SISTEMAS DISTRIBUIDOS I: DOCUMENTO DE DISEÑO DE SISTEMA (SDD)

Proyecto: Asymetric Puzzle

INTEGRANTES:

Julia Alejandra Rodríguez Abud - Líder de proyecto y Diseñador de software

Carlos Cárdenas Ruiz– Desarrollador de software

Gustavo Alfonso Orozco Velázquez – Analista de Software

STAKEHOLDERS:

Dr. Félix Francisco Ramos Corchado

16 DE ABRIL DE 2021

MERCURIO SYSTEMS

CINVESTAV Unidad Guadalajara

Prefacio

Se espera que este documento proporcione la información suficiente para el diseño y la implementación de una aplicación distribuida que utiliza el modelo cliente servidor para proporcionar un servicio en un videojuego cooperativo enfocado en la colaboración y el trabajo conjunto entre usuarios, para la solución de problemas dentro de un modelo de vista isométrica.

Este documento será utilizado principalmente por:

- Clientes del sistema.
- Arquitecto de software.
- Ingeniero administrador de proyecto.
- Ingeniero de implementación del sistema.
- Ingeniero de pruebas del sistema.
- Ingeniero de mantenimiento del sistema.

HISTORIAL DEL DOCUMENTO			
Fecha	Versión	Comentarios	Autor
28 marzo 2021	0.1	Versión inicial.	Gustavo Orozco
28 marzo 2021	0.1	Revisada por el equipo.	Julia Abud
15 abril 2021	1.0	Agregado de especificaciones.	Gustavo Orozco
15 abril 2021	1.0	Revisada por el equipo.	Julia Abud
16 abril 2021	1.0	Versión final.	Dr. Félix Ramos

Contenido

Introducción

Este documento analiza el aspecto del diseño y modelado de un sistema que utiliza un entorno virtual para el trabajo colaborativo dentro de un mundo conectado. El objetivo de este proyecto está enfocado en la necesidad de unir un conjunto de fuerzas en beneficio de resolver y tratar problemas que afectan a los agentes dentro de un entorno de simulación.

El diseño propone atributos de calidad enfocados a la persistencia y gestión de los datos en el servidor, eficiencia, usabilidad y tolerancia a fallas como atributos de calidad de alta prioridad que se espera como respuesta del sistema.

Resumen del documento

Los objetivos de diseño específicos están contenidos en este documento, así como la estructura arquitectónica que espera ser implementada; además de identificar cualidades de un patrón de diseño con las características de diseño del sistema propuesto. El patrón de diseño arquitectónico permite modelar la arquitectura apropiada específicamente a este requerimiento mediante un modelo para separar la comunicación de los datos dentro del sistema; así como persistencia y reacción a eventos. Para el diseño de pruebas, se establece la descomposición en subsistemas (componentes), así como web services permanentes en respuesta a una mejor implementación y mantenibilidad del sistema distribuido.

1. Propósito

Este documento describe los requerimientos de software del sistema para el desarrollo de un videojuego cooperativo multijugador en vista isométrica con el objetivo principal de utilizar los recursos y capacidades del jugador para resolver problemas dentro de un entorno de simulación.

1.1 Alcance del proyecto

La aplicación se desarrolla utilizando tecnologías que pueden ejecutarse y adaptarse en dispositivos con acceso a internet con el uso de una arquitectura cliente servidor que soporte la conexión e interacción de varios jugadores dentro de un videojuego de no suma cero donde los usuarios colaboran para un bien común con otros usuarios, fomentando la relación de compañerismo y el uso de capacidades y recursos en un entorno de simulación.

Alcance:	Este documento de requerimientos de software es la base del desarrollo de software del proyecto. Describe los siguientes tópicos: - Requerimientos del usuario. - Requerimientos del sistema. - Requerimientos funcionales. - Requerimientos no funcionales. - Requerimientos de dominio.

2. Referencias:

IEEE Estándar 1016-1998: Práctica recomendada de IEEE para el diseño de sistema de software. IEEE, 1998.

Antes de leer este documento de diseño del sistema (SDD), se recomienda leer el documento de análisis de requerimiento (SRS), que proporciona todos los requerimientos funcionales, requerimientos no funcionales y de dominio descritos sobre este sistema.

Para obtener más información sobre las herramientas de desarrollo Node.js y Unity, es posible consultar las páginas web oficiales y su respectiva documentación; la cual describe en detalle las estructuras de datos, así como las tecnologías implementadas para este desarrollo de software a la medida.

3. Definiciones

Web Socket: Es un protocolo de comunicación full-dúplex de una sola conexión permanente, que permite realizar Stream de mensajes y transferencia de contenido en tiempo real.

Agente: Un sistema cooperativo está formado por un conjunto de agentes (jugadores) comunicándose y que trabajan para lograr un objetivo global o un trabajo común.

RPC: Llamadas a procedimientos remotos realizadas al servidor.

RMI: Comunicación entre objetos distribuidos, extensión del modelo de programación de objetos (POO), de invocación local de un método conocido hacia invocación remota.

Servicio: Web services consumido mediante la comunicación durante la ejecución de la aplicación.

Modelo Cliente servidor: La arquitectura cliente-servidor es un modelo de diseño de software en el que las tareas se reparten entre los proveedores de recursos o servicios, llamados servidores, y los demandantes, llamados clientes.

Middleware: Módulos “plug and play” que se pueden apilar arbitrariamente en cualquier orden y provén cierta funcionalidad.

Node.js: Es un controlador de tareas que permite a JavaScript ejecutarse del lado del servidor es un entorno de programación construido sobre el motor Chrome V8.

Treads: Los hilos son accesos al sistema de administración de memoria de un dispositivo para la gestión del consumo y la prioridad de ejecución de una operación que se ejecuta durante el procesamiento de información de la aplicación.

Usuario: Persona (jugador) que ingresa al sistema.

No suma cero: Sistema desarrollado en base a la teoría de juegos cooperativos donde todos los participantes de la partida colaboran entre sí para obtener resultados en igualdad de condiciones, dando como resultado que todos ganen o todos pierdan.

Unity: Software para el desarrollo de proyectos en entornos gráficos, interfaz de usuario, game object, componentes, capas y escenas; Así como recursos Assets y prefabs para la construcción y parametrización de videojuegos.

Especificación de requisitos de software (SRS): Un documento que describe las funciones de un sistema propuesto y las restricciones bajo las cuales debe operar.

Parte interesada (Stakeholder): Cualquier persona interesada en el proyecto que no sea desarrollador.

Lobby: Sala de espera en una sesión multijugador donde se realiza la interconexión de usuarios dentro de la aplicación.

4. Propósito del sistema

El propósito de este sistema es proporcionar un entorno de desarrollo mediante la plataforma Unity con servicio a componentes, el uso de un middleware que permite la interacción con soporte multijugador en un entorno de simulación, la solución de problemas con el uso de recursos compartidos y fomentar el comportamiento colaborativo simulado. El usuario accede a la aplicación para generar una partida y utilizar una conexión a internet para acceder a todas las funciones y operaciones del sistema de juego. El rol de usuario jugador tendrá acceso a determinadas funciones multijugador dentro del sistema.

4.1 Objetivo de diseño

El diseño de la arquitectura y estructura del sistema está pensado de acuerdo con las características abstraídas de los requerimientos del usuario en términos de tiempo de respuesta, la persistencia en la gestión de los datos que permita el cálculo de partidas a partir de la posición geográfica del usuario, así como iniciar sesión en el sistema, configurar y actualizar información en tiempo real desde un dispositivo que consulta a un servidor.

Se espera que el sistema mantenga una comunicación constante con el usuario y servicios web mediante un middleware, manteniendo de manera persistente una conexión permanente RPC durante la ejecución de la aplicación.

