Communication and Web Interaction Aspects for the Construction of Simulation Scenarios: Applied on Natural Disaster Response

Gino Moena
Departamento de Ingeniería Informática
Unversidad Católica deTemuco
Temuco, Chile
gmoena2013@alu.uct.cl

César Navarro
Centro de Innovación e Investigación Aplicada
Osorno, Chile
cnavarro@ceinina.cl

Roberto Aldunate
College of Applied Health Science
University of Illinois
Urbana-Champaign, USA
aldunate@illinois.edu

Oriel Herrera
Departamento de *Ingeniería Informática*
Universidad Católica de Temuco
Temuco, Chile
oherrera@uct.cl

***Abstract*—Interaction and communication in simulation scenarios on a Web platform presents several problems and limitations. One particular problem, when using video-games, is the optimal visualization of 3D terrain in the scenarios. Another problem is the interplay between the video-game scenarios and the Web platform. Given the scenario is not a static and isolated module from the Web platform, it must interact with it, becoming a complimentary tool that directly support the content that the platform brings to the end user. In particular, using the Unity3D plug-in, the key issue is to set a bidirectional communication channel between Unity WebGL and HTML5, using JavaScript. This development was used in a dynamic Web environment (simulator) aimed at developing decision making skills on people fighting wildfires. This article describes functionality that simplify and improves the real-time communication between the different modules of the simulator, enhancing also integration flexibility. Specifically, the work focused on bidirectional communication channel that enables real-time updating of the WebGL based on the status of the Web platform. Results are very encouraging, given the performance and consistency contained between the Web platform and the WebGL container, in particular in maintaining synchronous information between elements of the simulator such as between Google Maps plug-in, Unity3D plug-in, and the rest of the Web platform. This functionality proves to be very successful for presenting highly dynamical phenomena, such as wildfires, on different containers/viewers of the Web platform.**

Keywords—Serious Videogames, disaster response, Unity, WebGL

I. INTRODUCTION

Chile ha vivido diversas catástrofes naturales durante estos últimos años en las cuales la población y autoridades se han visto sobrepasadas respecto a la toma de decisiones apropiadas para enfrentar la catástrofe. Capacitar a las personas que participan de la toma dedisiones frente a desastres naturales no es una tarea fácil, ya que el momento en que ocurre el desastre es cuándo se puede evaluar el comportamiento de estas personas frente a las situaciones diversas que se presenten. Replicar un escenario de desastre es muy costoso o bien impracticable; por ejemplo, quemar un bosque, generar un aluvión, fracturar o derribar edificios, etc. Frente a esta problemática se ha estado trabajando en un proyecto que implementa un simulador donde el usuario/jugador se enfrenta a situaciones similares a las que ocurren durante el desastre natural, teniendo que tomar las deciones apropiadas o no, las que posteriormente pueden ser evaluadas y retroalimentadas. La tecnología que hay detrás de este simulador conlleva una serie de requerimientos que presentan desafíos de implementación. Algunos de ellos se centran en aspectos de comunicación e interacción para un simulador embedded en un ambiente Web.

Dado la exponencial evolución de la Web se ha podido dar soporte a cada vez más tipos de sistemas dentro del formato HTML. Desde el año 2011, dado al desarrollo de la librería WebGL y declarado como estándar, el desarrollo 3D ha podido incursionar en el mundo Web siendo unas de las tecnologías que más tarde ha entrado con respecto a otros medio digitales [1]. Los principales beneficios de esto es la visualización de objetos/escenarios 3D con los fines que el usuario estime convenientes sin la necesidad de instalar software adicional, sino que directamente desde la Web utilizando JavaScript. Esto hace que el soporte del escenario sea multiplataforma, por lo que permite que un usuario pueda compartir sus archivos Web3D a través de Internet con usuarios de todo el mundo con visualización instantánea sin necesidad de tener complementos especiales. Sin embargo, se debe tener en consideración que dependiendo de la máquina en la que se visualiza la información podría existir diferencias en el comportamiento. Así, máquinas más antiguas con un hardware no tan actualizado podrían verse afectadas en la visualización por falta de memoria o poder gráfico para la correcta visualización de dichos complementos.

