleva a la creación de videoconsolas con mejores prestaciones y por lo tanto más costosas. Sin embargo, existen personas que no están dispuestas o no cuentan con las condiciones económicas para comprar una videoconsola o el propio videojuego por el simple hecho de jugar. A raíz de esto emergen los videojuegos de navegador, los cuales brindan a los usuarios la posibilidad de jugar sin la necesidad de tener que comprar o instalar nada. Simplemente se necesita un navegador web y conexión a la red.

1.2.1. Videojuegos de navegador

Hoy en día es muy común acceder a una página en Internet y encontrar publicidad de juegos tal como “Regístrate y juega”, que venden su producto con un “No necesitas instalar nada”. En los últimos años el

mercado de los videojuegos de navegador se ha incrementado de manera acelerada. Suelen ser por lo general juegos de estrategia y multijugador, ambos a la vez. Los videojuegos multijugador *online* de navegador, poseen una serie de características que los hacen competitivos frente a los videojuegos de consola o de estaciones de trabajo, pese a que son juegos menos atractivos en términos gráficos o de interacción de juego (VANHATUPA, 2010). Algunas de estas características son:

- Se compite contra otros jugadores de todo el planeta en tiempo real.
- No necesitan ningún tipo de instalación, tan sólo conexión a Internet y un navegador *web*.
- Son juegos gratuitos, aunque bien es cierto que la mayoría suelen poseer formas de pago para facilitar mejoras individuales o ahorro de tiempo.
- Pueden ser jugados por personas de todo el mundo. En general se encuentran en inglés, español o traducidos a varios idiomas.

Un videojuego de navegador multijugador no es más que un videojuego que permite la interacción de varios jugadores al mismo tiempo desde distintas partes del mundo mediante el uso de la red y un navegador *web* que brinda la posibilidad de no tener que instalar nada. Aunque algunos poseen paquetes gráficos para mejorar la visualización del juego (VANHATUPA, 2013).

Los videojuegos de navegador multijugador generalmente son buenos en la creación de grandes mundos con una gran cantidad de jugadores, lo que explica el gran número de juegos de estrategia y juegos de rol que existen. Los juegos de estrategia enfatizan las habilidades de toma de decisiones del jugador. Por lo general hay un mundo grande con una gran cantidad de jugadores donde estos pueden crear alianzas, donde el jugador tiene que luchar por abrirse camino a la cima. Un ejemplo de este tipo de juego es *Travian*, donde en un inicio el jugador recibe un pueblo para construir una nación poderosa. Los jugadores pueden formar alianzas y guerras salariales.

Por otra parte, los juegos de rol se concentran en el desarrollo del personaje y de un rol. El jugador controla sólo un personaje en lugar de una nación o grupo. Un ejemplo de los juegos de rol es *Forumwarz*, un juego de rol que trata sobre la parodia que se lleva sobre Internet.

Debido al auge alcanzado por los juegos de rol y estrategia en navegadores web. El desarrollo de videojuegos de navegador multijugador se encuentra enfocado principalmente en la creación de un servidor, que maneja las conexiones de miles de usuarios simultáneamente, por lo que uno de los renglones a medir en su desarrollo son los tiempos de respuesta entre el cliente y el servidor. Es por eso que se hace necesario en su creación, la selección de un buen protocolo de comunicación entre ambos enfocado en alcanzar la interacción de los jugadores en tiempo real.

1.3. Protocolos de comunicación para videojuegos de navegador multijugador

Un protocolo de comunicación es una serie de convenciones y reglas que establecen cómo deben comunicarse las computadoras a través de la red. Para que las computadoras puedan funcionar en redes necesitan de estos protocolos, los cuales transmiten la información fragmentada (TANENBAUM, 2003). Un protocolo de comunicación describe los siguientes aspectos:

- El tiempo relativo al intercambio de mensajes entre dos sistemas de comunicación.
- El formato que el mensaje debe contener para que el intercambio entre dos computadoras que emplean protocolos diferentes, puedan establecer comunicación.
- Las acciones que deben realizarse en el caso de producirse un error en la comunicación.
- Las suposiciones acerca del entorno en el cual se ejecutará el protocolo.

Los protocolos de comunicación son programas que se ejecutan tanto en la computadora de origen como en la de destino. Estos programas añaden una serie de datos de control a la información que se pretende trasmitir. Los datos de control son agregados por el transmisor y suprimidos por el receptor antes que la información llegue al usuario (VALDÉS, 2012).

Dentro de los protocolos de comunicación más utilizados para videojuegos de navegador multijugador se encuentran los siguientes:

Long Polling

Esta técnica se basa en el Protocolo de Transferencia de Hipertexto o *Hypertext Transfer Protocol* (HTTP, por sus siglas en inglés). Primero el cliente abre una solicitud pendiente con el servidor, el servidor solo responde a esta petición cuando tiene datos que enviar al cliente y tan pronto llegue la respuesta, el cliente abrirá una nueva solicitud pendiente con el servidor. Cabe destacar como ventaja de HTTP, que se encarga de todos los problemas de topologías de red, como son el *proxy* o *firewall*. Otra ventaja es que hay una gran variedad de opciones disponibles para implementar el servidor de juego. El principal inconveniente de este método es cuando llega al rendimiento, cada petición por parte del cliente establece una nueva conexión subyacente del protocolo *Transmission Control Protocol* (TCP) lo que aumenta los tiempos de envío y respuesta. Otro aspecto a tener en cuenta es el tamaño de los grandes mensajes debido a propias cabeceras del protocolo, aumentando el ancho de banda y el procesamiento en el cliente y el servidor (PIMENTEL y NICKERSON, 2012).

WebSocket

Ante las nuevas necesidades de los usuarios de la *web* en lograr tiempo real en la actualización de la información, surge el protocolo de comunicación *WebSocket*. Este protocolo a diferencia de su precedente

HTTP no sigue un esquema solicitud-respuesta, garantizando una comunicación *full-duplex* (bidireccional) entre el cliente y el servidor. *WebSocket* hoy día constituye un cambio de paradigma para el desarrollo de aplicaciones *web* estacionarias y móviles. La forma de comunicación entre el cliente y el servidor sin una previa solicitud del usuario hace que la información sea inmediata logrando tiempos reales en la *web* (PUJOL, 2012).

Es una tecnología que proporciona un canal de comunicación *full-duplex* sobre un único punto de conexión TCP. Está diseñada para ser implementada en navegadores y servidores web, pero puede utilizarse en cualquier aplicación cliente-servidor. Como las conexiones TCP ordinarias sobre puertos distintos al 80 por lo general son bloqueadas por los administradores de redes, el uso de esta tecnología proporciona una solución a este tipo de limitaciones, ya que provee una funcionalidad similar a la apertura de varias conexiones en distintos puertos, pero multiplexando diferentes servicios *WebSocket* sobre un único puerto TCP (CASTRELO CID, 2014). Además, es el protocolo idóneo para aplicaciones *web* donde el tiempo de respuesta de los mensajes entre el cliente y el servidor sea lo primordial, dígase *chat* y juegos multijugador en tiempo real.

El estudio de los protocolos de comunicación quizás sea la parte más importante en el desarrollo de un sistema informático que se ejecute en tiempo real, independiente a la cantidad de personas que puedan interactuar o de la técnica que se seleccione y del modo en que opere esta, si no se define un protocolo de comunicación, no va existir intercambio de mensajes entre el cliente y servidor y por tanto, los usuarios no podrán interactuar en un mismo sistema desde diferentes lugares al mismo tiempo.

1.4. Plataforma de Gestión de Videojuegos de Navegador

Es una plataforma *web* construida en el Centro de Entornos Interactivos 3D Vertex de la UCI enfocada actualmente en el objetivo de facilitar la gestión de videojuegos terapéuticos, los cuales son asignados a un paciente para su rehabilitación. Esta herramienta fue desarrollada haciendo uso del lenguaje de programación *PHP* y se utilizó como marco de trabajo *Yii*; mientras que la gestión de los datos es administrada utilizando el Sistema Gestor de Bases de Datos (SGBD) *MySQL* y como servidor *Apache Server* (GUTIERREZ, 2015). Dentro de las principales funcionalidades que brinda la plataforma se encuentra:

- **Gestión de videojuegos**: Esta funcionalidad se encarga de la integración, actualización y eliminación de videojuegos en la plataforma, así como la sincronización de los datos de los videojuegos. Controla la posibilidad de descargar o no los videojuegos y permite ejecutarlos directamente en el navegador. Posibilita cargar al servidor videojuegos que estén compactados en formato *.zip*.
- **Generar estadísticas de los videojuegos**: Esta funcionalidad se encarga de la generación de las estadísticas grupales e individuales de la aplicación. A partir de los datos almacenados en el sistema, genera un grupo de estadísticas grupales e individuales asociadas a los videojuegos y a los pacientes asociados a estos.

- **Gestionar tratamientos**: Esta funcionalidad permite que los doctores puedan crear y actualizar tratamientos en el sistema. Además, permite asignar uno o varios tratamientos a los pacientes asignados a un doctor.
- **Gestionar usuarios**: Esta funcionalidad se encarga de la creación, actualización y eliminación de usuarios en el sistema. Además, es responsable del proceso de asignación de roles y permisos en la aplicación.

Proceso de integración de videojuegos

La funcionalidad principal de la plataforma es la gestión de videojuegos. Para llevar a cabo este proceso de manera organizada y correcta, se establecen una serie de requisitos o restricciones. Los videojuegos que se necesiten integrar deben estar comprimidos en formato *.zip*. Dentro del comprimido debe aparecer una carpeta con el nombre del videojuego. En su interior deben aparecer, de manera obligatoria, una carpeta llamada *saves* que contenga un fichero *.json* donde se almacene la información de salida del videojuego y un fichero que sirva de ejecutable principal (*index.html*). En el interior de la carpeta también deben estar los directorios y ficheros asociados al funcionamiento del videojuego. A partir de las variables almacenadas dentro del archivo *.json* la plataforma es capaz de generar una serie de estadísticas de la evolución del paciente (ibíd.).

Como se puede apreciar la plataforma actualmente es capaz de integrar cualquier tipo de videojuegos que cumplan con las restricciones establecidas. Sin embargo, no está preparada para manejar estadísticas relacionadas con videojuegos que no se utilicen para rehabilitaciones de los pacientes y menos para videojuegos multijugador *online*. Debido a esto no provee un servidor para este tipo de videojuegos, por lo que a la hora de integrar un videojuego de este tipo se hace necesario crear un servidor independiente. En relación a estas debilidades la plataforma se encuentra en proceso de desarrollo con el fin de enfocarla específicamente en la gestión de videojuegos serios de navegador, desarrollados en el centro, brindando de este modo la posibilidad de que todos los videojuegos estén disponibles en un mismo lugar y facilitar el control sobre estos. Entre los videojuegos serios que debe gestionar la plataforma se encuentra Mundo Verde, del cual se realizó un análisis para comprender su funcionamiento.

1.5. Videojuego Mundo Verde

Mundo Verde es un videojuego de navegador desarrollado sobre el motor de videojuegos *Phaser.io* empleando el lenguaje de programación *JavaScrip*. Está concebido para que puedan interactuar dos o tres jugadores al mismo tiempo en una misma estación cliente. Durante el juego los usuarios obtendrán conocimiento acerca de las especies invasoras que existen en Cuba y de ellos gana el primero en llegar a la meta. El videojuego funciona en navegadores web como *Mozilla Firefox* o *Internet Explorer*. A continuación, se muestra la estructura lógica y visual del videojuego.

1.5.1. Estructura Lógica

El juego cuenta con un archivo *index.html* que funciona como el ejecutable principal donde se inicializa el *framework Phaser.io* y se adicionan todos los estados del juego, siendo los más importantes el *Boot*, *Preloader*, *MainMenu*, *PlayerSelect*, *Game* y *Game3*, y que además se inicializa el estado *Boot*. A continuación, se ofrece una breve explicación de cada uno.

- *Boot*: Una vez que se carga este estado se crea la variable principal de juego y se le da paso al estado *Preloader*.
- *Preloader*: Es el estado encargado de añadir el audio y todas las imágenes que serán utilizadas en el juego, además de los archivos externos que serán utilizados, dando paso al estado *MainMenu*.
- *MainMenu*: Es el estado encargado de brindarle la posibilidad al usuario de seleccionar o no el audio, de ver los créditos e iniciar la partida, dando paso al estado *PlayerSelect*.
- *PlayerSelect*: Es el estado encargado de brindar la posibilidad de que el usuario escoja la cantidad de jugadores con los que desea jugar, como está concebido en el juego. Luego pasará al estado *Game* en caso que se escoja la opción de 2 jugadores o *Game3* en el caso de 3 jugadores.
- *Game*: En este estado se ejecutan las funcionalidades que conforman la lógica principal del juego, permitiendo la dinámica del mismo para el caso de 2 jugadores.
- *Game3*: En este estado se ejecutan las funcionalidades que conforman la lógica principal del juego, permitiendo la dinámica del mismo para el caso de 3 jugadores.