El sistema debe ser preciso en solicitar peticiones dentro de la aplicación y desplegar los componentes en la interfaz de usuario; la generación de la partida asigna al jugador que la crea como coordinador para sincronizar la partida multijugador.

El servidor utiliza llamadas RPC al servidor, el servidor debe responder de manera óptima y notificar pertinentemente al usuario cuando ocurren problemas para establecer conexión de acuerdo con las especificaciones del cliente.

La información proporcionada por el usuario dentro del sistema será registrada con la finalidad de utilizar los datos generados permitiendo una mejor comprensión en el estudio del comportamiento durante la sesión, esto con la finalidad de mejorar la experiencia de juego.

4.2 Rendimiento

La eficiencia está enfocada en el desempeño al momento de consultar la base de datos distribuida, se espera una conexión permanente síncrona que permita el consumo web services para realizar operaciones en tiempo real; el sistema gestiona el control de los datos mediante un middleware que administra los eventos a servicios que consumen recursos RPC y de obtener páginas del sistema en el dispositivo mediante un servidor.

El uso de información guardada en memoria cache, como son las páginas y librerías son limitados por la capacidad del dispositivo.

Estrategia:

Se espera que el sistema despliegue en tiempo real la información al usuario dentro de la interfaz para sincronizar la conexión con tolerancia a fallos de desconexión.

Táctica:

- Demanda de recursos: El rendimiento de la aplicación se basa en separar en pasos necesarios para la ejecución de componentes RMI y rutinas de conexión dentro de la aplicación orientado a eventos para procesar los subsistemas en el middleware.
- Gestión de recursos: La prioridad en los recursos está relacionado con reflejar el estado de los recursos mostrados al usuario, optimizando la concurrencia a los procesos y componentes dentro de la partida pensando en que los recursos son limitados dentro del dispositivo.
- Reducción de sobrecarga: Cada tarea está cubierta por subsistemas encargados de manejar procesos concretos que responden a actividades específicas de manera redundante, optimizando la consulta y lectura al servidor, así como consultas a servicios externos consumiendo los recursos o componentes necesarios y suficientes para la ejecución de procesos.
- Manejo de tasa de eventos: Cada evento es controlado mediante un bus de eventos dentro del middleware, los eventos emitidos por el subsistema se ejecutan agrupando en componentes específicos reduciendo la carga del componente que consume.
- Tolerancia a fallos: El consumo al web services recibe y procesa los eventos mediante web sockets al momento de ingresar al lobby multijugador, este calcula a los usuarios conectados y establece conexión coordinando y emparejando a los jugadores en una partida que se mantiene conectada, esto se toma como una transacción coherente en tiempo real de información entre los jugadores, los componentes y los recursos.

4.3 Usabilidad

Se espera sea accesible al usuario mediante un diseño de interfaz intuitivo para desplazarse dentro de la interfaz en vista isométrica dentro del mapa para interactuar con su entorno en la partida dentro del sistema; esta aplicación se relaciona con los conceptos de dominio y acciones que contextualizan las interacciones en el sistema.

El usuario puede operar el sistema con un máximo de tres pasos:

- Paso 1: Iniciar nombre de usuario para ingresar al sistema.
- Paso 2: El usuario ingresa al multijugador en el sistema.
- Paso 3: Iniciar partida con otros usuarios emparejados.

El sistema se encarga de controlar la funcionalidad de acuerdo con el modelo arquitectónico.

En primera instancia el sistema será accesible con controladores a medida con el uso del middleware, funcionalidades de otros servicios que soporte llamadas RMI de asistencia a usuario o componentes propios del dispositivo, así como desarrollos de terceros: como es reconocimiento de puertos, detectar dispositivos conectados, parámetros de accesibilidad y otras ayudas visuales. Otras propuestas de accesibilidad pueden ser implementadas en próximas versiones.

La información mostrada al usuario será en idioma español (México) para uso coherente que refleje el sistema en la interfaz de usuario.

Estrategia:

La curva de aprendizaje es mínima, ya que el usuario puede reconocer de otros videojuegos interfaces e ítems a los cuales puede acceder, ingresar a una partida creada; iniciar y cancelar una partida en curso y notificar a los usuarios conectados en cualquier momento disponibles en la interfaz de usuario.

Táctica:

- El objetivo de la usabilidad es permitir brindar información y asistencia adecuada al usuario, la interfaz (cliente) del sistema está separada las llamadas del controlador y páginas del modelo de datos distribuido.
- El contexto de la interfaz del mapa dentro de la partida permite que la iniciativa del usuario sea coherente con el despliegue de información dentro de la vista del usuario.
- El usuario tiene total control en la operación de la aplicación, permitiendo cancelar partidas, deshacer los datos guardados o cargar parámetros de lugares en el entorno de simulación ingresados por el usuario en la memoria interna del sistema y consultas externas RPC.

4.4 Confiabilidad

La recuperabilidad y tolerancia a fallos que se espera como respuesta del sistema, permite un modelo que funcione en un servidor y separe el controlador de la vista mediante un middleware al que consulta el usuario.

En caso de No encontrar un servidor o jugadores conectados ejecutar excepciones.

El tiempo real es el tiempo medio de respuesta esperado del sistema, la resiliencia a fallos depende de los servicios externos al sistema, se espera que la resiliencia a desconexión permita que el mapa siga respondiendo en todo momento.

Estrategia:

La tolerancia a fallos es un equilibrio entre los módulos de gestión dentro del modelo de diseño, donde el servidor requiere una mayor eficiencia controlando las conexiones y el consumo de servicios; evitando en lo posible que dos procesos críticos como lo es el cálculo de acciones y el mapa se despliegue soportando el servicio, al ser detectado un fallo se reporta y registra para su corrección en la fase de pruebas del sistema.

Táctica:

- Las acciones de autenticación validan al usuario creando una sesión para acceder al sistema.
- Las directivas de autorización de sesión se validan de acuerdo con el servidor que conecta al usuario.
- La confidencialidad de los datos garantiza que los datos sólo los puede ver quien tiene autorizado.
- Garantizar la integridad compuesta por los datos, limitar su exposición a personas externas. Limitando el acceso a la información y separando la información de usuario del uso de los subsistemas de la aplicación.
- El sistema realiza un respaldo de la información mensualmente.
- En caso de ataques informáticos, el sistema puede recuperarse restaurando una copia de respaldo del sistema para que este vuelva a estar disponible.

4.5 Mantenibilidad

La mantenibilidad de este sistema permitirá una estructura con capacidad de operaciones orientadas a eventos previsto de proveer una misma respuesta a una solicitud del usuario.

El sistema será implementado en tecnologías de desarrollo accesibles para desarrolladores con el conocimiento de Node.js y Unity.

El sistema está dispuesto en módulos (subsistemas) con capacidad de separar en componentes independientes pero acoplados en una única versión en producción para la cobertura de pruebas y análisis con el uso de herramientas CASE; la arquitectura de diseño provee la modularidad del sistema, cada módulo tiene la capacidad de dar funcionalidad específica y modificar el sistema de acuerdo a las dependencias de otros subsistemas enfocados en realizar procesos a criterios específicos en base a los requerimientos solicitados por el cliente.

Estrategia:

El sistema permite la escalabilidad e incorporación de nuevas funcionalidades al sistema, que complementen las características funcionales actuales.