La tecnología WebGL ha sido aprovechada por distintos usuarios alrededor del mundo. Por ejemplo, en el área de la biología se ha utilizado para proyectos como Pycortex en el que desarrollan un software Web para la visualización de imagen por resonancia magnética funcional (fMRI) siendo el elemento visual 3D primordial para mostrar al usuario una vista más comprensible de la actividad cortical. El uso de WebGL en este caso viene a reemplazar la imagen estática que se usaría para mostrar al usuario la actividad cortical, usando un método más interactivo que permite al usuario tener una mejor representación del proceso que se desarrolla. Esto permite que mapas complejos de superficie de fMRI sean distribuidos de forma abierta en línea sin la necesidad de la instalación de software complejos [2].

Es preciso destacar también el uso en videojuegos, esto dado que muchos juegos hoy en día hacen uso de librerías de desarrollo en 3D como es OpenGL. WebGL en este caso, es una librería Web basada en la versión OpenGL ES 2.0, por lo que su funcionalidad desde la base es muy parecida. Esto permitió que motores de videojuegos en 3D como Unity tuvieran la chance de realizar un port de su sistema a WebGL y hasta la actualidad Unity se ha encargado de mejorar su plataforma en el área de desarrollo Web. De este modo, hoy en día es posible desarrollar aplicaciones Unity que pueden ser incrustadas directamente en páginas Web. Hay situaciones en que la API por sí sola no es tan útil para mostrar contenidos; dado que al estar en una página Web eventualmente es necesario que interactúe con otros elementos de la página para que el sitio sea más interactivo. Así es como otros desarrolladores han ido con el paso del tiempo probando nuevas formas de interacción con el sistema, como por ejemplo aprovechar el uso de otras tecnologías de HTML5 para complementar lo que sería la API WebGL. Un ejemplo de esto es el uso de la herramienta WebSocket para una conexión persistente en el sitio Web para actualizar datos dentro de la API siendo recibidos directamente del servidor; esto es lo que postula Andrioti [3], indicando que es posible aprovechar dichas características para potenciar el uso de WebGL dirigido a los videojuego en línea.

Así es como ahora en el marco del Proyecto FONDEF IT16i10096 "Simulador basado en video juegos para respuesta a desastres", financiado por el gobierno de Chile, se requiere la implementación de comunicación WebGL para la interacción y comunicación entre un sitio Web y la API WebGL, para refrescar dentro del escenario 3D de Unity situaciones que están ocurriendo fuera del simulador a través de otras APIs como Google Maps por ejemplo. Por lo tanto, la API WebGL deja de ser un módulo aparte de la página Web y se integra a través de la intercomunicación al sistema interconectado del sitio Web.

La necesidad de comunicación en este caso es el problema principal, debido a que es necesario mostrar por ejemplo, focos de incendio con distintos tamaños de matrices, representación de un aluvión, terremotos, etc. El simulador, desde el lado de Unity, utiliza escenarios 3D con relieves extraídos satelitalmente utilizando componentes en Unity que permiten esto.

Por lo tanto, uno de los objetivos es simular que el usuario está experimentando el desastre, esto para ayudar en la toma de decisiones a nivel administrativo del qué hacer ante dicha situación. Es por esto que el software va dirigido a gente que tiene manejo del tema (usualmente entes del gobierno) con el fin de mejorar la toma de decisiones a través de la simulación. Más detalle del proyecto en sí encontramos en [6][7].