1.5.2. Estructura Visual

El juego está compuesto por 2 ó 3 personajes en dependencia de la cantidad de jugadores que se escoja justo antes de comenzar a jugar, identificados por los colores rojo, amarillo y azul. Estos se desplazan por un tablero en forma de ***"S"*** dividido en 34 casillas que representan un camino por la fauna cubana. Lo jugadores se desplazarán por las casillas en dependencia del valor del dado que aparece al dar clic en el botón "Tirar". Las casillas del tablero son de 5 tipos diferentes, exceptuando sólo las casillas en blanco, al caer en una de ellas aparecerán cartas que brindan información a los jugadores.

- Casillas en blanco: Este tipo de casilla es también conocida como casilla pasiva, debido a que, una vez que los jugadores estén en ellas, se quedan esperando su próximo turno.
- Casillas de avanzar: Cada una de estas casillas posee una casilla en blanco asignada para avanzar desde su posición.
- Casillas de retroceder: Cada una de estas casillas posee una casilla en blanco asignada para retroceder desde su posición.
- Casillas desconocidas: En estas casillas aparece una carta aleatoria que, según su tipo, el jugador avanzará o retrocederá desde su posición.
- Casillas sorpresa: En estas casillas aparece una carta aleatoria que, según su tipo, el jugador avanzará o retrocederá desde su posición.

Figura 1.1. Menú de inicio del videojuego Mundo Verde.

Figura 1.2. Opción de seleccionar la cantidad de jugadores.

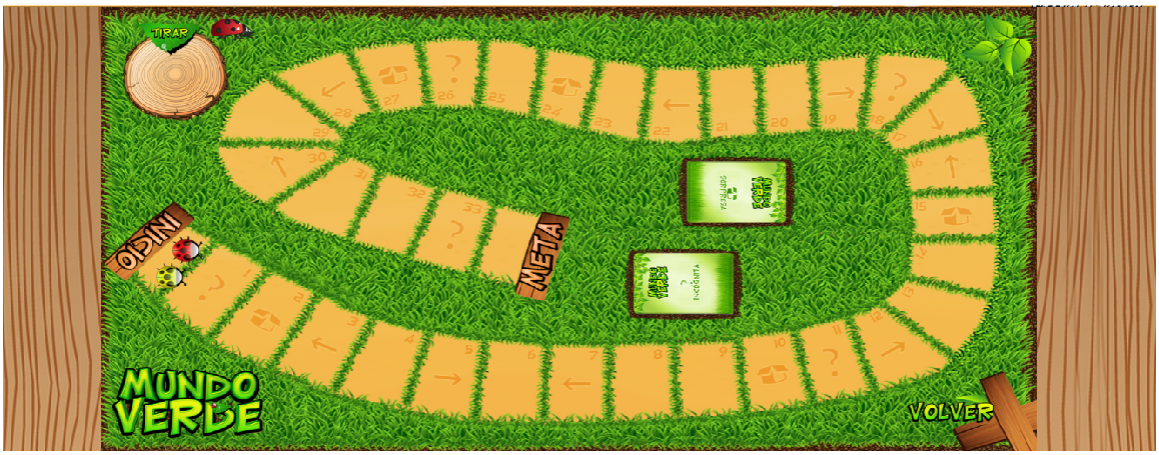

Figura 1.3. Dinámica del videojuego para 2 jugadores.

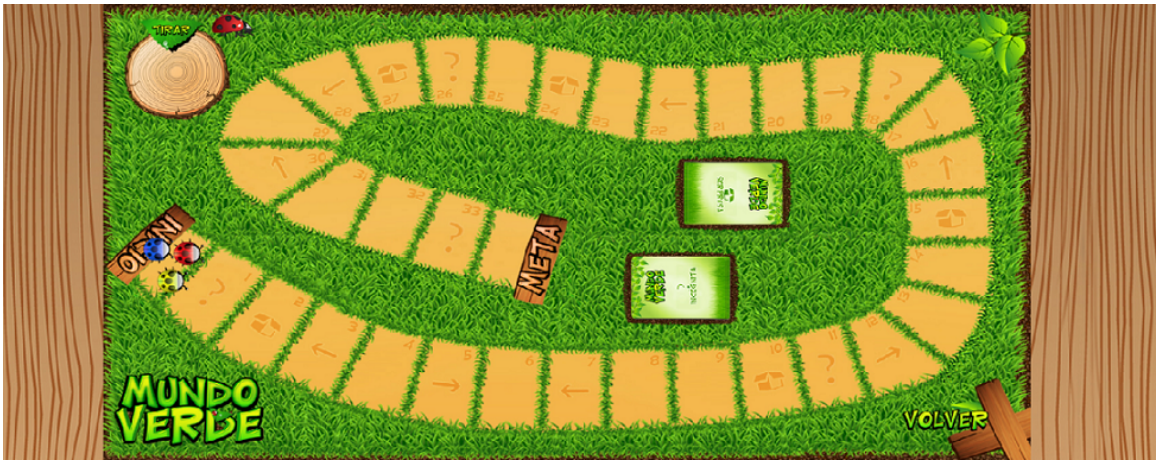

Figura 1.4. Dinámica del videojuego para 3 jugadores.

Figura 1.5. Información que muestra el videojuego cuando un usuario cae en una casilla determinada.

1.6. Tecnologías del lado Servidor

Atendiendo a la necesidad de introducir el carácter multijugador en el videojuego Mundo Verde, se hace necesario realizar un estudio y selección de las tecnologías a utilizar para dar solución al problema de la presente investigación.

Un servidor *web* es un programa que procesa cualquier aplicación realizando conexiones bidireccionales y/o unidireccionales, síncronas o asíncronas con el cliente, generando o cediendo una respuesta en cualquier lenguaje o aplicación del lado del cliente (GREGORIO, 2011). Existen numerosos servidores *web* dentro de los que se encuentran *Microsoft-IIS, Apache, Nginx, LiteSpeed, Google Servers, Tomcat, Lighttpd, IBM Servers, Yahoo Traffic Server, Jetty, AOLserver, Roxen* y *Caudium*. Algunos son más usados que otros debido a que poseen características distintivas que marcan la diferencia en cuanto a su selección.

Según estadísticas históricas y de uso diario proporcionadas por *Netcraft* (sitio inglés especializado en ofrecer estadísticas de diferentes tecnologías *web*), *Apache* tiene amplia aceptación en la red, ya que desde 1996 es el servidor HTTP más usado. Alcanzó su máxima cuota de mercado en 2005 siendo el servidor empleado en el 70 % de los sitios *web* posteados en Internet; sin embargo, ha sufrido un descenso en su cuota de mercado en los últimos años debido a su incapacidad de manejar de manera exitosa grandes cantidades de peticiones concurrentes respecto a otras tecnologías, como por ejemplo *Node.js* (ibíd.).

1.6.1. Servidor *HTTP Apache*

Apache fue lanzado en el año 1995. Es un servidor *web* de código abierto, robusto, flexible, rápido y eficiente, continuamente actualizado, adaptado a los nuevos protocolos y cuya implementación se realiza de forma colaborativa. El proyecto está dirigido y controlado por un grupo de voluntarios de todo el mundo conocido como el Grupo Apache. Su última entrega es *Apache 2* que es una profunda revisión del servidor *Apache*, centrándose en la escalabilidad, seguridad y rendimiento (KNAUS; DRAPER; WAGNER y ZIMMERMAN, 1985). *Apache* cuenta con características importantes como:

- Popular (fácil de conseguir ayuda/soporte)
- Modular
- Código abierto
- Multi-plataforma
- Extensible

1.6.2. *Node.js*

El 8 de noviembre de 2009 Ryan Dahl presentó *Node.js* como un entorno de desarrollo en *JavaScript* de lado del servidor, asíncrono y orientado a eventos. Usa el motor de *JavaScript* V8 de *Google*, dotando a este entorno de una gran velocidad y agilidad, y sumando además una capacidad de entrada/salida altamente potente y ligera. Este permite construir aplicaciones escalables y escribir código que maneje miles de cone-

xiones simultáneas en un solo equipo. Entre las compañías que usan esta tecnología se destacan *LinkedIn, eBay, Yahoo!, Microsoft*, entre otros (TILKOV y VINOSKI, 2010).

V8 es el motor de *JavaScript* que *Google* usa en su navegador *Chrome* y que interpreta código además de ejecutarlo. El V8 es rápido, está escrito en *C++* y se puede descargar por separado e incorporarlo a cualquier aplicación. Así nace *Node.js*, cambiando el propósito por el que se creó V8 y usándolo en el lado del servidor (ESCOBAR; URANGO; CASTRO et al., 2015).

La meta principal de *Node.js* es la de proporcionar una manera fácil de construir programas de red escalables. En lenguajes como *Java* y *PHP*, cada conexión genera un nuevo hilo con su respectiva reserva de memoria. Esto limita la cantidad de peticiones concurrentes que puede tener un sistema (TILKOV y VINOSKI, 2010). Por ejemplo, un servidor muy común es *Apache*, el cual crea un nuevo hilo por cada conexión cliente-servidor. Esto funciona bien con pocas conexiones, pero a partir de 400 conexiones simultáneas, el número de segundos para atender las peticiones crece de manera considerable. Así pues, *Apache* funciona bien en entornos clásicos, pero no es el mejor servidor para lograr máxima concurrencia (ESCOBAR; URANGO; CASTRO et al., 2015). *Node.js* resuelve este problema cambiando la forma en que se realiza una conexión con el servidor. En lugar de generar un nuevo hilo para cada conexión, cada conexión entra en el bucle de ejecución y dispara el evento dentro de la pila de trabajo. De esta forma, nunca se quedará en punto muerto, dado que no se permiten bloqueos y no se bloquean directamente las llamadas de entrada/salida, permitiendo así miles de sesiones concurrentes (CANTELON; HARTER; HOLOWAYCHUK y RAJLICH, 2014).

Esta tecnología está diseñada para situaciones en la que se espere una gran cantidad de tráfico y donde la lógica del servidor y el procesamiento requeridos, sean sencillas para dar una respuesta lo antes posible. *Node.js* es recomendable en aplicaciones *web* que necesiten conexiones persistentes con el navegador del cliente (CASTRELO CID, 2014). Usando ciertas librerías, en concreto *socket.io*, se puede hacer que una aplicación envíe datos al usuario en tiempo real; es decir, que el navegador mantenga la conexión siempre abierta y reciba de manera continua nuevos datos cuando se requiera (MUÑOZ DE LA TORRE, 2013).

Existen aplicaciones que por su naturaleza se benefician de las características a las que puede dar soporte *Node.js*. Aun así, sigue siendo el programador quien debe adaptar el programa a estas propiedades y saber aprovecharlas (ÍSCAR MARTÍNEZ, 2015). Algunos ejemplos de su uso son:

- *Chat*
- Datos en *Streaming*
- Gestión de base de datos
- Servidor *proxy* y balanceador de carga
- Servidor SMTP
- Servidor DNS
- Juegos *Online* multijugador

Mediante el análisis de las anteriores tecnologías para el desarrollo de servidores, se puede apreciar que no existen criterios exactos para definir cuál es mejor. Todo depende de la aplicación que se desee realizar,

específicamente del objetivo principal que persiga dicha aplicación y en función de eso, se analizan las ventajas que pueda poseer una sobre otra. A partir del estudio realizado se evidencia la ventaja que ofrece *Node.js* sobre *Apache* en cuanto al desarrollo de aplicaciones en tiempo real, debido principalmente al manejo de las conexiones y al tratamiento de solicitudes concurrentes.

1.7. Lenguaje de Programción y *Framework*

Como principales *frameworks* para el desarrollo de la aplicación se tienen *Phaser.io* como motor de videojuego y *Socket.io* para la implementación del protocolo *WebSocket*, ambos sobre el lenguaje de programación *JavaScript* que gracias a la elección de *Node.js*, también se puede emplear del lado servidor.

Phaser.io

Es un entorno de desarrollo de código abierto, rápido y libre para hacer juegos en el lenguaje *HTML5* en navegadores de escritorio y móviles, caracterizado por ser potente y sencillo de utilizar (VAN DAMME y LINDE, 2014). Se puede programar en *JavaScript* o *TypeScript*, además de que no es necesario ningún entorno de desarrollo integrado, ya que es para *HTML5* (VERA; MORENO; RODRÍGUEZ et al., 2015). Cuenta con las siguientes ventajas:

- Curva de aprendizaje rápida.
- *Open Source*(código abierto).
- Soportado por los últimos exploradores tanto de los sistemas operativos *Mac* o *Linux* como *iOS*, *Android*, *Windows Phone*, entre otros.