Táctica:

- Las dependencias a los componentes están definidas de forma que existe una alta cohesión y un acoplamiento ligero entre subsistemas previniendo la afección a otros componentes.
- Cada módulo es mantenible por coherencia semántica y la abstracción de servicios comunes de dependencias a servicios externos con módulos separados definidos, generalizando aquellos aspectos que lo permitan de tal forma que, atributos específicos de cada subsistema no afecten los límites de otros componentes.
- Generar un sistema con componentes acoplados de forma ligera que se limita a comunicarse con las dependencias directas como intermediarios compatibles con los conectores de web services a otros eventos.
- Existen dos enlaces definidos uno es a los registros en ejecución y otro es el archivo de configuración, que permiten la consulta a componentes internos y externos de datos del sistema.

4.6 Disponibilidad

El sistema estará disponible las 24 horas, los 7 días de la semana, permitiendo al usuario un acceso en cualquier momento, con una excepción semanal, en base a la frecuencia de usuarios para realizar mantenimiento y actualización de la información interna del sistema; planteando 3 horas de mantenimiento y actualización del sistema, disponibilidad del 95% mensual.

Estrategia:

En caso de desconexión el sistema persiste mediante web sockets para reconectarse a internet en un tiempo de reconexión limitado.

Táctica:

- Manejo y registro de excepciones para conocer el estado y la disponibilidad del sistema conociendo exactamente el fallo.
- Detección: Redundancia activa, garantiza que todos los mensajes de entrada lleguen a los componentes de los subsistemas dentro de la aplicación.
- Recuperación y reintroducción: Algunos componentes se sincronizan con los estados, en caso de que la respuesta de los componentes externos falle o sea inválida se sincronice en un nuevo estado al componente.
- Prevención: Un monitoreo activo de procesos permite controlar la inconsistencia de los procesos en ejecución en caso de falla o comportamientos no definidos.

4.7 Portabilidad

El sistema es responsivo adaptable a dispositivos.

El sistema depende fuertemente de un dispositivo físico que lo soporte con función en sistemas operativos Windows, estos cuentan con tecnologías para una adaptabilidad nativa a servicios; garantizando el despliegue de la aplicación e instalación del sistema compatible con modelos recientes o dispositivos con versiones de actual generación vigentes en 2021.

El código será implementado con estándares y convenciones de desarrollo para la codificación y algoritmia del sistema, con la finalidad de obtener una adaptabilidad responsiva con relación al modelo del dispositivo, la disponibilidad de la información y el despliegue del mapa dentro de la partida.

El caso ideal será que el sistema tenga un soporte dentro de la república mexicana, quienes pueden acceder a la aplicación desde un dispositivo que requiere recursos de memoria para que el sistema funcione, este servicio consume a otros servicios y sistemas gestionados compatibles con sistemas operativos Windows.

La estrategia utilizada muestra la prioridad de la reacción a estímulos con tácticas como respuestas del sistema.

5. Funcionalidad general del sistema

El sistema propuesto contiene funciones solicitadas por el cliente, para mantener un diseño estructural que muestra el comportamiento de los componentes y su comunicación mediante conectores a aplicaciones externas (web services) orientado a eventos permitiendo reaccionar a estos mediante llamadas síncronas a objetos que evalúan las solicitudes del usuario. Esto permite al sistema tratar los eventos mediante un modelo cliente servidor en un middleware como el controlador de eventos principal, dando la prioridad a los datos, que son mandados como mensajes desde un componente que realiza peticiones RPC de eventos (interfaz) a otro componente que resuelve peticiones a los eventos del mapa, manteniendo una conexión y permitiendo comunicar a varios jugadores en una sesión multijugador.

A continuación, se sigue la estructura del modelo arquitectónico propuesto:

5.1 Adecuación

El sistema detecta el tipo de dispositivo para mostrar el despliegue de la interfaz de usuario dentro de la aplicación.

5.2 Exactitud

El sistema realiza cálculos polinomiales y operaciones sobre un vector en un sistema de coordenadas para desplazarse en un mapa y realizar acciones en tiempo real en una partida multijugador.

5.3 Interoperabilidad

Utiliza la funcionalidad del dispositivo para representar la localización de un vector y su desplazamiento en tiempo real. Así como consulta a servicios a un servidor y un modelo de arquitectura orientada a eventos.

5.4 Seguridad

Los datos son manejados de acuerdo con las sesiones proporcionadas por el usuario y delimitado por el servidor proporcionado para soportar la ejecución de la aplicación mediante una conexión segura.

El usuario sólo puede acceder a la vista proporcionada por la interfaz de usuario, de tal manera que el modelo sólo permite pasar al controlador, conectando al usuario al sistema en una sesión individual registrada con su perfil.

Para acceder al sistema el usuario utiliza un nombre de jugador para realizar el ingreso, la sesión se mantiene en un modelo cliente servidor distribuido.

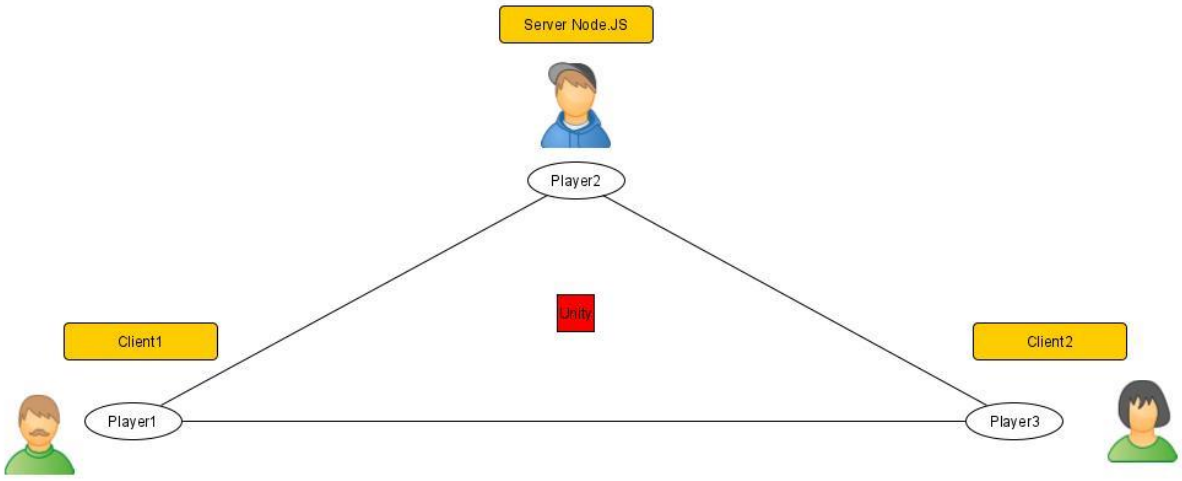
5.5 Compatibilidad

El sistema se mantiene operativo de manera nativa al sistema operativo del dispositivo, esto con la prioridad de interoperar con funcionalidades propias del sistema aprovechando los recursos que sirven de soporte para el trabajo con el consumo de datos a servicios para mantener la comunicación que permite una intercomunicación mediante web sockets con el modelo dentro de la aplicación.

Todas las características del sistema determinadas como objetivos serán puestas a prueba y validadas una vez implementadas para garantizar la calidad esperada de la aplicación, así mejorar y obtener nuevos requerimientos con la posibilidad de ser implementados en próximas versiones.

6. Arquitectura de software

El modelo arquitectónico describe a alto nivel el diseño propuesto vital para el funcionamiento del sistema.



6.1 Resumen

Para el desarrollo del sistema se utilizan herramientas de desarrollo de videojuegos a través de Node.js y Unity para implementar una arquitectura cliente servidor orientada a eventos que consume información del servidor por medio del sistema que conecta al cliente con el servidor para mantener la comunicación.