La solución al tema en cuestión es abordada estableciendo normas de comunicación que deberían satisfacer al menos la mayoría de las necesidades del proyecto. Además, se exploran posible soluciones a otras problemáticas, como el uso de variables especiales en Unity, como es el tipo WWW que permite descargar información contenida en una página Web externa mientras no sea una carga de datos muy grande (mayor a 1 MB). En complemento al tema de la comunicación también se trabaja la recepción de datos para mostrar en la API WebGL de forma correcta los cambios establecidos en la parte Web. De esta manera se valida el uso de la comunicación entre la página Web con la API WebGL generada por Unity a través de Javascript.

II. TRABAJOS RELACIONADOS

A. *Funcionalidades actuales de comunicación*

Como fue mencionado con anterioridad, el uso de WebGL se ha extendido en otros trabajos relacionados con la comunicación entre API's en páginas Web. Este es el caso de la tesis postulada por Andrioti [3] en la cual implementa el uso de WebGL con herramientas como WebSocket, conexiones peer-to-peer, para crear un videojuego multiplayer online que se juega desde el navegador abriendo las puertas a un videojuego multiplataforma sin necesidad de instalar software extra, para ser utilizado en distintas plataformas. Esto es una de las posibilidades en el uso de WebGL con comunicación entre sistemas.

También se ha experimentado en el ámbito educativo con fines médicos en lo que es comunicación Unity. Este es el caso de un estudio denominado "Development of Smart Multiplatform Game App using UNITY3D Engine for CPR Education" [5], a lo cual refiere al desarrollo de juego multiplataforma orientado a educar a las personas en la resucitación cardiopulmonar, lo cual puede ayudar a salvar vidas. En dicho artículo se describe el uso de las funciones de Comunicación de Unity *GetUnity().SendMessage* y *Application.ExternalCall* que permiten el envío y recepción de datos entre Unity y HTML5-JS. Esto con el fin de conectar el Web player a una API denominada OpenSocial de Google para obtener o enviar datos de perfil de usuario entre el Web player y el sitio Web.

B. *Uso de GMaps+Unity en otros contextos*

Existen otros usos de Google Maps en conjunto con Unity. En un momento existió un complemento de Unity que permitía usar Google Maps dentro de los videojuegos; éste se llamaba *Google Maps for Unity*. Sin embargo, ha sido discontinuado. También se han desarrollado proyectos en donde Google Maps ha contribuido en el desarrollo de la imitación de un terreno del mundo real dentro de un escenario de videojuego, similar a como en el proyecto actual se utiliza con el componente Terrain Composer. Así es como lo explica Vincent Ortells Rectalà [4] quien hace uso de un terreno real de Google Maps para la recreación dentro de Unity. Además, en marzo 2018 ha sido anunciado una nueva API de Google Maps que permite la creación de terreno en base a mapas tridimensionales de Google incluyendo edificaciones por lo que es muy probable que eventualmente existan más investigaciones con el uso de esta nueva herramienta.

III. ASPECTOS DE COMUNICACIÓN E INTERACCIÓN

Para la implementación de los aspectos de comunicación e interacción se decidió trabajar con el escenario de incendios forestales, ya que cubre la mayoría de los aspectos de comunicación que se necesitan, pudiendo reutilizar ciertos aspectos de ese problema para abordar los otros escenarios de desastres.

Además del problema de comunicación en el envío de datos, también se presenta el problema en el reflejo de acciones

requeridas por el navegador en el mismo elemento WebGL. Idealmente se busca representar una matriz con distintos estados en Unity la cual es conocida en el lado Web del simulador, por lo tanto el problema y objetivo principal es no sólo conectar ambos sistemas, sino también trabajar con Unity en la recepción de datos, esto para visualizar en pantalla la propagación de un incendio que está ocurriendo en una simulación que utiliza herramientas Web, como Google Maps, para mostrar la propagación del incendio.