Socket.io

Desde el inicio de las aplicaciones *web*, los desarrolladores han trabajado para obtener diferentes formas de comunicación *full duplex* (bidireccional) entre el servidor y el navegador. Por primera vez, hay una técnica para construir un sistema de comunicación *full duplex* completo mediante el uso de HTML5, y es precisamente el protocolo *WebSocket*. Esta es una nueva función de comunicación revolucionaria en la especificación de HTML5 que define un canal de comunicación bidireccional que opera a través de la red por un solo *socket* o punto de conexión. *Socket.io* es una capa de abstracción para *WebSocket*, que proporciona un servidor y una biblioteca cliente, para realizar en tiempo real la transmisión de las actualizaciones que ocurren entre un servidor *web* y un cliente de navegador. Es un módulo de *Node.js* disponible a través del manejador de paquetes de *Node* o *Node Package Manager* (NPM, por sus siglas en inglés) (RAI, 2013).

JavaScript

Es un lenguaje de programación interpretado con capacidades orientadas a objetos. El núcleo de uso general de la lengua ha sido incorporado en *Netscape*, *Internet Explorer* y otros navegadores *web* para brindar

mayor dinamismo en la programación *web* (FLANAGAN, 2006). A diferencia de los idiomas más tradicionales tales como *Java*, *C* o incluso *Smalltalk*, no tiene clases, y no fomenta la encapsulación o incluso la programación estructurada. En su lugar *JavaScript* se esfuerza en maximizar la flexibilidad del sistema (RICHARDS; LEBRESNE; BURG y VITEK, 2010).

1.8. Persistencia de datos

MySQL es muy usado en el desarrollo de aplicaciones *web*, ya que por lo general cuentan con muchas lecturas y pocas escrituras. Es rápido y puede satisfacer las demandas de alta velocidad en Internet, además de ser de código abierto, permite a todos los usuarios descargar y ampliar el código para satisfacer sus necesidades. Esta tecnología es muy empleada además en las aplicaciones de nivel empresarial, pues ofrece soporte directo a través de la compañía *MySQL* AB (MYSQL, 2001; NIXON, 2012).

El gestor de base de datos *MySQL* tiene la ventaja de que se ejecuta en varias plataformas, además de que ofrece a sus usuarios estabilidad, apoyo y bajo costo. Su documentación es amplia, además de tener un sitio *web* oficial que contiene material de referencia. Esta tecnología ofrece soporte de alta calidad para las aplicaciones en la *web* (MYSQL, 2001).

MySQL es la base de datos número 1 para las aplicaciones basadas en la *web*, utilizada por *Facebook*, *Twitter*, *LinkedIn*, *Yahoo!*, *Amazon Web Services*. Cuenta con grandes volúmenes de datos, controlados por las redes sociales, y una conexión de banda ancha móvil de alta velocidad (ORACLE, 2016a). Además, es respaldada por 17 de los 20 principales proveedores de software de todo el mundo, pues confían en *MySQL* como gestor de la base de datos de sus productos (ORACLE, 2016b).

1.9. Metodología de desarrollo de Software

Las metodologías de desarrollo de software son un conjunto de procedimientos y técnicas que ayudan a que un software tenga la mejor calidad en el menor tiempo posible, con los menores costos de producción y, que a su vez, satisfaga las demandas del cliente, permitiendo además a los analistas documentar los aspectos más significativos en cada una de las etapas de su desarrollo (SOMMERVILLE y GALIPIENSO, 2005). Una metodología de ingeniería de software es la encargada de guiar el proceso de construcción de un software. Reúne aquellos documentos asociados a la configuración de datos que son necesarios para hacer que estos programas operen de manera correcta. Estos productos se desarrollan para algún cliente específico o para un mercado en general.

Disímiles metodologías han servido de apoyo en los últimos tiempos para el desarrollo de software, las cuales con el transcurso de los años han evolucionado de manera significativa, permitiendo así el éxito o fracaso de muchos de los sistemas desarrollados en distintas ramas. Estas pueden tener un enfoque ágil o tradicional y son utilizadas en dependencia de las características del producto que se desea desarrollar.

1.9.1. Metodologías Ágiles

En febrero de 2001, tras una reunión celebrada en Utah-EEUU, nace el termino ágil aplicado al desarrollo de software. En esta reunión participan un grupo de 17 expertos de la industria del software, incluyendo algunos de los creadores o impulsores de metodologías de software. Su objetivo fue esbozar los valores y principios que deberían permitir a los equipos, desarrollar software rápidamente, respondiendo a los cambios que puedan surgir a lo largo del proyecto. Se pretendía ofrecer una alternativa a los procesos de desarrollo de software tradicionales, caracterizados por ser rígidos y dirigidos por la documentación que se genera en cada una de las actividades desarrolladas (CANÓS; LETELIER y PENADÉS, 2003). A continuación, se describen un grupo de metodologías ágiles para guiar el proceso de desarrollo de software.

Scrum

Es un modelo de referencia que define un conjunto de prácticas y roles, que puede tomarse como punto de partida para definir el proceso de desarrollo que se ejecutará durante un proyecto. *Scrum* es el nombre con el que se denomina a los marcos de desarrollo ágiles caracterizados por adoptar una estrategia de desarrollo incremental, en lugar de la planificación y ejecución completa del producto, y en el solapamiento de las diferentes fases del desarrollo, en lugar de realizar una tras otra en un ciclo secuencial o de cascada (LETELIER, 2006). Esta metodología basa la calidad del resultado más en el conocimiento de las personas que conforman los equipos, que en la calidad de los procesos empleados. Un principio clave de *Scrum* es el reconocimiento de que durante un proyecto los clientes pueden cambiar de idea sobre lo que quieren y necesitan, y que los desafíos impredecibles no pueden ser fácilmente enfrentados de una forma predictiva y planificada. Por lo tanto, adopta una aproximación pragmática, aceptando que el problema no puede ser completamente entendido o definido, centrándose en maximizar la capacidad del equipo de entregar rápidamente y responder a requisitos emergentes (DINGSOYR; NERUR; BALIJEPALLY y MOE, 2012).

AUP

La metodología Proceso Unificado Ágil (AUP, por sus siglas en inglés) es una versión simplificada del Proceso Racional Unificado (RUP, por sus siglas en inglés) usando técnicas ágiles y conceptos que aún se mantienen válidos en esta última. El proceso unificado es un marco de desarrollo de software iterativo e incremental, que a menudo es considerado como un proceso altamente ceremonioso porque especifica muchas actividades y artefactos involucrados en el desarrollo de un software. Dado que es un marco de procesos, puede ser adaptado. AUP se preocupa especialmente de la gestión de riesgos, proponiendo que aquellos elementos con alto riesgo obtengan prioridad en el proceso de desarrollo y se aborden en etapas iniciales. Para ello, se crean y mantienen listas identificando los riesgos desde etapas iniciales del proyecto. Especialmente relevante en este sentido, es el desarrollo de prototipos ejecutables durante la fase de elaboración del producto, donde se demuestre la validez de la arquitectura para los requisitos clave del producto y que determinan los riesgos técnicos (CORDERO, 2010; DINGSOYR; NERUR; BALIJEPALLY y MOE, 2012).

XP

En esta metodología todos los requerimientos se expresan como escenarios llamados historias de usuario, los cuales se implementan directamente como una serie de tareas. Todas las pruebas se deben ejecutar de manera satisfactoria cuando el código nuevo se integra al sistema, existiendo un pequeño espacio de tiempo entre cada entrega del sistema. Programación Extrema (XP, por sus siglas en inglés) se diferencia de las metodologías tradicionales principalmente en que pone más énfasis en la adaptabilidad que en la previsibilidad (LETELIER, 2006). Una de sus ventajas sobre otras metodologías, es que se centra en potenciar las relaciones entre el equipo de desarrollo, el cual debe estar integrado por cuatro personas como máximo, y el cliente; propiciando así la existencia de un entorno de trabajo agradable. Esto es a su vez, uno de los requisitos para lograr el éxito de un proyecto (TIWARI; PRAVEEN y MANUJ, 2013). Entre las características que posee se destaca el desarrollo iterativo e incremental, permitiendo pequeñas mejoras del producto; realización de pruebas unitarias y pruebas de aceptación continuas, así como la corrección de errores antes de añadir una nueva funcionalidad, permitiendo al equipo hacer entregas frecuentes (BASHIR y QURESHI, 2012; DINGSOYR; NERUR; BALIJEPALLY y MOE, 2012).

Consideraciones finales del capítulo

Con el estudio de los principales conceptos asociados a videojuegos multijugador, específicamente los juegos de navegador multijugador *online*, se adquirió un mayor conocimiento acerca de estos en cuanto a las principales métricas a tener en cuenta a la hora de su creación. Añadiendo a esto la caracterización del videojuego Mundo Verde y de la Plataforma de gestión de videojuegos de navegador, sumado a el estudio de las diferentes tecnologías, se puede desarrollar un componente que permita que este videojuego sea multijugador *online*.

A partir del estudio realizado de los dos protocolos de comunicación para aplicaciones *web* en tiempo real anteriormente vistos, se selecciona el protocolo *WebSocket* por ser el ideal en esta rama de desarrollo, ya que ofrece ventajas sobre *Long Polling* en cuanto a rendimiento y tiempos de respuesta. Se selecciona como tecnología del lado servidor *Node.js* por sus ventajas en cuanto a aplicaciones con gran cantidad de solicitudes concurrentes al servidor, y además que a través de sus múltiples módulos podemos utilizar de manera sencilla el protocolo *Websocket*. Se selecciona el gestor de base de datos *MySQL* por ser ideal para aplicaciones *web* que no requieran de muchas solicitudes, además de que la plataforma de gestión con la cual se debe integrar el sistema hace uso del mismo gestor, existiendo así un mayor acoplamiento entre ambos sistemas. Para guiar el proceso de desarrollo de la aplicación se selecciona metodología Programación Extrema, por ser muy útil en proyectos pequeños, permite hacer entregas continuas al cliente y es muy efectiva si ocurren cambios en los requisitos.

CAPÍTULO 2

Propuesta de solución

Introducción

En este capítulo se describen los elementos necesarios para dar solución al objetivo propuesto, desarrollando un componente que nos permita comunicar de manera *online* a los jugadores del videojuego Mundo Verde, haciendo uso del protocolo de comunicación *WebSocket*. Además, se documentan los aspectos más relevantes de la Ingeniería de Software utilizando la metodología XP, como son la captura de requisitos, el tiempo estimado que debe durar el proyecto basado en las tareas de desarrollo y la arquitectura que sostendrá al software en cuestión.

2.1. Propuesta de solución

Para dar solución al problema de investigación planteado se propone desarrollar un componente de intercomunicación *online* que permita a los jugadores del videojuego Mundo Verde interactuar desde diferentes estaciones de trabajo, ubicadas en distintos puntos de la red. En el desarrollo de dicha solución se empleará la arquitectura cliente-servidor, que permite establecer la comunicación necesaria entre los diferentes clientes a través del servidor. Para la creación del servidor se hará uso de *Node.js* y para la implementación del protocolo *WebSocket* se hará uso del *framework Socket.io*, el cual proporciona el canal de comunicación entre el cliente y el servidor.

En un inicio se creará el servidor de juego a través de la funcionalidad *io.listen*, a la cual se le especifica el puerto por el cual escuchará. El servidor se dedicará específicamente a gestionar las conexiones de los clientes y escuchar los eventos que se activan en estos, permitiendo de este modo informar a los clientes cuando ocurre un cambio en uno de ellos para que todos se actualicen. Una vez que el servidor se encuentre listo, se dará paso a crear las conexiones de los clientes. Para ello se debe utilizar la funcionalidad *io.connect*, a la cual se le especifica la dirección donde esté alojado el servidor y el puerto por el cual escucha. Cuando se ejecute esta funcionalidad en cada uno de los clientes, en el servidor se debe activar el evento "*Connection*" el cual será escuchado con la funcionalidad *io.on* y de este modo se establece la comunicación entre cliente

y servidor, de este modo el canal de comunicación queda listo para la transmisión de los mensajes.

Llegado a este punto, todo queda listo para realizar las configuraciones necesarias tanto en el cliente como en el servidor, para cuando se realice una acción por parte de uno de los clientes conectados, los demás también puedan ver los cambios ocurridos. Para esto se hará uso de la funcionalidad *io.emit*, que envía eventos; mientras que se debe emplear la funcionalidad *io.on* para escucharlos. A ambas funcionalidades se le especifican el nombre del evento y el mensaje que se quiera enviar y se utilizarán en el cliente o en el servidor según sea necesario, con el objetivo de lograr que la dinámica del juego se ejecute en cada uno de los clientes de manera concurrente.