6.2 Arquitectura de software propuesta

El sistema propuesto hace uso de Node.js y Unity. Dado que el sistema es una aplicación, se utiliza la arquitectura distribuida del sistema cliente servidor para la gestión de transacciones de información en un sistema distribuido con la ventaja de que un componente No afecta a otros componentes que se utilizan internamente en el entorno; además del patrón orientado a eventos para la provisión de eventos realizados por el usuario, así mostrar el despliegue de información en la partida en tiempo real del lado del cliente.

El patrón de arquitectura fue decidido con criterio en base a los requerimientos que utilizan internamente un sistema distribuido con una estructura óptima, esto para mantener una comunicación eficiente entre componentes y servicios externos.

A continuación, se explica la lógica interna del sistema en respuesta al manejo de información con el uso de la arquitectura de software propuesta:

- La capa del modelo servidor es responsable del manejo de datos, esto permite al usuario realizar peticiones, así como actualizar información en la base de datos correspondiente a su nombre de usuario y partidas registradas.
- La capa de controlador middleware que actúa como intermediario entre el modelo que permite la gestión de eventos en el sistema como lo es el cálculo y la posición del vector sobre el mapa de la partida.
- La información es mostrada en la capa de vista por una interfaz que le presente la información al usuario, este proceso muestra los datos procesados en el modelo en base a las peticiones del usuario.

6.3 Diseño de juego

Juego hecho en Unity 3D

Multijugador: Con soporte para 3 jugadores.

Tipo de juego: Estrategia, Puzzle.

Vista: Isométrica.

Componentes de juego:

Objetivo de nivel

Moneda

Requiere: Que alguno de los jugadores obtenga la moneda.

Acción: pasa a jugadores al siguiente nivel

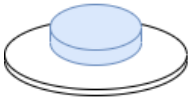
Jugador

Robotcito

Caracteristicas: 1 unidad de peso y 1 unidad de fuerza

Acción: Movimiento en 4 direcciones (input por el usuario WASD/flechas)

Objetos utilizados en el diseño de los puzzles



Botón

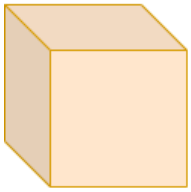
Requiere: 1 unidad de peso

Acción: Activarse a si mismo

Barrera

Requiere: Lista de botones necesarios en modo 'activado' (color coded)

Acción: activa animación que baja la barrera y permite el paso



Caja

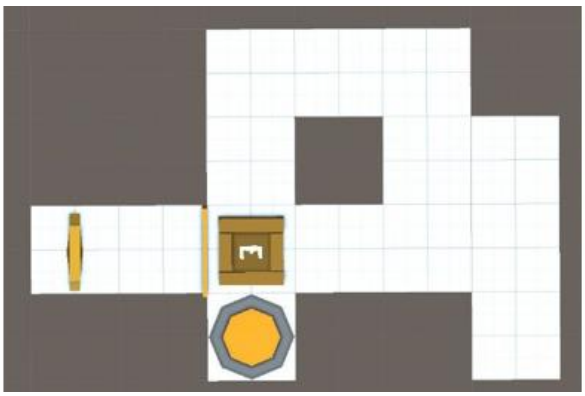
Caracteristicas: 1 unidad de peso

Requiere: x fuerza (variable individual a cada caja)

Acción: Activa rigidbody quer permite el movimiento de la caja

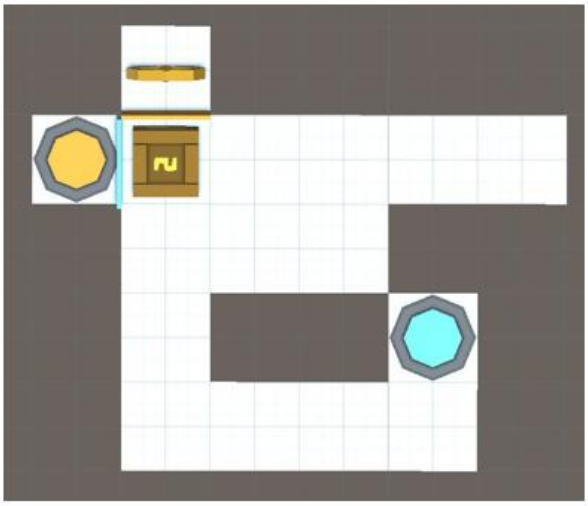
Diseño de niveles:

Los tres jugadores deben trabajar en equipo para completar cada uno de los tres niveles. Cada nivel se gana llegando a la moneda.



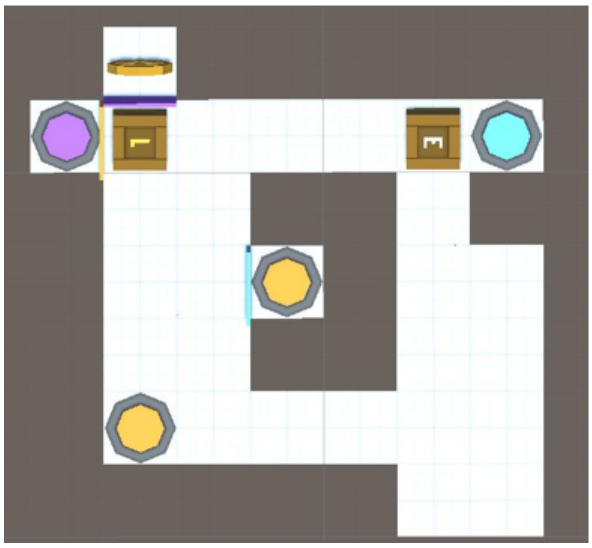
Nivel 1

1. Requiere que los 3 jugadores empujen la caja hacia el botón naranja
2. Se abre compuerta naranja
3. Tomar moneda



Nivel 2

1. Requiere que un jugador se ponga sobre el botón azul para abrir compuerta azul
2. Y que los otros 2 jugadores empujen la caja hacia el botón naranja
3. Se abre compuerta naranja
4. Tomar moneda



Nivel 3

1. Los 3 jugadores deben empujar una de las cajas hacía el botón azul para abrir compuerta azul
2. Requiere que dos jugadores se ubiquen en los botones naranjas para abrir compuerta naranja
3. Y que el jugador restante empuje la caja hacia el botón morado
4. Se abre compuerta morada
5. Tomar moneda

7. Mapeo de hardware/software

El dispositivo del usuario será la herramienta de acceso a la aplicación, soportado por sistemas operativos Windows, en sus versiones más recientes. El dispositivo del usuario tiene integrados componentes que serán utilizados por el software como es la localización y el acceso a internet mediante el uso de datos móviles para realizar la transferencia de información que consume la aplicación y los servicios relacionados a este.

Estos equipos, aparte de los diferentes equipos de interconexión entre los equipos de red, pueden crear diferentes clústeres de servidores para que el sistema mantenga una mayor redundancia y de este modo un mayor soporte a errores.

