UNIVERSIDAD CATÓLICA DE TEMUCO

FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERÍA INFORMÁTICA

EXPANSIÓN Y COMPLEMENTOS DE ESCENARIOS DE CATÁSTROFE PARA UN VIDEOJUEGO SERIO

Por

GINO ALESSANDRO MOENA BARRAZA

Trabajo de Título presentado a la
Facultad de Ingeniería de la Universidad Católica de Temuco
Para Optar al Título de Ingeniero Civil Informático.
Temuco, 2018 -

UNIVERSIDAD CATÓLICA DE TEMUCO
FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERÍA INFORMÁTICA

COMISION EXAMEN DE TITULO

Este Examen de Título ha sido realizado en el Departamento de Ingeniería Informática:

Presidente Comisión: .....
Sr. Marcos Lévano Huamaccto
Magister en Ingeniería en Informática
Ingeniero Informático
Jefe Carrera Ing. Civil Informática

Profesor Guía: .....
Sr. Roberto Aldunate Vera
Ingeniero Civil Informático
Magíster en Ingeniería Civil
Magíster en Ciencias de la Computación
Doctor en Ciencias de la Ingeniería

Profesor Informante: .....
Sr. Oriel Herrera Gamboa
Director Escuela de Ingeniería Informática
Ingeniero Civil Industrial M/Informática
Doctor en Ciencias de la Computación

Jefe de Carrera de Ingeniería Civil Informática … …….
Sr. Marcos Lévano Huamaccto
Magister en Ingeniería en Informática
Ingeniero Informático
Jefe Carrera Ing. Civil Informática

Temuco, Julio 2018.

UNIVERSIDAD CATÓLICA DE TEMUCO
FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERÍA INFORMÁTICA

INFORME TRABAJO DE TITULO

TITULO: **“EXPANSIÒN Y COMPLEMENTOS DE ESCENARIOS DE CATÀSTROFE PARA UN VIDEOJUEGO SERIO”**

ALUMNO: **GINO ALESSANDRO MOENA BARRAZA**

En mi calidad de Profesor Guía, mis apreciaciones del presente informe de Trabajo de Titulo, son las siguientes:

- El trabajo desarrollado por el alumno se enfoca en el diseño e implementación de aspectos de componente 3Dpara escenarios de desastres (Incendio Forestal, inundación, Terremoto) y mejoras en la comunicación inter/intra modular sobre la arquitectura general del sistema. El contenido del trabajo presenta una profundización adecuada, la necesidad/problemática planteada de satisface con los objetivos establecidos, y a su vez los resultados logrados están acordes a estos últimos.

- La estructura y línea de trabajo se mantienen bajo los propios del proyecto FONDEF IT16I10096 que da contexto a este trabajo de título y según las pautas determinadas por la Escuela de Informática de la Universidad Católica de Temuco.

- El trabajo de título cubre los aspectos comprometidos de manera responsable, con calidad, trabajo autónomo y en constante seguimiento junto a equipo de trabajo proyecto FONDEP. El formato de informe se estructura y redacta en forma y contenido de manera adecuada.

- Este trabajo se encuentra presente en dos frentes. Por una parte, el componente desarrollo se enmarca en una implementación fundamental y necesaria para el proyecto FONDEF y su correspondiente avance; y paralelamente el desarrollo de este trabajo de título evidencia las competencias de la carrera de la cual esta emergiendo el alumno donde ha demostrado características y competencias que lo habilitan como Ingeniero Civil Informático.