Cuando un cliente se conecte al servidor, este chequea si existen capacidades, es decir, aun no hay 3 jugadores conectados o la partida no ha comenzado, en tal caso el servidor le enviará al cliente un identificador según el orden de conexión y este se quedarán esperando por que se conecten más jugadores. En el momento que existan al menos dos jugadores conectados, el servidor dará permiso para que se muestre el botón "Comenzar" que da comienzo a la partida. Una vez comenzada la partida cada cliente solicitará permiso al servidor para mostrar el botón "Tirar", el cual responderá a dicha solicitud; sin embargo, solo en el cliente que le corresponde el turno de juego se mostrará el botón. Además, se le notificará al cliente con un mensaje que es su turno. En el momento en que el jugador en turno de clic en el botón "Tirar", se calcula el valor del dado y se envía al servidor, quien lo reenvía a todos los clientes para que estos muestren el dado con ese valor. Luego con ese valor y la posición del jugador se calcula la nueva posición del jugador y se envía al servidor, para que lo reenvíe a todos los clientes y estos actualicen la posición de ese jugador.

El proceso de dar clic en el botón "Tirar" se repetirá hasta que uno de los jugadores llegue a la meta, momento en que se mostrará el jugador ganador y se enviará al servidor los resultados de la partida: un identificador de cada jugador con el lugar que obtuvo. Si durante la partida uno de los jugadores decide desconectarse puede dar clic en el botón "Volver", esta acción provoca que regrese al menú de inicio del juego. Las acciones de cerrar o actualizar la pestaña del navegador, así como dirigirse a otro enlace también serán tomadas como desconexiones, y al desconectarse uno de los jugadores el servidor se reiniciará para dar comienzo a una partida nueva.

Una vez creada las configuraciones necesarias se dará paso a la creación de una base da datos que contenga una tabla llamada "jugadores" con los campos nombre del jugador y lugar obtenido. Para ello se hará uso del SGBD *MySQL*, y para crear la conexión del servidor hacia la base de datos se utilizará el módulo *mysql* que provee *Node.js* para el trabajo con bases de datos *MySQL*. Esto ocurre a través de la funcionalidad *mysql.createConnection*, a la cual se le especifica la dirección donde esté alojada la base de datos, el usuario, la contraseña si posee y el nombre de la base de datos. Establecida la conexión del servidor a la base de datos, mediante la funcionalidad *connection.query* se enviarán los datos del videojuego.

Logrado que el sistema permita guardar la información generada al finalizar una partida en la base de datos, el videojuego se encuentra listo para la integración con la Plataforma de gestión de videojuegos de navegador. Para ello es necesario realizar una solicitud a la base de datos y para obtener todos los datos almacenados en la tabla "jugadores". Con los resultados de la consulta mencionada, utilizando además el

módulo *fs* que provee *Node.js* para la lectura y escritura de archivos externos, mediante la funcionalidad *fs.writeFile*, se genera el archivo *salva.json* dentro de la carpeta *saves*, ubicada dentro de la carpeta principal del videojuego. Posterioemente es necesario crear un comprimido en formato *.zip* de la carpeta principal del videojuego que tenga el nombre del videojuego sin espacios y que contenga en su interior el *index.html* con sus dependencias. Este archivo comprimido será cargado por la plataforma, quedando de este modo integrado el videojuego.

Con la utilización e implementación de los mecanismos y funcionalidades antes descritos, se logrará una mayor versatilidad en el videojuego, puesto que no será un videojuego multijugador que solo se puede jugar en una misma máquina cliente, sino que además se puede jugar desde distintos lugares ubicados en la red, logrando de este modo una mayor experiencia de juego entre sus jugadores. También permite que pueda ser reutilizada en otros sistemas que cumplan con algunas de las características siguientes:

- Basados en la arquitectura cliente-servidor.
- Aplicación en tiempo real.
- Uso de tecnologías tales como *Node.js* como servidor, *WebSocket* como protocolo de comunicación y el *framework Socket.io* para la implementación de dicho protocolo.

2.2. Ingeniería de *Software*

La nueva versión del videojuego debe contar con un conjunto de requerimientos funcionales y no funcionales para lograr convertirse en el producto que se espera. Para la captura de estos requisitos es primordial el uso de la Ingeniería de *Software*, aplicando paso a paso las etapas por la que transcurre la metodología XP.

2.2.1. Etapa de planificación

Una de las primeras tareas del equipo de desarrollo es realizar la captura de las funcionalidades y atributos de calidad que tendrá el sistema. Las especificaciones obtenidas deben comunicar las necesidades del sistema a través del cliente (SOMMERVILLE y GALIPIENSO, 2005). En esta fase el cliente establece la prioridad de cada requisito lo que se traduce en la metodología XP como historia de usuario, y posteriormente se realiza una estimación del esfuerzo necesario de cada una de ellas. Se toman acuerdos sobre el contenido de la primera entrega la cual debería obtenerse en no más de tres meses. Esta fase debe durar solo unos pocos días (LETELIER, 2006).

Requisitos funcionales

Los requerimientos funcionales son declaraciones formales de las prestaciones que proporciona el sistema. Estas deben argumentar la manera en que la aplicación puede reaccionar a las entradas y cómo se debe comportar en situaciones particulares (SOMMERVILLE y GALIPIENSO, 2005). El proceso de levantamiento de requisitos concluyó con un total de 5 requisitos, los cuales se enumeran a continuación:

RF 1. Conectar el cliente y el servidor
RF 2. Configurar el cliente y el servidor
RF 3. Crear base de datos
RF 4. Crear conexión del servidor con la base de datos
RF 5. Integrar el videojuego con la Plataforma

Requisitos no funcionales

Los requisitos no funcionales representan restricciones a los servicios que debe brindar el software (SOMMERVILLE y GALIPIENSO, 2005). A continuación, se establecen los requisitos no funcionales establecidos para el desarrollo de la aplicación.

RnF 1. *Software*

- Cliente: Sistema Operativo *Windows XP* o superior, *Linux*, navegador web *Firefox 12.0* o superior.
- Servidor: Sistema Operativo *Windows XP* o superior, *Linux*, servidor *Node.js 4.1.1*, lenguaje de programación *JavaScript* y gestor de BD *MySQL 5.6.30*.

RnF 2. *Hardware*

- Cliente: Procesador *Pentium III*, 512 MB de RAM, 50 GB de disco duro.
- Servidor: Procesador *Pentium IV* a 2.2 GHz, 1 GB de RAM, 50 GB de disco duro.

RnF 3. *Usabilidad*

- El servidor se debe implementar para que permita 3 conexiones como máximo.

RnF 4. *Rendimiento*

- Los tiempos de respuesta de los mensajes entre el cliente y el servidor deben oscilar entre 0 y 1 segundo.

Descripción de historias de usuario

Las historias de usuario son una técnica utilizada para especificar los requisitos del software. Se trata de tarjetas de papel en las cuales el cliente describe brevemente las características que el sistema debe poseer, sean requisitos funcionales o no funcionales. El tratamiento de las historias de usuario es muy dinámico y flexible. Cada historia de usuario es comprensible y delimitada para que los programadores puedan implementarla en unas semanas (CANÓS; LETELIER y PENADÉS, 2003). Las estimaciones de esfuerzo asociado a la implementación de las historias la establecen los programadores utilizando como medida el punto, el cual equivale a una semana ideal de programación. Las historias generalmente valen de 1 a 3 puntos (LETELIER, 2006).

Tabla 2.1. Historia de usuario # 1

Historia de usuario	
Número: 1	**Nombre:** Conectar el cliente y el servidor
Usuario: Desarrollador	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Alta
Puntos estimados: 3	**Iteración asignada:** 1
Programador responsable: José Manolo Díaz Aguila	
Descripción: Debe permitir la creación del servidor y la conexión a este desde el cliente a través del protocolo *WebSocket*.	
Observaciones:	

Tabla 2.2. Historia de usuario # 2

Historia de usuario	
Número: 2	**Nombre:** Configurar el cliente y el servidor
Usuario: Desarrollador	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Alta
Puntos estimados: 3	**Iteración asignada:** 1
Programador responsable: José Manolo Díaz Aguila	
Descripción: Se debe configurar el cliente y el servidor de modo que cada cual sepa cuando enviar y recibir datos, además de qué hacer con ellos.	
Observaciones:	

Tabla 2.3. Historia de usuario # 3

Historia de usuario	
Número: 3	**Nombre:** Crear base de datos
Usuario: Desarrollador	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Baja
Puntos estimados: 2	**Iteración asignada:** 2
Programador responsable: José Manolo Díaz Aguila	
Descripción: Se debe crear una base de datos con una tabla "Jugadores"que contenga los datos Nombre del jugador y Lugar obtenido.	
Observaciones:	

Tabla 2.4. Historia de usuario # 4

Historia de usuario	
Número: 4	**Nombre:** Crear conexión del servidor con la base de datos
Usuario: Desarrollador	

Continúa en la próxima página

Tabla 2.4. Continuación de la página anterior

Prioridad en negocio: Alta	**Riesgo en desarrollo:** Media
Puntos estimados: 3	**Iteración asignada:** 2
Programador responsable: José Manolo Díaz Aguila	
Descripción: Debe permitir la conexión entre el servidor de juego y la base de datos creada con anterioridad.	
Observaciones:	

Tabla 2.5. Historia de usuario # 5

Historia de usuario	
Número: 5	**Nombre:** Integrar el videojuego con la Plataforma
Usuario: Desarrollador	
Prioridad en negocio: Media	**Riesgo en desarrollo:** Baja
Puntos estimados: 2	**Iteración asignada:** 3
Programador responsable: José Manolo Díaz Aguila	
Descripción: Se debe crear un archivo *salva.json* que contenga los datos alamcenados en la tabla jugadores de la base de datos.	
Observaciones:	

2.2.2. Tareas de desarrollo

Cada una de las historias de usuario se transformarán en una o más tareas de desarrollo, las cuales corresponderán a un período ideal de una a tres semanas y serán asignadas a desarrolladores específicos del equipo.

Tabla 2.6. Tarea de ingeniería # 1

Tarea	
Número de tarea: 1	**Número de Historia de usuario:** 1
Nombre de la tarea: Crear servidor y conexión del cliente al servidor	
Tipo de tarea: Desarrollo	**Puntos estimados:** 3
Fecha de inicio: 1 de febrero de 2016	**Fecha de fin:** 21 de febrero de 2016
Programador responsable: José Manolo Díaz Aguila	
Descripción: Se debe crear un servidor para la comunicación entre los clientes,y ademas la conexion de estos al servidor haciendo uso de *Node.js* y *Socket.io*	

Tabla 2.7. Tarea de ingeniería # 2

Tarea	
Número de tarea: 2	**Número de Historia de usuario:** 2

Continúa en la próxima página

Tabla 2.7. Continuación de la página anterior

Nombre de la tarea: Configurar el cliente y el servidor	
Tipo de tarea: Desarrollo	**Puntos estimados:** 3
Fecha de inicio: 29 de febrero de 2016	**Fecha de fin:** 21 de marzo de 2016
Programador responsable: José Manolo Díaz Aguila	
Descripción: Se deben configuar el servidor y el cliente de manera que cada uno sepa cuando enviar y recibir datos haciendo uso de *Node.js*, *Socket.io* y *Phaser.io*	

Tabla 2.8. Tarea de ingeniería # 3

Tarea	
Número de tarea: 3	**Número de Historia de usuario:** 3
Nombre de la tarea: Crear base de datos	
Tipo de tarea: Desarrollo	**Puntos estimados:** 2
Fecha de inicio: 28 de marzo de 2016	**Fecha de fin:** 1 de abril de 2016
Programador responsable: José Manolo Díaz Aguila	
Descripción: Se debe crer una base de datos que contenga una tabla "Jugadores" con los campos: Nombre del jugador y Lugar obtenido.	

Tabla 2.9. Tarea de ingeniería # 4

Tarea	
Número de tarea: 4	**Número de Historia de usuario:** 4
Nombre de la tarea: Crear conexión del servidor con la base de datos	
Tipo de tarea: Desarrollo	**Puntos estimados:** 3
Fecha de inicio: 18 de abril de 2016	**Fecha de fin:** 6 de mayo de 2016
Programador responsable: José Manolo Díaz Aguila	
Descripción: Debe permitir la conexión entre el servidor de juego y la base de datos creada, haciendo uso de *Node.js* y del módulo *mysql*.	

Tabla 2.10. Tarea de ingeniería # 5

Tarea	
Número de tarea: 5	**Número de Historia de usuario:** 5
Nombre de la tarea: Integrar el videojuego con la Plataforma	
Tipo de tarea: Desarrollo	**Puntos estimados:** 2
Fecha de inicio: 16 de mayo de 2016	**Fecha de fin:** 27 de mayo de 2016
Programador responsable: José Manolo Díaz Aguila	

Continúa en la próxima página

Tabla 2.10. Continuación de la página anterior

Descripción: En primer lugar se debe generar un archivo en formato *.json* con los datos del juego almacenados en la base de datos haciendo uso para esto de los módulos *myslq* para realizar la consulta a la base de datos y *fs* para generar el fichero externo. Luego se creará un archivo comprimido en formato *.zip* que contenga en su interior una carpeta con el *index.html* y sus dependencias, además del archivo generado.