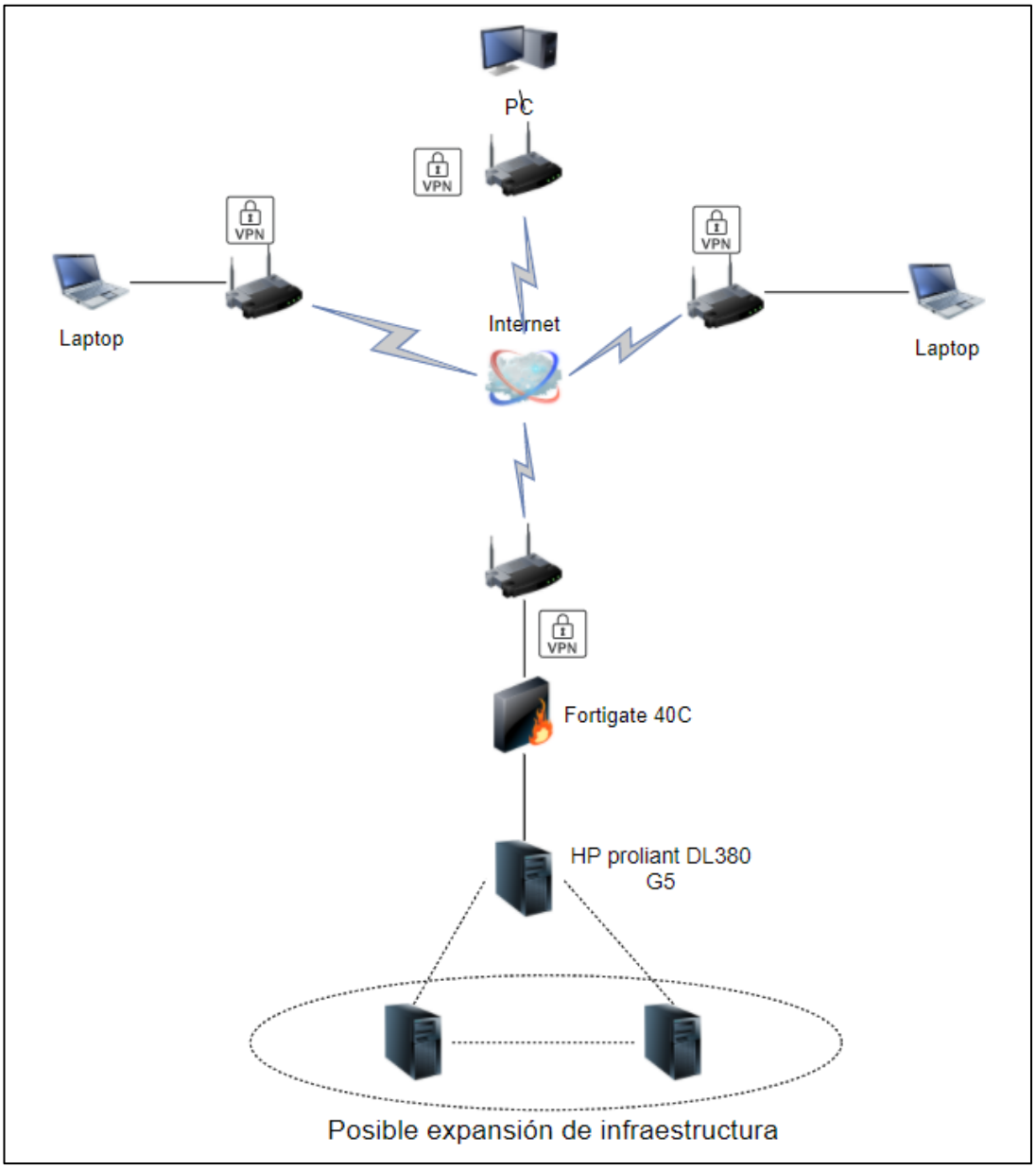


Diagrama de interconexión por medio de VPN.

7.1 Infraestructura

El software mantiene una conexión permanente al web services que es consumido por el usuario, esto aumenta la confiabilidad y exactitud en los cálculos mediante la comunicación en tiempo real.

La precisión de los eventos dentro del sistema guarda los eventos que afectan el modelo, estos datos contienen el estado del sistema leído de manera secuencial; los eventos permiten conocer el estado del sistema y a través del histórico son utilizados para comprender el estado en cualquier momento en su ciclo de vida y la presentación de los datos.

Para el desarrollo del sistema se utiliza la siguiente arquitectura física para la implementación del proyecto, la implementación del proyecto considera los siguientes dispositivos físicos, las especificaciones de estos se muestran a continuación:

- Servidor HP Proliant DL380 G5:

Procesadores	1 x Intel Xeon Quad Core E5420 4 cores x 2,5Ghz 12Mb L2 cache Virtualization Technology, 64-Bit. 1 x Intel Xeon Quad Core E5420 4 cores x 2,5Ghz 12Mb L2 cache Virtualization Technology, 64-Bit.
Memoria	8GB DDR2 a 667MHz PC5300 ampliable hasta 64GB.
Ranuras de expansión Libres	1 x PCI Express x8 perfil bajo. 1 x PCI Express x8 altura completa. 2 x PCI Express x16 altura completa.
Tarjeta Controladora RAID	HP Smart Array P400 con módulo de 512MB de Cache + batería BBWC Controller (RAID 0/1/1+0/5/6)
Unidades de disco duro	4 X HD 146GB 10K SAS
Unidades ópticas	DVD
Tarjeta Red integrada	2x NC373i Gigabit
Puertos de E/S	3 x RJ-45 (1 for iLO 2), Un puerto serie de 9 patillas, 5 X USB (1 interno), 2x vídeo, ratón PS/2, teclado PS/2.
Alimentación	2 X Fuentes de alimentación redundantes de 800W vatios conectables en marcha 110/220 voltios
Chasis	1 X Fuentes de alimentación redundantes de 800W vatios conectables en marcha 110/220 voltios
Tarjetas gráficas	ATI ES1000 32MB SDRAM

- Fortigate 40C:

Puertos LAN 10/100/1000	5
Puertos WAN 10/100/1000	2
USB (Client/Server)	1
RJ-45 Consola Serial	1
Almacenamiento local	4GB
Firewall Throughput	200 Mbps
IPSec VPN Throughpu	60 Mbps
Client-to-Gateway IPSec VPN Tunnels	20
Sesiones concurrentes (TCP)	40,000

8. Gestión de datos persistente

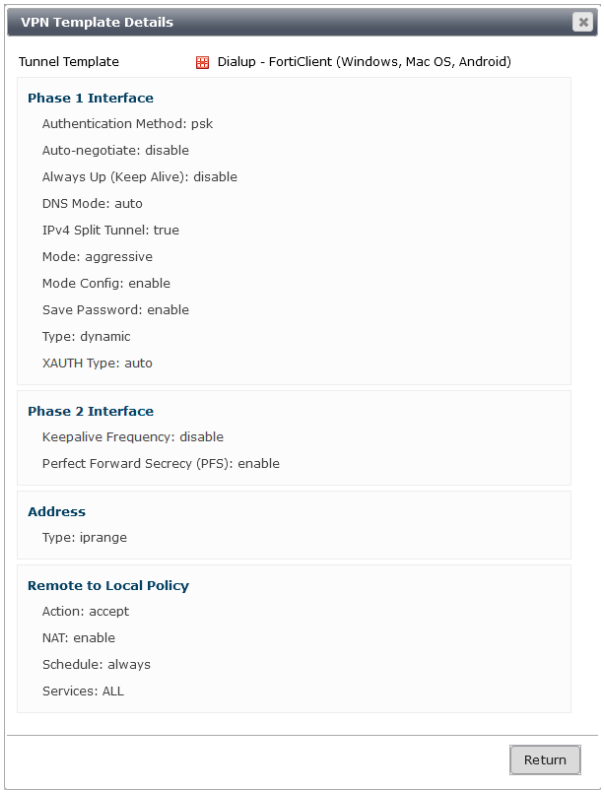
Para la gestión de datos persistente, será utilizado un servidor dedicado, este gestiona la consulta a los datos del usuario, así como la conexión. El servidor es el responsable de mantener y recuperar los datos de manera persistente y consumir información mediante servicios que requieren transmisión de datos generando tráfico en la red.

8.1 Control de acceso y manejo de información

El control de acceso se aplica al inicio de sesión requerido.