IV. Prototipo de simulador para entrenamiento en Incendios Forestales

El simulador busca como objetivo entrenar al usuario en la toma de decisiones frente a desastres naturales. Esto, desde un punto de vista administrativo, significa que el simulador no sumerge al usuario dentro de un bosque en donde ocurre un incendio, sino que más bien el usuario observa la situación y propagación del incendio, encargándose principalmente de la gestión de recursos necesarios para lidiar con el incendio. De esta forma, debe preocuparse del despliegue de bomberos a zonas específicas para el control de los focos de incendios, la distribución de recursos, etc. Existen distintos factores que pueden afectar la propagación de un incendio, los cuales son controlados por el lado del navegador Web, por lo que el lado de WebGL es más estético que algorítmico, es decir, representa la salida visual de lo que el simulador quiere mostrar durante la propagación.

Como se explicó anteriormente, existen ciertas alternativas para la comunicación, como es el uso de otro tipo de variables. Por tanto la comunicación puede efectuarse de distintas maneras. En temas de comunicación se preparó un esquema que resume cómo se genera la comunicación entre Unity y la página Web.

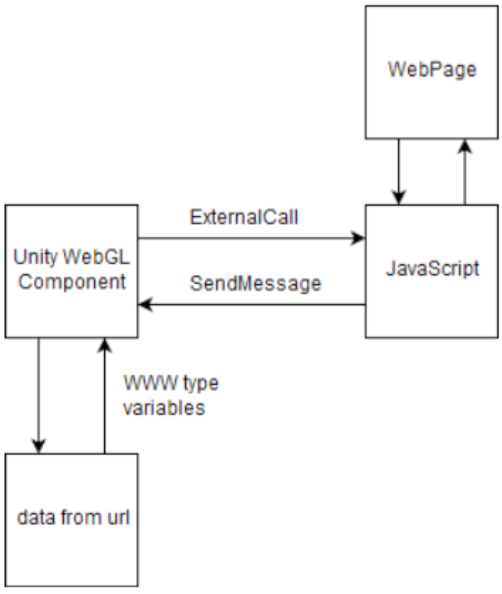

Fig. 1. Esquema Comunicación Unity-HTML.

Como se puede ver en la figura 1, la página Web que opera con Javascript se comunica con el componente WebGL a través de la función *SendMessage* y Unity se comunica con Javascript utilizando la función *ExternalCall*. También se muestra el uso de las variables del tipo WWW; un caso de uso es cuando Javascript en lugar de enviar a través de *SendMessage* una matriz de gran tamaño, envía la *url* del sitio Web, accediendo a éste a través de la variable de tipo WWW, extrayendo la información de la matriz directamente desde ese sitio Web. Este sitio Webl no necesariamente debe ser del tipo *.html*, sino que puede ser *.dat*, o cualquier formato posible.

Para comenzar con el uso de Google Maps en conjunto al plugin de Unity se definió y desarrolló una estandarización del manejo de la comunicación para el desarrollo del simulador. Idealmente se buscó que sea simple para el usuario que necesite usarla y para los métodos que la llamarán durante su ejecución. De esta manera, toda comunicación que quiera realizarse entre sistemas lo hará a través de esta metodología. Para esto se establecen ciertas reglas generales, pero primero hay que tomar en cuenta cómo funcionan las vías de comunicación de ambos sistemas para encontrar puntos en común en donde aplicar reglas similares. Por ejemplo, la comunicación de HTML a Unity al enviar un mensaje con la función SendMessage, ésta tiene como requisito apuntar hacia un objeto en el videojuego y este objeto debe tener un script con la función a la que se llama. En este caso, tanto en Unity como en HTML al hacer un llamado de un sistema al otro se accede a una función dentro de otro sistema enviando un string, por lo que la función de recepción en el otro sistema debe aceptar un tipo de datos string dentro de sus argumentos. Conociendo ambos puntos se concluye que primeramente debe existir un objeto en Unity que sea estático y que cada vez que se quiera utilizar el Script para la comunicación (el cual llamaremos *UnityAPI*) se debe hacer invocando al *GameObject* que contiene al script. En base a esto se establece como regla del proyecto que al momento de querer trabajar con comunicación bidireccional entre HTML y Unity se debe crear un *GameObject* vacío, agregarle el script *UnityApi* y cambiar el nombre del Objeto a *UnityCOM*. Como resultado, cada vez que se quiera utilizar la comunicación de HTML a Unity se invocará una función en HTML la cual realizará un envío de un mensaje al script *UnityAPI* contenido en el objeto *UnityCOM*. La sintaxis sería de la forma