2.2.3. Estimación de esfuerzo por historias de usuarios

Una vez definidas las historias de usuario y las tareas para dar cumplimiento a cada una de ellas. El equipo de desarrollo realiza la estimación de esfuerzo asociada a cada historia de usuario en relación a la importancia que poseen para el cliente y la complejidad de desarrollo.

Tabla 2.11. Estimación de esfuerzo por historia de usuario

Iteración	Historias de usuario		Puntos estimados (semanas)
1	1	Conectar el cliente y el servidor	3.0
	2	Configurar el cliente y el servidor	3.0
2	3	Crear base de datos	2.0
	4	Crear conexión del servidor con la base de datos	3.0
3	5	Integrar el videojuego con la Plataforma	2.0
Total			**13.0**

2.2.4. Plan estimado de entregas

El plan de entregas especifica exactamente qué historias de usuario serán implementadas en cada entrega del sistema y sus prioridades, de modo que también permita conocer con exactitud qué historias de usuario serán implementadas en la próxima liberación. Debe ser negociado y elaborado en forma conjunta entre el cliente y el equipo desarrollador durante las reuniones de planificación de entregas. El objetivo es hacer entregas frecuentes para obtener una mayor retroalimentación.

Tabla 2.12. Plan de entregas

Iteración	Entrega	Fecha
1	Versión 0.1	24 de marzo del 2016
2	Versión 0.2	9 de mayo del 2016
3	Versión 0.3	30 de mayo del 2016

2.2.5. Plan de duración de las iteraciones

En este plan se propone la prioridad con que se irán implementando las historias de usuarios organizadas por iteraciones, así como las posibles fechas de liberación.

- **Iteración 1:** Tiene como objetivo darle cumplimiento a las historias de usuario 1 y 2, consideradas de gran valor para el desarrollo de la aplicación, pues permiten la comunicación entre los jugadores del videojuego y la actualización en tiempo real en cada uno de los clientes.
- **Iteración 2:** Tiene como objetivo darle cumplimiento a las historias de usuario 3 y 4, la cuales son dos importantes funcionalidades que debe poseer el software para almacenar los datos del juego.
- **Iteración 3:** Tiene como objetivo darle cumplimiento a la historia de usuario 5 la cual permite la integración con la plataforma.

Tabla 2.13. Plan de duración de las iteraciones

Iteración	Historias de usuario		Duración (semanas)
1	1	Conectar el cliente y el servidor	6.0
	2	Configurar el cliente y el servidor	
2	3	Crear base de datos	5.0
	4	Crear conexión del servidor con la base de datos	
3	5	Integrar el videojuego con la Plataforma	2.0
Total			**13.0**

2.3. Etapa de diseño

La metodología XP establece recomendaciones o premisas a la hora de abordar esta etapa, como es el caso de la simplicidad. Siempre es más rápido implementar un diseño sencillo que uno complejo, por lo que se debe tratar siempre de realizar las cosas de la manera más sencilla posible. Si alguna parte de la implementación resulta seriamente compleja, esta debe replantearse; así de este modo, cualquier modificación será menos costosa de realizar.

2.3.1. Tarjetas CRC

Para poder diseñar el sistema como un equipo, se debe cumplir con tres principios: Clase, Responsabilidad y Colaboración (CRC). Las tarjetas CRC permiten desprenderse del método de trabajo basado en procedimientos y trabajar con una metodología basada en objetos. Posibilitan además que el equipo completo contribuya en la tarea del diseño. A continuación, se muestran las tarjetas CRC del sistema en cuestión.

Tabla 2.14. Tarjeta CRC # 1

Tarjeta CRC	
Clase: *Socket.io*	
Responsabilidad	**Colaboración**

Continúa en la próxima página

Tabla 2.14. Continuación de la página anterior

• Es un módulo encargado de crear el servidor y la conexión del cliente a este, además de las configuraciones necesarias tanto en el servidor como en el cliente para enviar y recibir mensajes en el momento requerido, según la lógica de juego.	*io.sockets.on* *io.connect* *socket.on* *io.sockets.emit* *socket.emit*

Tabla 2.15. Tarjeta CRC # 2

Tarjeta CRC	
Clase: *mysql*	
Responsabilidad	**Colaboración**
• Es un módulo encargado de crear la conexión de la base de datos con el servidor, además se encarga de insertar los resultados al finalizar cada partida en la base de datos y solicitar de esta la información deseada para generar el archivo json necesario para la integración del videojuego con la plataforma.	*mysql.createConnection* *connection.query* *connection.end*

Tabla 2.16. Tarjeta CRC # 3

Tarjeta CRC	
Clase: fs (*File System*)	
Responsabilidad	**Colaboración**
• Es el módulo encargado de crear el archivo *.json* con los datos guardados del juego. Este archivo una vez que se integre el videojuego con la plataforma, es utilizado por esta para generar estadísticas.	*fs.writeFile* *JSON.stringify*

2.3.2. Patrones de diseño

Los patrones de diseño nos ofrecen esquemas para refinar subsistemas y componentes de un sistema software, o las relaciones entre ellos. Describen una estructura de comunicación recurrente entre componentes, que sirve para resolver un problema general de diseño dentro de un contexto particular (GARCÍA PEÑALVO, 1998). Algunos de los patrones empleados en el desarrollo del sistema son los que se muestran a continuación.

Los patrones *GRASP* representan los principios básicos de la asignación de responsabilidades a objetos, expresados en forma de patrones. Estos tipifican buenos principios de programación, cuya eficacia ha sido demostrada con anterioridad en escenarios similares.

- **Alta cohesión**: La cohesión es una medida que expresa cuan relacionadas y enfocadas están las responsabilidades de una clase. Una alta cohesión caracteriza a las clases con responsabilidades estrechamente relacionadas que no realicen un trabajo enorme (LARMAN, 2005). Este patrón se evidencia en las relaciones entre las clases que conforman el componente, de modo que cada una de ellas necesita conocer solamente las funciones que realiza.
- **Bajo acoplamiento**: El acoplamiento es una medida de la fuerza con la que una clase está vinculada a otras. Es decir, con la forma en la que se relacionan para el funcionamiento íntegro de la aplicación. Este patrón se encarga de que existan las menores relaciones posibles entre las clases, evitando así que existan dependencias innecesarias entre ellas (ibíd.). Este patrón se evidencia en las relaciones entre las clases que constituyen el componente, de modo que las clases no poseen relaciones entre sí, logrando total independencia.
- **Experto**: Este patrón se utiliza con el objetivo de asignar una responsabilidad a la clase que posee toda la información necesaria para realizarla. Es el patrón más utilizado debido a su esencia, por lo cual es empleado en todas las clases que contiene el componente.

2.3.3. Arquitectura

La arquitectura de software es la columna vertebral para desarrollar un sistema informático y es en gran medida responsable de permitir o no, ciertos atributos de calidad del sistema entre los que se destacan la confiabilidad y el rendimiento de la aplicación. Además, es un modelo abstracto reutilizable que puede transferirse de un sistema a otro y que representa un medio de comunicación y discusión entre participantes del proyecto. Esto permite la interacción e intercambio entre los desarrolladores con el objetivo final de establecer el intercambio de conocimientos y puntos de vista entre ellos (SOMMERVILLE y GALIPIENSO, 2005).

Para el desarrollo de la aplicación se seleccionó la arquitectura cliente-servidor, a través de la cual se crea un canal de comunicación entre ambas partes, donde el cliente es el encargado de procesar toda la lógica del juego y el servidor de comunicar los clientes entre sí, manejando los constantes eventos y actualizaciones que se desarrollan en una partida. En el cliente se encuentran las clases encargadas de establecer la comunicación

con el servidor, además de las clases encargadas de enviar y recibir datos del servidor. Por su parte en el servidor se alojan las clases encargadas para la creación de este y el manejo de las peticiones de los clientes. Es aquí donde se gestiona la comunicación y las solicitudes a la base de datos del juego, así como la creación del archivo *.json* para la integración del videojuego con la plataforma.

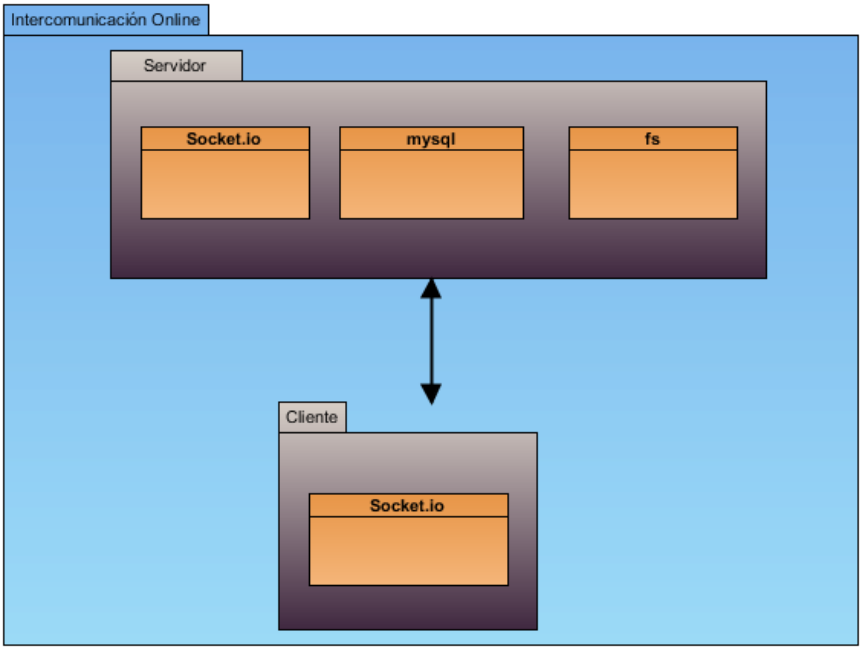

Figura 2.1. Arquitectura cliente-servidor

Servidor

- *Socket.io*: Este módulo permite escuchar las conexiones de los clientes al servidor así como los mensajes que estos envían y responder a estos en caso de que sea necesario. Además permite enviar mensajes al cliente cuando ocurre un cambio o una actualización. Permitiendo de este modo la comunicación entre los clientes en tiempo real.
- *mysql*: Este módulo es el encargado de crear la conexión con la base de datos y de realizar las solicitudes a esta. Permite además salvar los datos después de cada partida, brindando de este modo a la aplicación la posibilidad de guardar los datos más relevantes del videojuego.
- *fs*: Este módulo es el encargado de generar el archivo *.json* necesario para la integración del videojuego con la plataforma de gestión.

Cliente

- *Socket.io*: Este módulo permite crear la conexión desde el cliente al servidor y además enviar mensajes al servidor en el momento requerido, así como escuchar de este sus mensajes para ser procesados posteriormente en dependencia de la lógica del juego.

Consideraciones finales del capítulo

En el presente capítulo se realizó un levantamiento de las historias de usuario que intervienen en el desarrollo del sistema, además de la duración de cada una de ellas. Concluidos los procesos de la etapa de planificación, se procedió a la etapa de diseño, donde se estableció la arquitectura que tendría el software y las tarjetas CRC, todo esto con el objetivo de tener un mayor control y conocimiento sobre el desarrollo del sistema.

CAPÍTULO 3

Resultados

Introducción

En el presente capítulo se documentan los principales elementos de la etapa de implementación y pruebas del sistema, siguiendo las pautas que establece la metodología seleccionada XP. Se describen los estándares por los cuales se debe guiar el desarrollador en cuestión durante el desarrollo del software, así como las pruebas de aceptación realizadas con el objetivo de verificar el cumplimiento eficaz de cada requisito funcional del componente.

3.1. Estándares de implementación

Los estándares permiten que el código a desarrollar sea de una mejor calidad y que el mantenimiento de un software se vuelva menos complejo. A continuación, se muestra el estándar de implementación utilizado en la realización de la solución.

Estilos de comentarios

El estilo de los comentarios debe ser como el estilo de comentarios para C (/* */ o //).

Identación

La identación será de cuatro espacios, utilizándose la tabulación. No poseerán identación las llaves asociadas a los espacios de nombre o implementación de métodos.

Declaración de variables

Se evitará nombrar a las variables con abreviatura. Solo se hará uso de la abreviatura en aquellos casos donde el término sea de evidente comprensión. Todos los caracteres serán escritos en minúscula, excepto en caso de que existan nombres compuestos. En este caso, se escribirá con minúscula el primer nombre y los

restantes con letra inicial mayúscula, evitando usar el guión bajo.

Declaración de métodos

Siguen el mismo convenio de las variables.