Mediante un entorno de conexión para hacer las pruebas al proyecto, se optó por la utilización de conexiones a través de conexión vía VPN proporcionadas por el hardware de Fortinet, en este caso un Fortigate 40C y el cliente Forticlient, de esta forma como se muestra en la gráfica se puede realizar la interconexión de los equipos clientes externos los cuales por ahora son sistemas Windows para conectarse al ambiente de pruebas.

Las configuraciones tanto del cliente como del Fortigate se presentan a continuación en las siguientes imágenes.



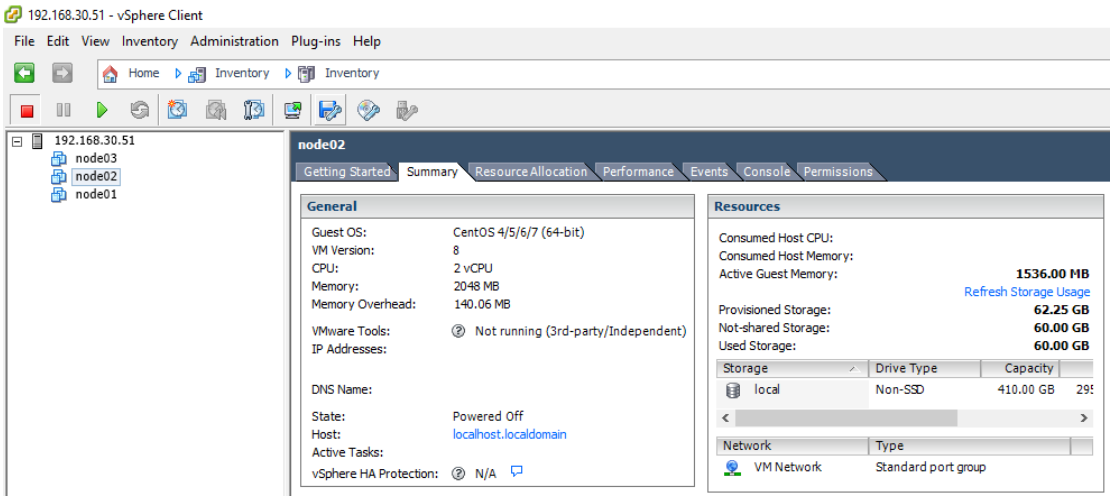
Configuración de VPN Fortigate 40C.

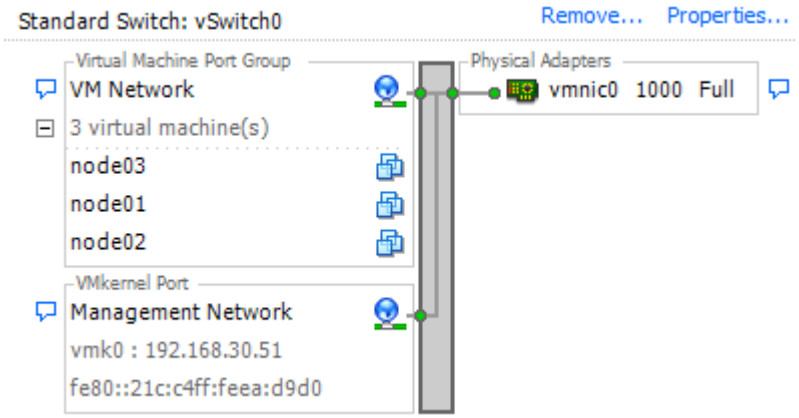
Configuración Forticlient.

Los servidores utilizados en el sistema están virtualizados utilizando VMware específicamente la versión 5.5 monta en el servidor, con este ambiente se levantaron 3 servidores Linux con las siguientes características:

Sistema operativo:	Linux Centos 7
VCPUs	2
Memoria	2048 MB
Versión Nodejs	10.24.1

Los servidores están conectados vía una infraestructura de red virtual la cual trabaja en el rango de red 192.168.30.0/24´, de esta forma la VPN tiene acceso a este segmento de red:





8.2 Control de colaboración de objetos

El flujo de control del sistema está controlado mediante eventos, el sistema debe asegurarse de soportar tres usuarios que puedan mandar peticiones e interactuar con el sistema a la vez, además que esto NO afecte la coherencia de los datos. Para la gestión de peticiones se almacena un histórico de eventos que se encargan de controlar los eventos implementados en el servidor de la arquitectura del sistema.

El servidor monitorea y registra los eventos *on-click()*, que interactúa en respuesta al usuario. Los usuarios pueden desencadenar diferentes eventos al mismo tiempo; la aplicación debe responder a los eventos en el orden en que llegan.

- Cada que el usuario interactúa con un componente, se envía un evento al subsistema específico.
- Se envían diferentes eventos a diferentes subsistemas.
- Un subsistema funciona según el orden de ocurrencia de los eventos.
- La lógica de programación de eventos se describe en las prioridades o relevancia de un subsistema para que este ignore o desencadene una acción en respuesta a la petición del usuario.

Websockets:

Se utiliza Node.Js para establecer la comunicación utilizando web sockets, la comunicación simula la conexión mostrada en la figura anterior; el código necesario para habilitar estos servicios varía solo en la sección necesaria para interconexión entre servidores ya que cada servidor se conecta únicamente a los otros servidores diferentes a este, esto con el fin de evitar el ciclo de información.

```
const WebSocket = require('ws')

//---------------------------------------- Servers ------------------------------------
const wss1 = new WebSocket.Server({port: 8082},()=>{
    console.log('server started')
})

wss1.on('connection', function connection(ws) {
   ws.on('message', (data) => {
      console.log('data received \n %o',data)
      wss1.broadcast(data);
   })
})


wss1.on('listening',()=>{
   console.log('listening on 8082')
})

wss1.broadcast = function broadcast(msg) {
   console.log(msg);
   wss.clients.forEach(function each(client) {
       client.send(msg);
    });
};
```

```
//--------------------------------------- Clientes ----------------------------------
const wss = new WebSocket.Server({port: 8081},()=>{
    console.log('server started')
})
wss.on('connection', function connection(ws) {
   ws.on('message', (data) => {
      console.log('data received \n %o',data)
      wss.broadcast(data);
   })
})
wss.on('listening',()=>{
   console.log('listening on 8081')
})
wss.broadcast = function broadcast(msg) {
   console.log(msg);
   wss.clients.forEach(function each(client) {
       client.send(msg);
    });
   wss1.clients.forEach(function each(client) {
       client.send(msg);
    });

};
//--------------------conectar aservidor---------------
async function init() {
  console.log(1);
  await sleep(20000);
  //-----------------------------------------------------
const url = 'ws://192.168.30.52:8082'
  const connection = new WebSocket(url)

  connection.onopen = () => {
    console.log(`me conecte`)
  }
  connection.onerror = (error) => {
    console.log(`WebSocket error: ${error}`)
  }

  connection.onmessage = (e) => {
    console.log(e.data)
    wss.clients.forEach(function each(client) {
       client.send(e.data);
       console.log(`mande a mis clientes`)
    });
  }
//-------------------------------------------------------------
const url2 = 'ws://192.168.30.54:8082'
  const connection2 = new WebSocket(url2)

  connection2.onopen = () => {
    console.log(`me conecte`)
  }
  connection2.onerror = (error) => {
    console.log(`WebSocket error: ${error}`)
  }
  connection2.onmessage = (e) => {
    console.log(e.data)
    wss.clients.forEach(function each(client) {
       client.send(e.data);
       console.log(`mande a mis clientes`)
    });

  }
```

```
//-------------------------------------------------------
  console.log(2);
}

function sleep(ms) {
  return new Promise((resolve) => {
    setTimeout(resolve, ms);
  });
}

init()
```