SendMessage('UnityCOM','UnityReceiver', mensaje)

Donde *UnityReceiver* es la función contenida en el script *UnityAPI* que recepciona la información). Dicha línea de código será entonces la que enviará la información finalmente a Unity. A continuación se explicará cómo este SendMessage se implementa en un nuevo método que se creará para el envío de información a Unity.

Entonces, ya que se pueden enviar mensajes entre sistemas, se puede aprovechar esto en conjunto con la función *Split* la cual permite separar un texto mediante un delimitador. De esta forma, se propone una sintaxis general para el mensaje que se envía durante la comunicación. Como se busca realizar la mejor estructura posible del sistema, se propone que al enviar un mensaje, éste debe contener información para llamar a una función dentro de Unity, y además de esto, pasar parámetros para trabajar en esa función. En base a esto se decide crear un formato estándar de mensaje el cual tiene la forma

miFuncion: parámetro 1, parámetro 2,..., parámetro n.

Por lo tanto se define éste como el formato final para el envío de mensajes. Así, podemos ahora desarrollar los métodos finales para la comunicación de HTML a Unity y de Unity a HTML.

A. ComunicacióndeHTMLaUnity

En HTML, se crea la función *WebSender*, la que recibe como primer parámetro el nombre de la función que se quiere invocar en Unity, y los siguientes parámetros son los valores que se quieren agregar como parámetros adicionales a esa función. Debido a que pueden existir distintas funcionalidades se utiliza la palabra reservada *arguments* en la función *WebSender*, de este modo la función puede recibir tan solo un argumento (que sería la función a llamar) o múltiples argumentos (que sería la función a llamar más los parámetros adicionales necesarios para llamar dicha función en Unity). Una vez recibidos estos argumentos, *WebSender* siempre concatenará un ':' seguido del primer argumento, y luego concatenará el resto de argumentos mediante ','. Una vez terminada la concatenación, ésta se guarda en una variable mensaje, ejecuta la función *SendMessage* que se armó anteriormente y se envía el mensaje en Unity.

Ahora en Unity esta información debe ser recibida, por lo que debe existir una función para recibir los datos y trabajarlos. Esta función, llamada UnityReciever, recepciona los datos en forma de string y utiliza la función Split para separarlos mediante el delimitador ':'. De este modo, lo que está a la izquierda de ':' es el nombre de la función y lo que está a la derecha son los parámetros. Entonces, *UnityReceiver* utiliza la primera información que contiene el nombre de la función y la introduce en una cláusula SWITCH-CASE donde se busca el caso en que la función se aplica. Una vez encontrado, se ejecuta la función pasándole como parámetros lo que está al lado derecho del delimitador. Así concluye la comunicación o envío de datos desde HTML a Unity.

B. ComunicacióndeUnityaHTML

Para establecer la bidireccionalidad en la comunicación de ambos sistemas, ahora Unity en lugar de actuar como un receptor esta vez será el emisor de los datos y HTML el receptor.

El proceso inverso es similar, con algunas variantes en el código. La función que envia información a HTML desde Unity es UnitySender, la cual recibe como parámetro una variable de tipo string y luego una cantidad variable de argumentos del tipo object. Se definen los argumentos de tipo object para dar mayor dinamismo a los datos. De este modo, un arreglo del tipo object puede tener datos en binario, texto, o float dando mucha versatilidad al envío desde Unity a HTML. Similar a HTML, en Unity el primer parámetro es concatenado a un delimitador ':' y luego siguen los parámetros separados por ','. Estos parámetros son convertidos a string para no tener inconvenientes y finalmente se genera una cadena que sería el mensaje a enviar a HTML. El envío se realiza a través de *Application.ExternalCall('WebReceiver', data),* siendo *data* el mensaje a enviar.