Código fuente 3.1. Ejemplo del estándar de codificación

```
socket.on('posicionesClient', function (mensaje)
{
if(c == 0)
 {
  for(var i=0; i<mensaje.length ; i++)
  {
   var query = connection.query('INSERT INTO jugadores(nombre, lugar)
       VALUES(?, ?)', [mensaje[i].n, mensaje[i].p], function(error,
       result)
   {
     if(error)
     {
       throw error;
      }
     else
      {
      console.log("datos guardados"); //mensaje que se muestra si los
          datos fueron guardados satisfactoriamente
       }
    }
    );
  }
})
```

3.2. Etapa de pruebas

Uno de los pilares de esta metodología es el proceso de pruebas. XP anima a probar constantemente tanto como sea posible. Esto permite aumentar la calidad de los sistemas reduciendo el número de errores no detectados y disminuyendo el tiempo transcurrido entre la aparición de un error y su detección. También permite aumentar la seguridad de evitar efectos colaterales no deseados a la hora de realizar modificaciones.

Las pruebas son un conjunto de actividades que se pueden planificar por adelantado y llevar a cabo siste-

máticamente. Por esta razón se deben definir en el proceso de la ingeniería del software. Todo esto contribuye a elevar la calidad de los productos desarrollados y a la seguridad de los programadores al introducir cambios o modificaciones. Las pruebas que se realizan a la aplicación son las pruebas de aceptación o pruebas funcionales y pruebas de rendimiento, las primeras destinadas a evaluar si al final de una iteración se consiguió la funcionalidad requerida, diseñadas por el cliente final y la segunda para medir el tiempo de los mensajes entre en cliente y el servidor. Esto permite aumentar la calidad de los sistemas reduciendo el número de errores no detectados y disminuyendo el tiempo transcurrido entre la aparición de un error y su detección.

3.2.1. Pruebas de sistema o pruebas de aceptación

Las pruebas del sistema tienen como objetivo verificar la funcionalidad del sistema a través de sus interfaces externas comprobando que dicha funcionalidad sea la esperada en función de los requisitos del sistema. Generalmente las pruebas del sistema son desarrolladas por los programadores para verificar que su sistema se comporta de la manera esperada, por lo que podrían encajar dentro de la definición de pruebas unitarias que propone XP. Sin embargo, las pruebas del sistema tienen como objetivo verificar que el sistema cumple los requisitos establecidos por el usuario por lo que también pueden encajar dentro de la categoría de pruebas de aceptación.

Las pruebas realizadas al software constituyen su única garantía de calidad, las mismas pueden estar dirigidas a los componentes de forma individual o al sistema como un conjunto. Siguiendo la línea de la metodología, estas pruebas están dirigidas por cada historia de usuario, que se traducen a su vez en casos de prueba. Durante una iteración la historia de usuario seleccionada en la planificación de iteraciones se convertirá en una prueba de aceptación. A continuación, se muestran las pruebas realizadas al sistema una vez concluida la implementación.

Tabla 3.1. Prueba de aceptación # 1

Caso de prueba de aceptación	
Código: HU1_P1	**Historia de usuario:** 1
Nombre: Se conecta un jugador	
Descripción: Con el objetivo de probar que existe comunicación entre el cliente y el servidor, se debe ejecutar el juego y seguidamente dar clic en el botón “Jugar” para determinar si existe o no, algún posible error en la comunicación entre el cliente y el servidor.	
Condiciones de ejecución: El servidor debe de estar ejecutándose.	
Pasos de ejecución: • Abrir el archivo principal del juego *index.html*. • Dar clic en el botón Jugar.	
Resultados esperados: Debe mostrarse en pantalla: "Jugador 1: Conectado".	

Figura 3.1. Un jugador conectado

Tabla 3.2. Prueba de aceptación # 2

Caso de prueba de aceptación	
Código: HU1_P2	**Historia de usuario:** 1
Nombre: Se conecta un segundo jugador	
Descripción: Con el objetivo de probar que existe comunicación entre el cliente y el servidor, se debe ejecutar el juego y seguidamente dar clic en el botón “Jugar” para determinar si existe o no, algún posible error en la comunicación entre el cliente y el servidor.	
Condiciones de ejecución: El servidor debe de estar ejecutándose.	
Pasos de ejecución: • Abrir el archivo principal del juego *index.html*. • Dar clic en el botón Jugar.	
Resultados esperados: Debe mostrarse en pantalla: "Jugador 1: Conectado", "Jugador 2: Conectado". Se le notificará al Jugador 1 que el Jugador 2 se ha conectado.	

Figura 3.2. Dos jugadores conectados

Tabla 3.3. Prueba de aceptación # 3

Caso de prueba de aceptación	
Código: HU1_P3	**Historia de usuario:** 1
Nombre: Se conecta un tercer jugador	
Descripción: Con el objetivo de probar que existe comunicación entre el cliente y el servidor, se debe ejecutar el juego y seguidamente dar clic en el botón “Jugar” para determinar si existe o no, algún posible error en la comunicación entre el cliente y el servidor.	
Condiciones de ejecución: El servidor debe de estar ejecutándose.	
Pasos de ejecución: • Abrir el archivo principal del juego *index.html*. • Dar clic en el botón “Jugar”.	
Resultados esperados: Debe mostrase en pantalla: "Jugador 1: Conectado", "Jugador 2: Conectado", "Jugador 3: Conectado". Se le notificará a los Jugadores 1 y 2 que el Jugador 3 se ha conectado.	

Figura 3.3. Tres jugadores conectados

Tabla 3.4. Prueba de aceptación # 4

Caso de prueba de aceptación	
Código: HU2_P1	**Historia de usuario:** 2
Nombre: Juega un jugador	
Descripción: Con el objetivo de probar que el cliente y el servidor están configurados de forma correcta, es necesario abrir el juego y jugar entre dos o tres jugadores, de forma tal que al producirse una jugada por parte de uno de ellos, se envíe un mensaje al servidor con la posición actual de ese jugador y este enviarlo de regreso a todos los clientes conectados para que actulicen la posición de ese jugador.	
Condiciones de ejecución: • El servidor debe de estar ejecutándose. • Debe haber al menos dos jugadores conectados.	
Pasos de ejecución: • Abrir el archivo principal del juego *index.html*. • Dar clic en el botón “Jugar”. • Dar clic en el botón “Comenzar”. • Dar clic en el botón “Tirar”.	
Resultados esperados: Debe mostrarse en cada uno de los clientes el avance o retroceso del jugador desde su posición hasta la casilla que le corresponde.	

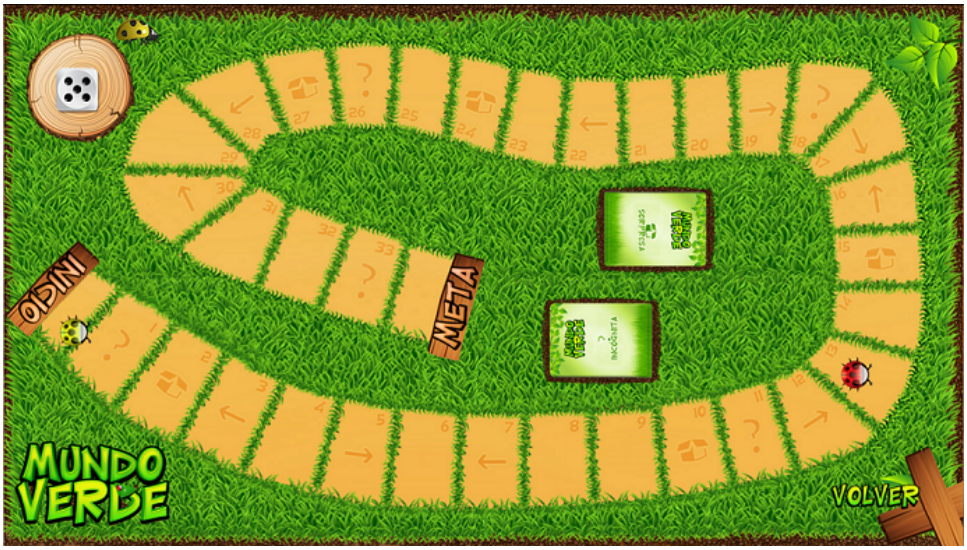

Figura 3.4. Visión del jugador 1

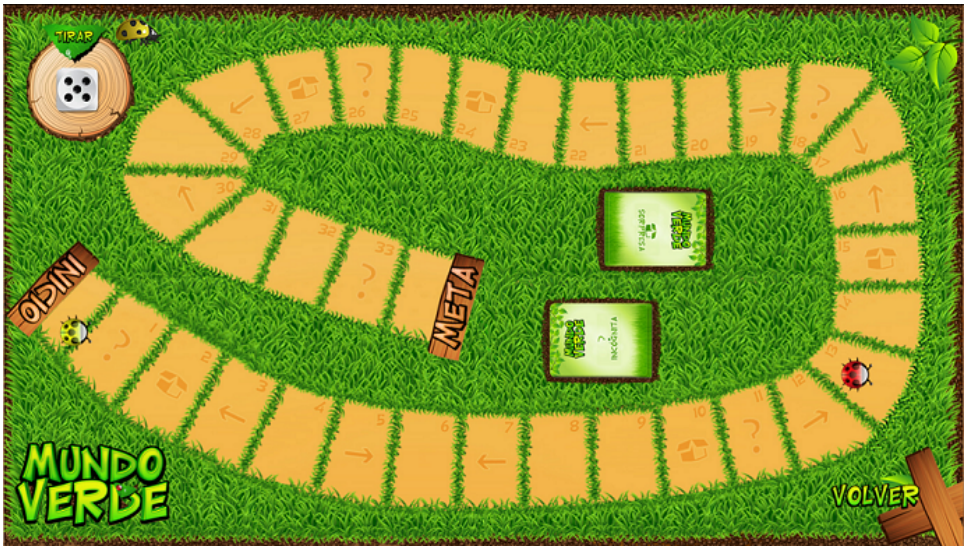

Figura 3.5. Visión del jugador 2

Tabla 3.5. Prueba de aceptación # 5

Caso de prueba de aceptación	
Código: HU2_P2	**Historia de usuario:** 2
Nombre: Se conecta un nuevo jugador	
Descripción: Con el objetivo de probar que el cliente y el servidor están configurados de forma correcta, es necesario abrir el juego y jugar entre dos o tres jugadores, de forma tal que una vez conectado el primer jugador, al conectarse otro jugador, se le notifica al cliente que hay un segundo o tercer jugador conectado.	
Condiciones de ejecución: • El servidor debe de estar ejecutándose. • Debe haber al menos dos jugadores conectados.	
Pasos de ejecución: • Abrir el archivo principal del juego *index.html*. • Dar clic en el botón “Jugar”. • Dar clic en el botón “Comenzar”.	
Resultados esperados: Debe mostrarse en cada cliente una notificación de que el Jugador 2 o el Jugador 3 se ha conectado.	

Figura 3.6. Nuevo jugador conectado

Tabla 3.6. Prueba de aceptación # 6

Caso de prueba de aceptación	
Código: HU2_P3	**Historia de usuario:** 2
Nombre: Se desconecta un jugador.	
Descripción: Con el objetivo de probar que el cliente y el servidor están configurados de forma correcta, una vez que están jugando dos o tres jugadores y uno de ellos se desconecta, es necesario mostrar a los restantes jugadores que existe un jugador que se acaba de desconectar. En este caso la partida se debe reiniciar.	
Condiciones de ejecución: • El servidor debe de estar ejecutándose. • Debe haber al menos dos jugadores conectados.	

Continúa en la próxima página

Tabla 3.6. Continuación de la página anterior

Pasos de ejecución: • Abrir el archivo principal del juego *index.html*. • Dar clic en el botón “Jugar”. • Dar clic en el botón “Comenzar”. • Dar clic en el botón “Tirar”. • Un jugador debe desconectarse.
Resultados esperados: Debe mostrarse en cada cliente una notificación de que un jugador se ha desconectado.

Figura 3.7. Jugador desconectado

Tabla 3.7. Prueba de aceptación # 7

Caso de prueba de aceptación	
Código: HU2_P4	**Historia de usuario:** 2
Nombre: Notificación de turno de juego	

Continúa en la próxima página

Tabla 3.7. Continuación de la página anterior

Descripción: Con el objetivo de probar que el cliente y el servidor están configurados de forma correcta, una vez que le corresponde el turno aun jugagador el sistema debe mostrar un mensaje de notificaion “Es tu turno: Juega”.
Condiciones de ejecución: • El servidor debe de estar ejecutándose. • Debe haber al menos dos jugadores conectados.
Pasos de ejecución: • Abrir el archivo principal del juego *index.html*. • Dar clic en el botón “Jugar”. • Dar clic en el botón “Comenzar”. • Dar clic en el botón “Tirar”.
Resultados esperados: Se debe mostrar en cada cliente una notificación, cuando sea el turno de un jugador.