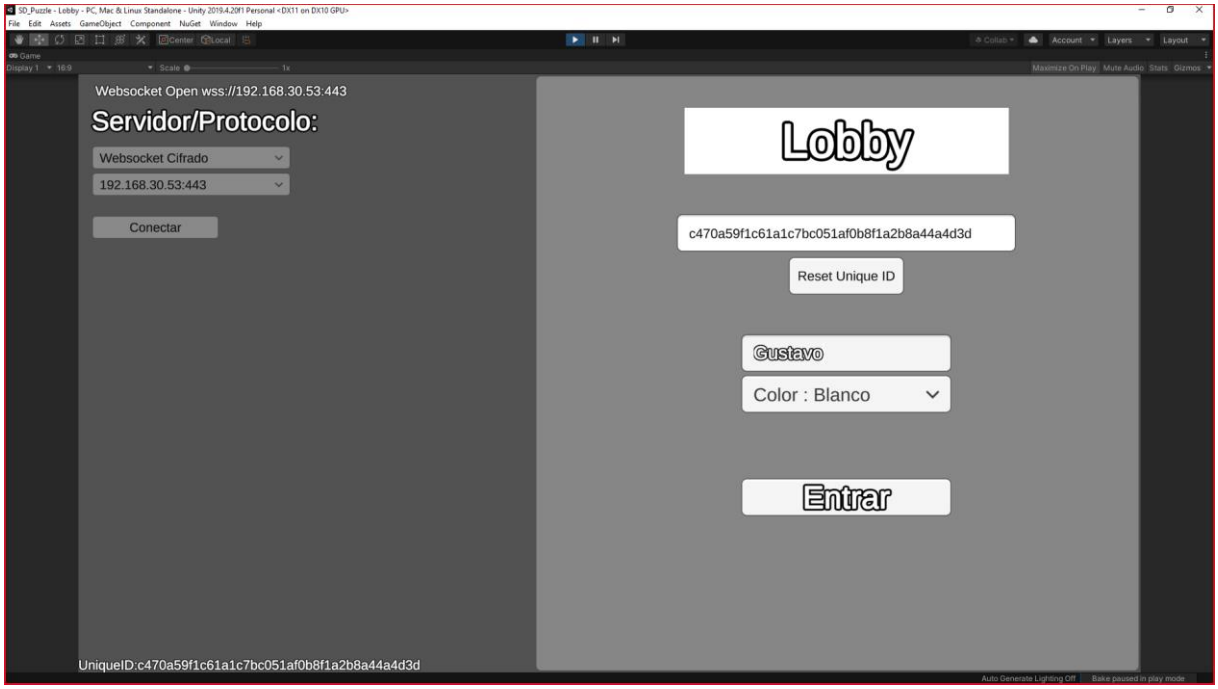
El funcionamiento del código anterior genera que cualquier usuario conectado a uno de los servidores retransmitirá la información a los otros servidores para de esta forma poder reenviar la información a los clientes conectados a cada servidor, la información es compartida por todos.

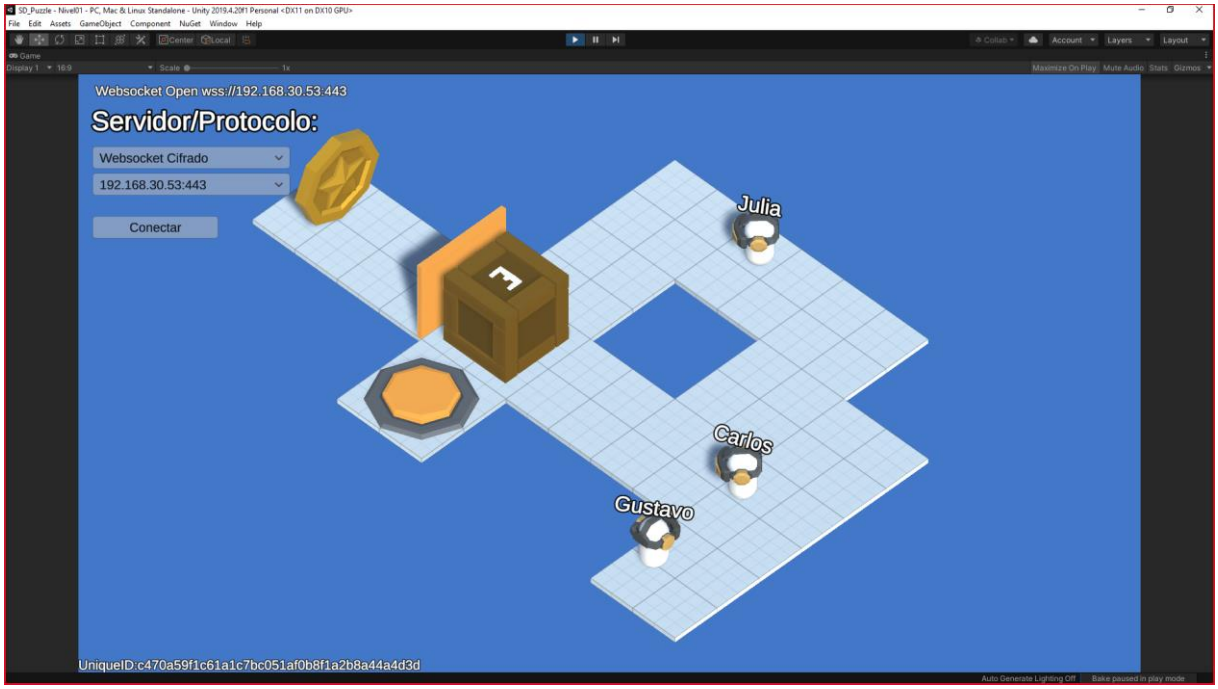
9. Evolución del sistema

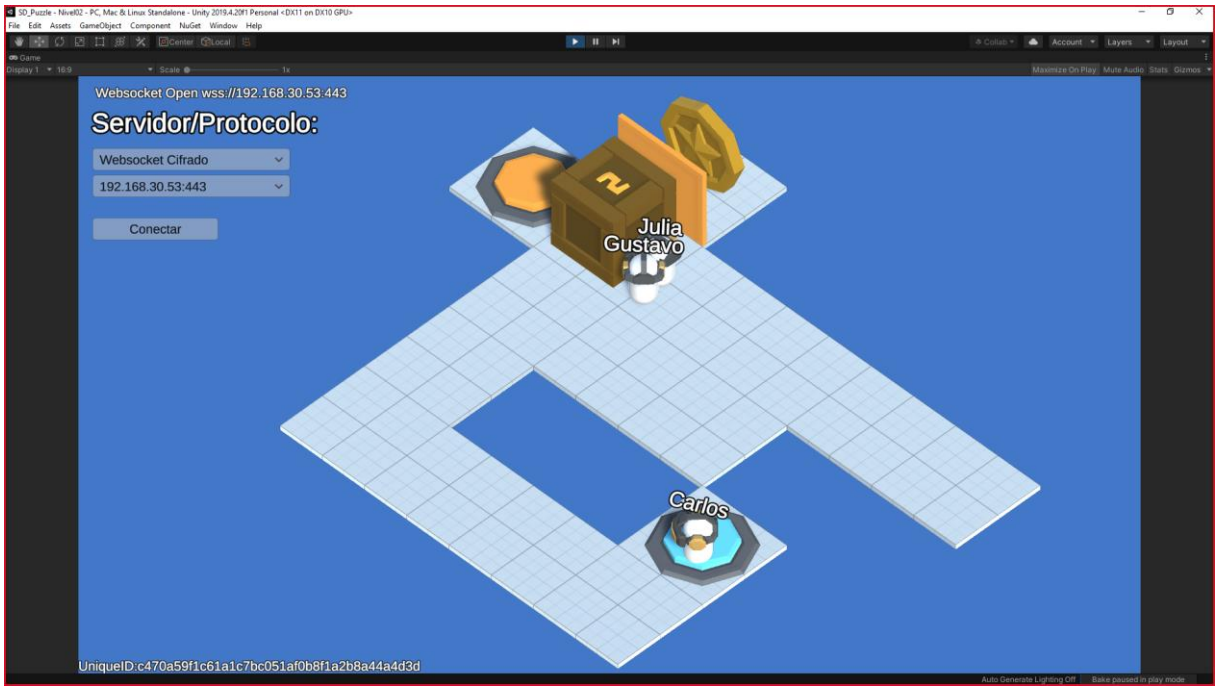
El sistema se ha implementado de acuerdo con el cronograma de actividades especificado en el plan maestro de gestión de proyecto, y en relación directa con el modelo general del proceso de diseño del proyecto de forma que se han realizado las actividades acordadas con los stakeholder implementando una aplicación con base en el análisis y la viabilidad de los requerimientos evaluados en este documento.