En HTML la cadena es recibida a través de la función WebReceiver y los datos son separados con un *Split* utilizando el delimitador ':'. Al igual que en Unity, HTML utiliza el primer argumento del *split* para buscar la función en el SWITCH-CASE y luego ejecuta la función indicada por SWITCH-CASE pasando como argumento los datos obtenidos por el segundo argumento del *split*.

Así concluye la comunicación bidireccional entre Unity y HTML. Adicional a esto, se establece un orden de la estructura de la información, creando un nuevo script en Unity. Éste tiene por objetivo contener todos los métodos, en su mayoría métodos de uso general o de gran importancia para el proyecto. El script lleva por nombre *UnityGlobals* y cada función llamada en el SWITCH-CASE de Unity redireccionará a una función contenida en *UnityGlobals*. Además de esto, *UnityGlobals* contendrá información de funciones para realizar la normalización tal como se hizo anteriormente y también variables estáticas como el tamaño del mapa entre otras cosas.

El uso de la comunicación bidireccional y *UnityGlobals* será evidenciado en la siguiente sección, aplicado al manejo de cámaras y objetos que se desplazan por un mapa que representa una situación de incendio forestal ubicado en Santa Olga, región del Maule, Chile.

C. Aplicación de la comunicación bidireccional en el simulador para un caso de incendio forestal

En esta sección se presenta un experimento que consiste en reflejar la ubicación del usuario de Unity en el mapa de Google Maps dentro del navegador, de modo que cada vez que el usuario se se desplace, se actualice su posición en GoogleMaps. Para realizar esto se necesita obtener la posición del usuario en el mapa de Unity y luego enviarla de manera normalizada hasta Unity (esto se explicó en secciones anteriores). Bajo esta planificación se determina como objetivo principal validar la comunicación entre Unity y HTML mostrando la actualización de posición del usuario en el mapa de GoogleMaps simulando como si este fuera un GPS. Existen diferentes vistas para la navegación: la vista en primera persona, satélite y un modo de exploración a través de un helicóptero. Para este ejemplo se utiliza tanto la vista desde el helicóptero como la vista satelital para tener una mejor visión de resultados. El mapa utilizado es un mapa reconstruido de la zona de Santa Olga en la región del Maule. Este terreno ha sido construido utilizando los complementos de Unity *Terrain Composer* y *World Composer* que permite reconstruir imágenes satelitales. El mapa tiene una dimensión de 7970x7970 pixeles.

Para enviar la información a HTML se necesita extraer la posición actual del jugador y transformarla a valores entre 0 y 1 a través de la normalización. Primero que nada se crea un nuevo script el cual se agrega al objeto que se mueva en el mapa. Este script ejecuta una función de *UnityGlobals* la cual realiza una llamada a HTML a través del Api de Unity para la comunicación. Esta llamada debe realizarse en el método *update* del objeto. Sin embargo, si se realiza la llamada en cada frame de vida del juego la comunicación se vería sobrecargada, dado que se requiere de 60 frames por cada segundo de juego; esto implicaría 60 llamadas a HTML por segundo. Por tanto, considerando la dimensión del mapa, se establece un umbral *s* de movimiento, de modo que si el jugador se desplaza una distancia inferior a s, no realizará ninguna llamada. De este modo, la llamada a HTML se realizará solo si el desplazamiento es mayor a s, actualizando en este caso la posición del jugador. Entonces, cuando se supera el umbral x se llama a la función *updateMiniMap* en *UnityGlobals*. Esta función recibe como