Figura 3.8. Turno de un jugador

Tabla 3.8. Prueba de aceptación # 8

Caso de prueba de aceptación	
Código: HU2_P5	**Historia de usuario:** 2
Nombre: Se conecta un usuario una vez iniciada la partida.	
Descripción: Con el objetivo de probar que el cliente y el servidor están configurados de forma correcta, una vez que están jugando dos o tres jugadores, y existe otro jugador que desea conectarse, el sistema muestra un mensaje que le dice que el servidor ya está lleno, por lo que debe esperar a que exista un lugar disponible en el servidor para que él se pueda conectar.	
Condiciones de ejecución: • El servidor debe de estar ejecutándose. • Debe haber al menos dos jugadores conectados. • La partida debe haber comenzado.	
Pasos de ejecución: • Abrir el archivo principal del juego *index.html*. • Dar clic en el botón “Jugar”. • Dar clic en el botón “Comenzar”. • Dar clic en el botón “Tirar”. • Un jugador debe conectarse.	
Resultados esperados: Se debe mostrar una notificación que le informe al usuario que el servidor se encuentra lleno en ese momento.	

Figura 3.9. Usuario conectado una vez iniciada la partida

Tabla 3.9. Prueba de aceptación # 9

Caso de prueba de aceptación	
Código: HU2_P6	**Historia de usuario:** 2
Nombre: Abrir nueva instancia del juego en el navegador.	
Descripción: Con el objetivo de probar que el cliente y el servidor están configurados de forma correcta, si un usuario desea abrir desde el mismo navegador dos pestañas del videojuego, el sistema muestra un mensaje que le dice que ya tiene una instancia abierta en ese navegador.	
Condiciones de ejecución: • El servidor debe de estar ejecutándose. • Debe haber al menos dos jugadores conectados.	
Pasos de ejecución: • Abrir el archivo principal del juego *index.html*. • Abrir el archivo principal del juego *index.html* en una nueva pestaña del navegador.	
Resultados esperados: Se debe mostrar una notificación que le informe al usuario que tiene abierta otra instacia de juego en otra pestaña del navegador.	

Figura 3.10. Nueva instancia del juego en el navegador

Tabla 3.10. Prueba de aceptación # 10

Caso de prueba de aceptación	
Código: HU3_P1	**Historia de usuario:** 3
Nombre: Enviar resultados de la partida a la base de datos	
Descripción: Con el objetivo de probar que existe conexión entre el servidor y la base de datos, es necesario abrir el juego y realizar una partida, de modo que al finalizarla el servidor reciba los nombres que identifican a cada jugador y el lugar obtenido concluida la partida. Estos datos deben ser enviados a la base de datos.	
Condiciones de ejecución: • El servidor debe de estar ejecutándose. • Debe haber al menos dos jugadores conectados. • La partida debe haber finalizado. • La base de datos debe existir.	

Continúa en la próxima página

Tabla 3.10. Continuación de la página anterior

Pasos de ejecución: • Abrir el archivo principal del juego *index.html*. • Dar clic en el botón “Jugar”. • Dar clic en el botón “Comenzar”. • Dar clic en el botón “Tirar”. • Repetir el paso anterior hasta que haya un ganador.
Resultados esperados: Deben mostrarse los datos guardados en la base de datos.

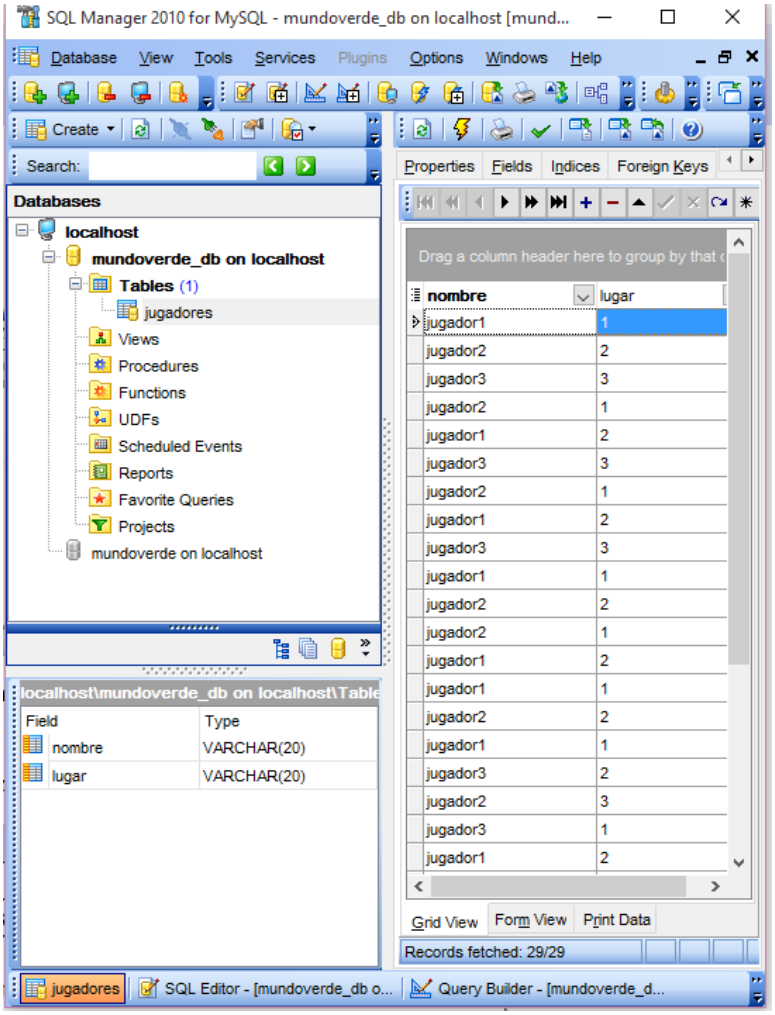

Figura 3.11. Resultados almacenados en la base de datos.

Tabla 3.11. Prueba de aceptación # 11

Caso de prueba de aceptación	
Código: HU4_P1	**Historia de usuario:** 3
Nombre: Generar archivo *.json* con los datos almacenados en la base de datos	
Descripción: Para que la integración sea correcta, se debe proveer a la plataforma de una carpeta que contiene todo el videojuego. Es necesario además, generar un archivo con el nombre *salva.json* que contiene los datos almacenados en la base de datos, y que se encuentra dentro de la carpeta que contiene el videojuego.	
Condiciones de ejecución: • El servidor debe de estar ejecutándose. • Debe haber al menos dos jugadores conectados. • La partida debe haber finalizado. • La base de datos debe existir. • Es generado el archivo *salva.json*.	
Pasos de ejecución: • La plataforma debe cargar el archivo *salva.json*.	
Resultados esperados: Debe cargarse el archivo *salva.json* con los datos almacenados en la base de datos..	

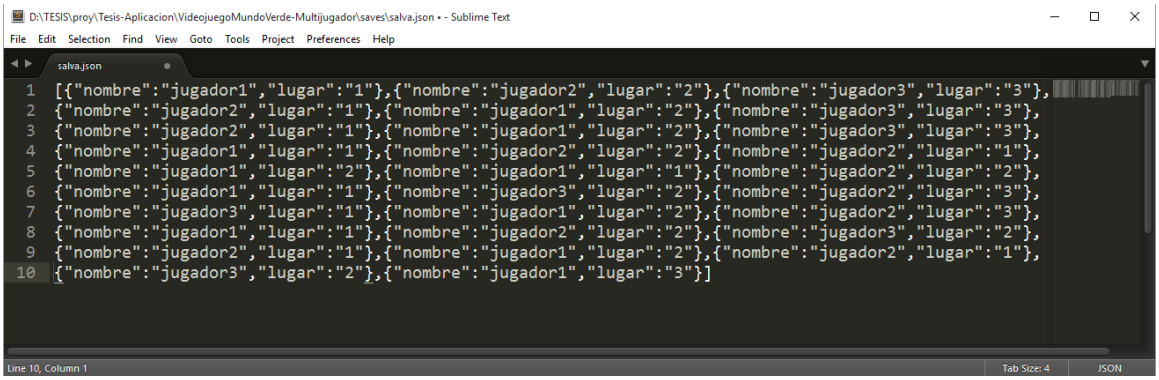

Figura 3.12. Representación del fichero *salva.json* con los resultados de las partidas.

Luego de haber realizado las pruebas de aceptación en cada una de las iteraciones con sus correspondientes casos de prueba, se obtuvieron los siguientes resultados:

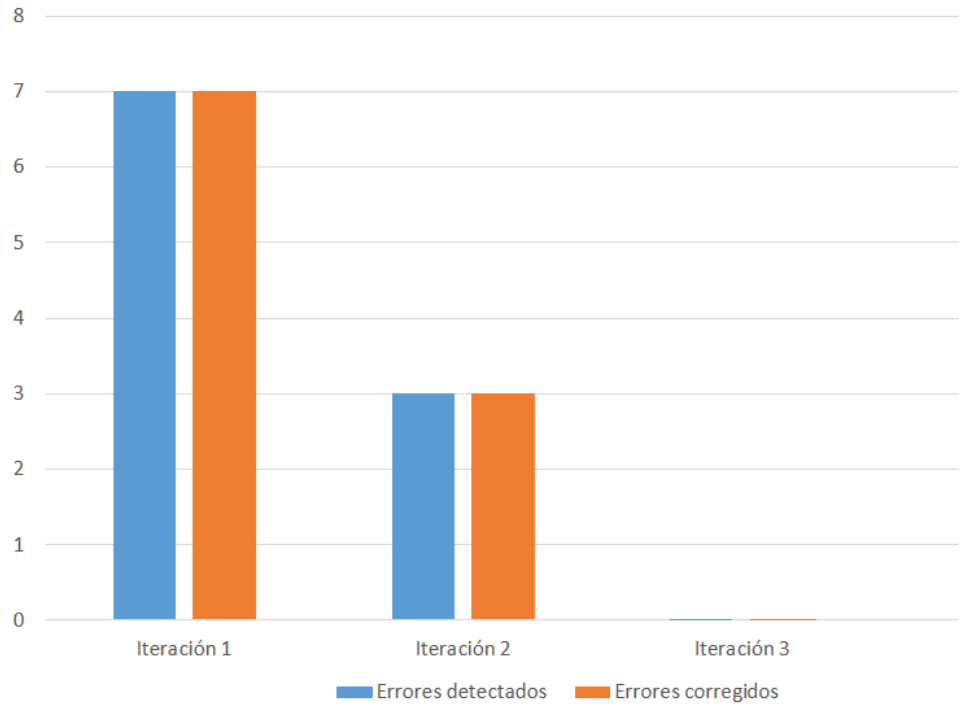

Figura 3.13. Pruebas de aceptación

3.2.2. Pruebas de rendimiento

Para un mayor conocimiento de cómo se comportó el servidor en cuanto a rendimiento, se realizaron pruebas para medir los tiempos de respuesta de los mensajes entre el cliente y el servidor. Estas fueron realizadas en el momento más crítico de juego, específicamente cuando los clientes se conectan. La herramienta seleccionada fue *Firebug*, una extensión del navegador *web Mozilla FireFox* que permite editar, depurar y monitorear el *CSS*, *HTML* y *JavaScript* de cualquier página *web*. A continuación, un grupo de imágenes que muestran el comportamiento del servidor para cada uno de los clientes. En este caso se seleccionaron tres clientes por ser el número máximo de conexiones que permite el juego.

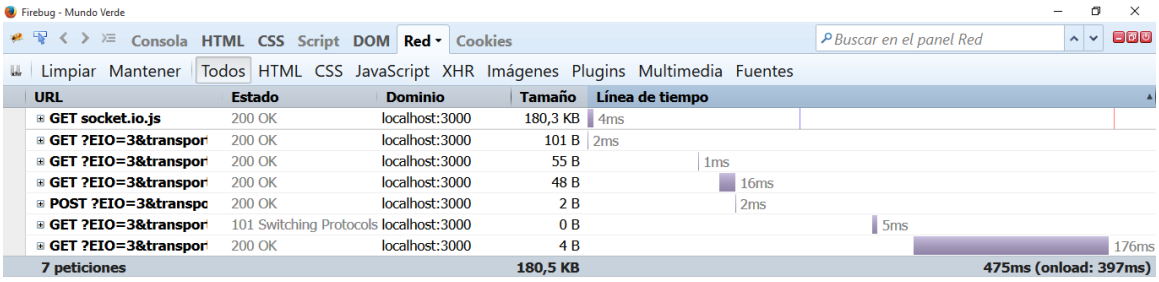

Figura 3.14. Tiempo de respuesta para el cliente 1.