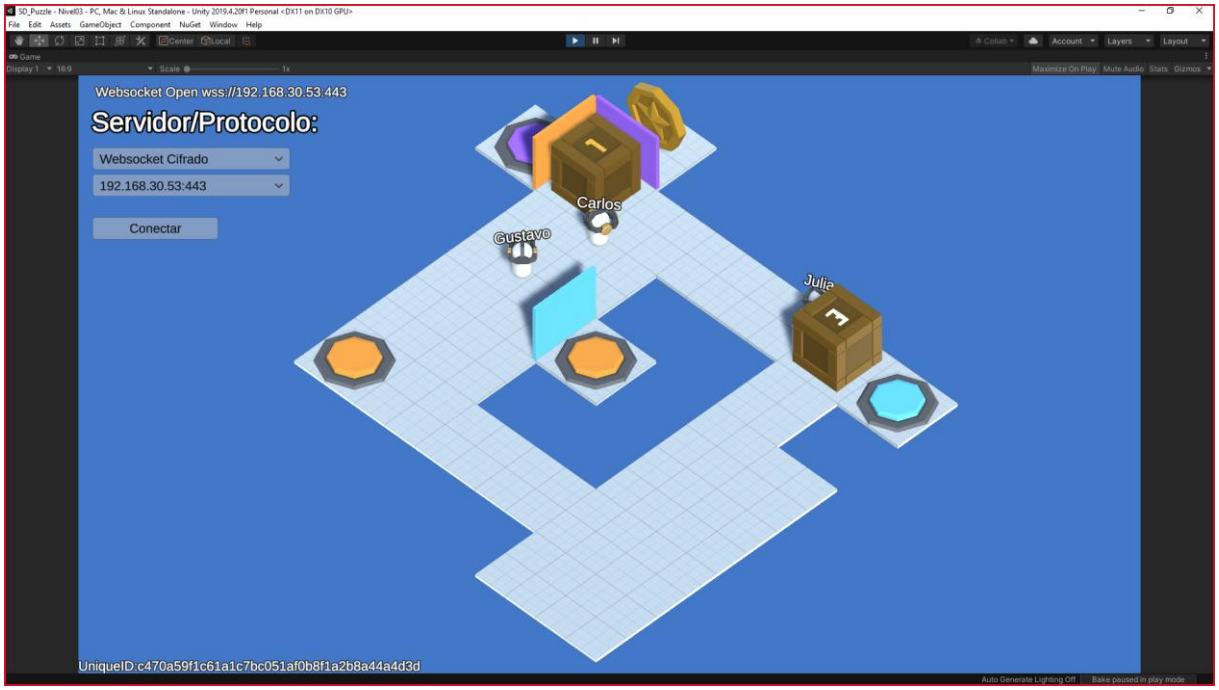
10. Apéndices

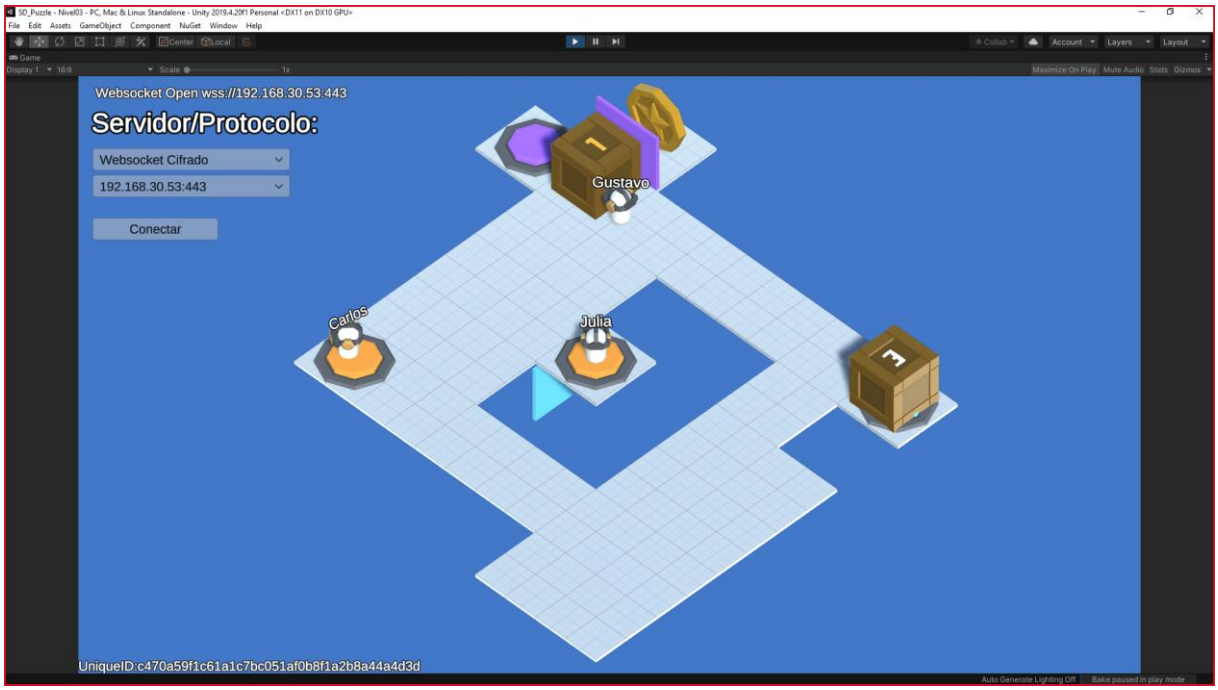
Capturas de Juego:






Websocket Open wss://192.168.30.53:443
Servidor/Protocolo:
Websocket Cifrado
192.168.30.53:443
Conectar
Julia
Gustavo
Carlos
UniqueID:c470a59f1c61a1c7bc051af0b8f1a2b8a44a4d3d


Websocket Open wss://192.168.30.53:443
Servidor/Protocolo:
Websocket Cifrado
192.168.30.53:443
Conectar
Carlos
Gustavo
Julia
UniqueID:c470a59f1c61a1c7bc051af0b8f1a2b8a44a4d3d


Websocket Open wss://192.168.30.53:443
Servidor/Protocolo:
Websocket Cifrado
192.168.30.53:443
Conectar
Gustavo
Carlos
Julia
UniqueID:c470a59f1c61a1c7bc051af0b8f1a2b8a44a4d3d


Websocket Open wss://192.168.30.53:443
Servidor/Protocolo:
Websocket Cifrado
192.168.30.53:443
Conectar
Gustavo
GANARON
Julia
UniqueID:c470a59f1c61a1c7bc051af0b8f1a2b8a44a4d3d