parámetros la posición x, z (el eje Z funciona como un eje Y en un entorno 3d si éste es mirado desde una vista cenital) y luego normaliza estas coordenadas a valores decimales entre 1 y 0 utilizando los bordes del mapa, para enviarlos a HTML con una orden de ejecutar la función UpdateFromUnity, pasándole los parámetros x, y. En HTML se recibe la función, y luego las posiciones recibidas en formato string son transformadas a valores decimales. Luego se obtienen los bordes del mapa actualmente visible (localidad de Santa Olga). Luego se obtiene longitud y latitud. Para la longitud se obtiene el valor mínimo del mapa (el borde izquierdo) y se le suma la multiplicación del valor x enviado de Unity por el rango del mapa (borde derecho menos borde izquierdo). Para la latitud se obtiene el valor mínimo del mapa (borde inferior) y se le suma la multiplicación del valor y enviado de Unity por el rango del mapa (borde superior menos borde inferior). El resultado es un punto (x, y) transformado a longitud, latitud. Luego se crea un objeto del tipo *LatLng* (que es compatible con los marcadores de Google Maps) y se le da como argumentos la longitud y latitud obtenidas anteriormente.

Finalmente se ejecuta la función *UpdatePlayer* que actualiza la posición del marcador y recibe como parámetro un objeto del tipo *LatLng*. Esta función pregunta si es que existe el cargador del jugador en Google Maps. Si éste no existe entonces lo crea en la posición indicada. En caso que ya existía solamente actualiza su posición a la indicada por el parámetro que se le pasó al momento de ejecutar la función. Las figuras 2 y 3 muestras dos vistas satelitales del simulador con su respectivo posicionamiento en Google Maps.

Fig. 2. Vista satelital sobre el río 1.

Fig. 3. Vista satelital sobre el río 2.

Durante las pruebas de funcionamiento se adaptó el umbral de movimiento *s* con el fin de reflejar el mínimo necesario para que la actualización del mapa fuera un cambio visible. Esto evitará que la función sea llamada constantemente, sino cuando exista un cambio notorio a reflejar.

V. Resultados

Como se detalló anteriormente, la comunicación está efectivamente funcionando y se puede utilizar para diversos fines en el marco del proyecto. A continuación se muestran los resultados del uso de la comunicación en un caso real de simulación. En la figura 4 se muestra el simulador identificando puntos (cuadrados) de diversos tonos. El fuego es representado por puntos rojos. Las zonas de ceniza se representan por puntos rojo oscuro y puntos negros. Los rojo oscuro son zonas de ceniza donde puede reactivarse el fuego. Esto si bien se indica para contextualizar lo que está ocurriendo, no son más que instrucciones visuales para que el Web Player sepa qué estados van dependiendo de la zona de la matriz en la que se trabaja.

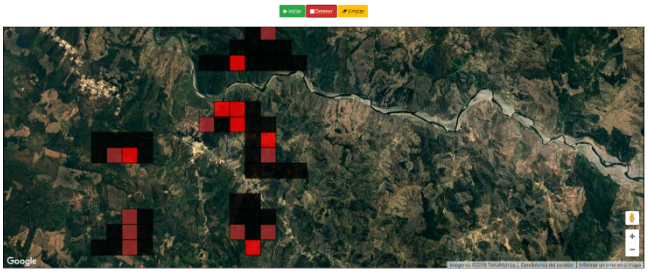

Fig. 4. Simulador vista con Google Maps.

Como se puede ver en la figura 4, hay una zona particular en la que cruza el río donde se pueden ver focos de incendio por la zona norte del río y por la zona sur. Esto se envía como estados en una matriz a Unity utilizando la comunicación desarrollada anteriormente. Los resultados se muestran en la figura 5.

Fig. 5. Simulador vista con Google Maps.

La principal diferencia entre usar solo Google Maps a implementar usando WebGL, es que se puede utilizar objetos y recorrer el sitio simulando una vista de primera persona. Además de que a futuro se planea la integración de multi-jugador sobre el escenario lo que permitiría que múltiples usuarios se conecten a través de la red y puedan conocer los distintos estados del escenario.