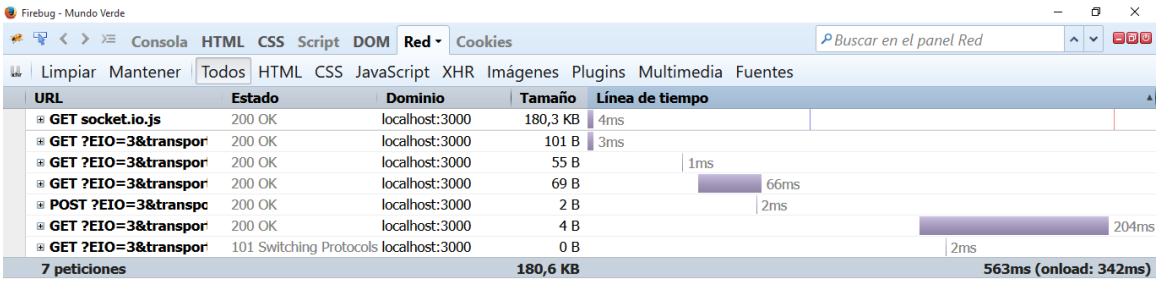

Figura 3.15. Tiempo de respuesta para el cliente 2.

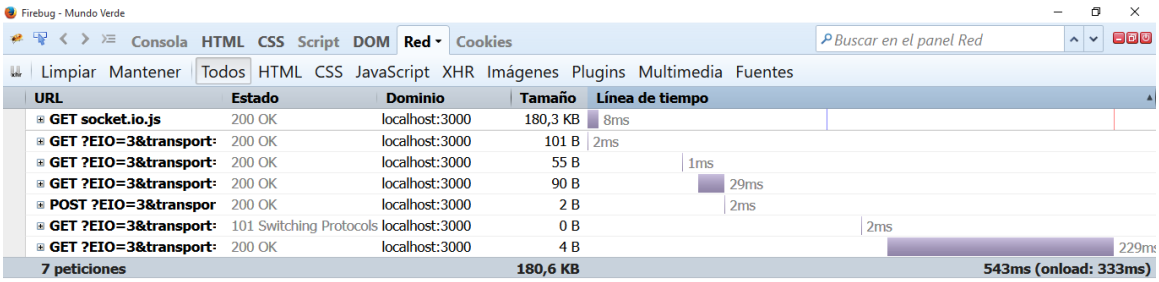

Figura 3.16. Tiempo de respuesta para el cliente 3.

Los resultados arrojados muestran el alto potencial de la tecnología seleccionada para la construcción del servidor y la implementación del protocolo de comunicación. Es válido mencionar que el número de conexiones simultáneas al servidor, debido a las características que presenta el videojuego, es muy pequeño, ya que solo permite tres conexiones.

3.2.3. Validación del componente

La validación es el proceso en el cual se verifica que el sistema cumple con todos los requerimientos que el cliente define. Una vez terminadas las pruebas de aceptación y de rendimiento, las cuales arrojaron resultados satisfactorios, se verificó que el componente desarrollado permite que varios usuarios se puedan conectar desde diferentes estaciones de trabajo, para jugar de forma *online* el videojuego de navegador Mundo Verde. Además, permite generar el archivo *salva.json* que contiene los principales datos de cada partida

como son el nombre de cada jugador y el lugar que obtuvo en el juego. Una vez concluido este proceso, la Plataforma puede cargar satisfactoriamente el archivo comprimido del juego, permitiendo su integración con el videojuego.

Consideraciones finales del capítulo

Tras la implementación y pruebas del sistema han quedado satisfechos todos los requerimientos definidos por el cliente. Para ello fue de gran utilidad cada uno de los artefactos generados en capítulos anteriores. Una vez concluidas las pruebas, se dieron por concluidas las iteraciones del proceso de desarrollo.

Conclusiones

Al término de la presente investigación se concluye que el componente de interconexión *online* del videojuego de navegador Mundo Verde:

- Permite a los jugadores interactuar en una misma partida desde diferentes estaciones de trabajo, ubicadas en diferentes puntos de la red.
- La utilización del protocolo de comunicación *WebSocket* permite que los tiempos de respuesta entre el videojuego y el servidor sean relativamente pequeños, existiendo un alto rendimiento en el sistema.
- El componente desarrollado puede ser reutilizado en aplicaciones similares, específicamente en aquellas que estén basadas en la arquitectura cliente-servidor y que sean aplicaciones que se ejecutan en tiempo real, donde los tiempos de respuestas de los mensajes es lo primordial.

Recomendaciones

Para dar continuidad al presente trabajo se recomienda:

- Crear diferentes salas de juegos para los jugadores de modo que el servidor permita mayor número de conexiones simultáneas.
- Permitir en una versión futura que cuando uno de los jugadores se desconecte los demás puedan seguir en la partida.
- Permitir que cada uno de los jugadores al conectarse al juego, pueda crear su propio perfil, con el objetivo de que los jugadores se sientan identificados, logrando una mejor organización de los datos almacenados.

Siglas

AUP Proceso Unificado Ágil. 17

CRC Clase, Responsabilidad y Colaboración. 27, 31

HTTP *Hypertext Transfer Protocol*. 7, 8, 13

MMOFPSG *Massively Multiplayer Online First Person Shooter Games*. 4
MMOG *Massively Multiplayer Online Games*. 4, 5
MMORPG *Massively Multiplayer Online Role Playing Games*. 5
MMORTEG *Massively Multiplayer Online Real Time Estrategy Games*. 5

NPM *Node Package Manager*. 15

RUP Proceso Racional Unificado. 17

SGBD Sistema Gestor de Bases de Datos. 8, 20

TCP *Transmission Control Protocol*. 7, 8

UCI Universidad de las Ciencias Informáticas. 1, 8

XP Programación Extrema. 18, 19, 21, 27, 32–34

Referencias bibliográficas

ALEGSA, Leandro. 2008. *DICCIONARIO DE INFORMÁTICA Y TECNOLOGÍA , Definición de interconexión*. 2008. Dirección: ⟨http://www.alegsa.com.ar/Dic/interconexion.php⟩.

BASHIR, M Salman y QURESHI, M Rizwan Jameel. 2012. Hybrid Software Development Approach For Small To Medium Scale Projects: Rup, Xp & Scrum. *Cell*. 2012, vol. 966, págs. 536474921.

CAMACHO, Jorge Hernández. 2015. Mundos Virtuales. 2015.

CANÓS, José H; LETELIER, Patricio y PENADÉS, Mª Carmen. 2003. Metodologías ágiles en el desarrollo de software. *Universidad Politécnica de Valencia, Valencia*. 2003.

CANTELON, Mike; HARTER, Marc; HOLOWAYCHUK, TJ y RAJLICH, Nathan. 2014. *Node. js in Action*. 2014.

CASTRELO CID, Alejandro. 2014. MMO de navegador en tiempo real con Node. js y WebSockets. 2014.

CORDERO, Jorge Luis. 2010. METODOLOGIAS AGILES PROCESO UNIFICADO AGIL (AUP). 2010.

DINGSOYR, Torgeir; NERUR, Sridhar; BALIJEPALLY, VenuGopal y MOE, Nils Brede. 2012. A decade of agile methodologies: Towards explaining agile software development. *Journal of Systems and Software*. 2012, vol. 85, n.º 6, págs. 1213-1221.

DUARTE, Manuel Palomo. 2011. Programacion en PHP a través de ejemplos. 2011.

ESCOBAR, Julio; URANGO, Alejandro; CASTRO, Libardo et al., 2015. Gestión de discusiones y notificaciones en el espacio virtual como base para el trabajo con comunidades de práctica en el contexto educativo. 2015.

FLANAGAN, David. 2006. *JavaScript: the definitive guide*. 2006.

GARCÍA PEÑALVO, Francisco José. 1998. Patrones. De Alexander a la Tecnología de Objetos. 1998.

GREGORIO, Jose. 2011. Servidor Web. 2011. Url: ⟨http://www.slideshare.net/josegregoriob/servidor-web-8451426⟩.

GUTIERREZ, E. 2015. *Plataforma web para la gestión de videojuegos serios de navegador con fines terapéuticos (Medicando)*. 2015. Url: ⟨http://publicaciones.uci.cu/index.php/SC/article/view/1742⟩.

ÍSCAR MARTÍNEZ, JUAN. 2015. *NODE. JS Do's and Don'ts*. 2015.

KNAUS, William A; DRAPER, Elizabeth A; WAGNER, Douglas P y ZIMMERMAN, Jack E. 1985. APACHE II: a severity of disease classification system. *Critical care medicine*. 1985, vol. 13, n.º 10, págs. 818-829.

LARMAN, Craig. 2005. *Applying UML and patterns: an introduction to object-oriented analysis and design and iterative development*. 2005.

LETELIER, Patricio. 2006. Metodologías ágiles para el desarrollo de software: eXtreme Programming (XP). 2006.

MUÑOZ DE LA TORRE, Arturo. 2013. Introducción a Node.JS a través de Koans. 2013.

MYSQL, AB. 2001. *MySQL*. 2001.

NIXON, Robin. 2012. *Learning PHP, MySQL, JavaScript, and CSS: A step-by-step guide to creating dynamic websites*. 2012.

ORACLE. 2016a. *10 razones para elegir MySQL para las aplicaciones web de la próxima generación*. 2016. Dirección: ⟨`http://www.mysql.com/why-mysql/white-papers/10-razones-para-elegir-mysql-para-las-aplicaciones-web-de-la-proxima-generacion/`⟩.

ORACLE. 2016b. *Las 10 razones principales para usar MySQL como base de datos integrada*. 2016. Dirección: ⟨`http://www.mysql.com/why-mysql/white-papers/las-10-razones-principales-para-usar-mysql-como-base-de-datos-integrada/`⟩.

ORNBO, George. 2012. *Sams Teach Yourself Node. js in 24 Hours*. 2012.

PIMENTEL, Victoria y NICKERSON, Bradford G. 2012. Communicating and displaying real-time data with WebSocket. *Internet Computing, IEEE*. 2012, vol. 16, n.º 4, págs. 45-53.

PRESSMAN, Roger. 2005. *Ingeniería de Software. Un enfoque práctico*. 2005.

PUJOL, Roylandi González. 2012. Tablero de cotizaciones en un entorno de tiempo real usando el marco de trabajo jWebSocket. 2012.

RADATZ, Jane; GERACI, Anne y KATKI, Freny. 1990. IEEE standard glossary of software engineering terminology. *IEEE Std*. 1990, vol. 610121990, n.º 121990, págs. 3.

RAI, Rohit. 2013. *Socket. IO Real-time Web Application Development*. 2013.

RICHARDS, Gregor; LEBRESNE, Sylvain; BURG, Brian y VITEK, Jan. 2010. An analysis of the dynamic behavior of JavaScript programs. En. *ACM Sigplan Notices*. N.º 6, 2010, págs. 1-12.

SOMMERVILLE, Ian y GALIPIENSO, María Isabel Alfonso. 2005. *Ingeniería del software*. 2005.

STEINKUEHLER, Constance A. 2004. Learning in massively multiplayer online games. En. *Proceedings of the 6th international conference on Learning sciences*. 2004, págs. 521-528.

TANENBAUM, Andrew S. 2003. *Redes de computadoras*. 2003.

TILKOV, Stefan y VINOSKI, Steve. 2010. Node. js: Using javascript to build high-performance network programs. *IEEE Internet Computing*. 2010, vol. 14, n.º 6, págs. 80.

TIWARI, Anurag; PRAVEEN, Patel y MANUJ, Darbari. 2013. Global Prospect of Distributed Agile Software Development: A Review. *UACEE International Journal of Advances in Computer Science and its Applications*. 2013, vol. 3: Issue 2, n.º ISSN 2250-3765.

VALDÉS, Yahima Vigo. 2012. SUBSISTEMA PARA LA SINCRONIZACIÓN DE INFORMACIÓN EN TIEMPO REAL EN LOS SISTEMAS DE GESTIÓN DE EMERGENCIAS CIUDADANAS. 2012.

VAN DAMME, Bramus y LINDE, Rogier van der. 2014. Por qué a los desarrolladores de juegos debería interesarles HTML5. *Cuatro décadas de Novática>02 en resumen Nuestra"niña bonita"se hace mayor>02 Llorenç Pagés Casas noticias de IFIP*. 2014, n.º 230, págs. 25.

VANHATUPA, Juha-Matti. 2010. BROWSER GAMES FOR ONLINE COMMUNITIES. 2010.

VANHATUPA, Juha-Matti. 2013. On the Development of Browser Games–Current Technologies and the Future. *Tool Support for Computer Role-Playing Game Programming: Foundations, Guidelines and Applications*. 2013.

VERA, Pablo M; MORENO, Edgardo J; RODRÍGUEZ, Rocío Andrea et al., 2015. Empleo de aprendizaje basado en juegos y técnicas de gamificación en el ámbito universitario. En. *XVII Workshop de Investigadores en Ciencias de la Computación (Salta, 2015)*. 2015.

VILARDELL, Ignasi Barri. 2012. Modelo y Planificación de Aplicaciones de Juegos Masivos Multijugador en Red en Entornos Distribuidos. 2012.