VI. Conclusiones

Se ha presentados algunos aspectos considerados para la implementación de la comunicación bidireccional entre un ambiente Web y el motor de videojuegos Unity.

Es preciso tener en cuenta dentro del sistema de comunicación que no se puede realizar una cantidad de consultas infinitas dentro de muy poco tiempo dado que todo se reduce a un "túnel" por el cual pasa cada petición. Las principales ventajas que se pudieron observar al usar este tipo de métodos es que aun cuando se pasan instrucciones es posible instanciar objetos o sistemas de partículas, lo cual beneficia el impacto visual del Web player. Aun así hay que considerar que el Web player no cuenta con una memoria ilimitada por lo que al momento de utilizar este tipo de métodos el usuario debe preocuparse de administrar de la mejor manera posible el gasto de memoria que ocurre dentro del Web player. De esta forma, el simulador no debería exceder el uso de la memoria reservada para el Web player. Esta memoria podría aumentar cuando hay muchos objetos en la pantalla siendo renderizados al mismo tiempo, como por ejemplo los árboles que son visibles en la Figura 5. Por suerte, la instanciación de árboles en Unity es muy avanzada, ya que utiliza una técnica llamada billboard que en lugar de renderizar todos los árboles, solo instancia los que están adelante dejando los de atrás como una copia de imagen de la renderización de adelante.

También se rescata la comunicación a otras páginas Web a través de otros métodos como la variable del tipo WWW, la cual puede ser de utilidad en otros contextos del simulador. Por ejemplo, mostrar encabezados de noticias importantes dentro del video juego, las cuales serían extraídas por comunicación con otras páginas Web. Un ejemplo podría ser acceder a datos del clima y mostrar un escenario con un clima que sea parecido al que se experimenta en la zona del mismo día. Finalmente, gracias a las nuevas tecnologías integradas en HTML5 ha sido posible llevar la Web a nuevos horizontes en un campo que aún tiene mucho por experimentar en el futuro.

Agradecimientos

Este trabajo recibe financiamiento del proyecto FONDEF IT16I10096 "Simulador basado en video juegos para respuesta a desastres".

Referencias

[1] G. Lavoué, L. Chevalier and F. Dupont, "Streaming Compressed 3D Data on the Web using JavaScript and WebGL" ACM. International Conference on 3D WebTechnology(Web3D), Spain, 2013.

[2] J. Gao, A. Huth, M. Lescroart and J. Gallant, "Pycortex: an interactive surace visualizer for fMRI" Front Neuroinform 2015. Pp 9-23, September 2015.

[3] Z. Andrioti, "Web 3D gaming over HTML5 and Web-based communication"(Master's thesis, Technological Educational Institute of Crete, Greece), 2015.

[4] O. Recatalà, "Real world to 3D terrain in Unity" Edited by Universitat Jaume I, 2017.

[5] J. Oak and J. Bae "Development of Smart Multiplatform Game App using UNITY3D Engine for CPR Education" in International Journal of Multimedia and Ubiquitous Engineering, vol. 9. No.7, 2014, pp 263-268.

[6] [6] R. Aldunate, O., Herrera, & Levano, M. (2015). "Video Game Script Design for Stability Assessment of Critical Physical Infrastructure Affected by Disasters". In Ubiquitous Computing and Ambient Intelligence. Sensing, Processing, and Using Environmental Information (pp. 492-498). Springer International Publishing.

[7] [7] O., Herrera, M., Lévano, M., Moreno, R. Aldunate, M., Bruno, M. (2014, November). Realistic terrain model in a wildfire context for El Yali reserve: serious videogame simulation. In Chilean Computer Science Society (SCCC), 2014 33rd International Conference of the (pp. 54-56). doi:10.1109/SCCC.2014.16. IEEE.