De acuerdo a estas consideraciones califico el presente informe con nota con **nota 7,0 (siete coma cero).**

Dr. ROBERTO ALDUNATE VERA
Profesor Guía

Temuco, 13 de Julio de 2018

UNIVERSIDAD CATÓLICA DE TEMUCO
FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERÍA INFORMÁTICA

INFORME TRABAJO DE TITULO

TITULO: **"EXPANSIÓN Y COMPLEMENTOS DE ESCENARIOS DE CATÀSTROFE PARA UN VIDEOJUEGO SERIO"**

ALUMNO: **GINO ALESSANDRO MOENA BARRAZA**

En mi calidad de Profesor Informante presento, mis apreciaciones del presente Informe de Trabajo de Titulo son las siguientes:

- El trabajo desarrollado aborda un tema emergente en el área de video juegos serios. Se enmarca en el contexto de un proyecto real, vinculado a la construcción de un simulador para entrenamiento de personas que participan en la asistencia ante catástrofes naturales.
  Específicamente se trabajaron aspectos de mejora en performance y funcionalidad de los escenarios simulados. Se logra cumplir con todos los objetivos propuestos

- El trabajo es minucioso y acabado, alcanzando un nivel de detalle técnico de alta calidad.

- Se ve un especial énfasis en dejar una documentación exhaustiva del trabajo realizado, con una profundidad de detalle adecuada, generando un documento de buena calidad en formato y contenido.

- Como plus adicional, el trabajo fue sometido a la conferencia internacional CLEI 2018

De acuerdo a estas consideraciones califico el presente Informe con **nota 7,0 (siete coma cero).**

_____

Dr. ORIEL HERRERA GAMBOA
Profesor Informante

Temuco, 13 de Julio de 2018.

Dedico este trabajo a mi familia y amigos que me apoyaron durante todo mi proceso de educación superior.

A los desarrolladores de videojuegos de origen Latinoamericano que intentan día a día entrar en la industria de videojuegos.

También lo dedico a mis profesores que me formaron durante estos años en la Universidad en las distintas áreas que se me presentaron a lo largo de mi formación académica.

Finalmente, dedico este trabajo a las futuras generaciones esperando que sirva como ayuda en su aprendizaje en desarrollo de videojuegos.

AGRADECIMIENTOS

Agradezco al Profesor Oriel Herrera por ayudarme a ser parte del Proyecto y por apoyarme en las dificultades que se fueron presentando durante el mismo. También agradezco al Dr. Roberto Aldunate quien fue el que accedió a que yo fuera parte del proyecto junto a mis otros compañeros y me ayudó en las tempranas etapas del proyecto orientándome en los primeros problemas que se fueron presentando. Además agradezco a Cesar Navarro, por ser quien me guió y prestó ayuda en la mayor parte del desarrollo del proyecto prestándome material complementario para el desarrollo de este trabajo y ayudándome en los inconvenientes que fueron apareciendo durante la integración del trabajo en el Proyecto final.

También agradezco a la comunidad de Unity a nivel mundial que ha construido con su tiempo y experiencia un sitio web de preguntas y respuestas que ha ayudado bastante a a dar respuesta a problemas comunes al momento de entrar a la nueva plataforma de desarrollo de videojuegos.

Además agradezco a los profesores Alejandro Mellado por enseñarme la importancia del software libre y motivarme a desarrollar videojuegos en lugar de jugarlos; al profesor Luis Caro por potenciar mis habilidades como programador a un nivel que pensaba imposible; al profesor Gastón Contreras por enseñarme las nuevas tecnologías en desarrollo y acostrumbrarme al constante cambio en las nuevas tecnologías, también al profesor Ciro González quien me enseñó el fracaso y la importancia del estudio y también al profesor Marcos Levano por motivarme a mejorar la calidad de mis trabajos.

Finalmente a mi Familia, principalmente mi Madre quien fue la que me apoyo durante todo el proceso. También a Yassna Araya quien estuvo siempre conmigo apoyándome y motivándome durante los momentos difíciles en la Universidad.

INDICE

INDICE DE FIGURAS

INDICE DE APENDICES

RESUMEN

El presente trabajo busca dar apoyo y respuesta a soluciones gráficas en marco del proyecto FONDEF IT16i10096 “Simulador basado en videojuegos para respuesta a desastres”(CEININA, 2016) el cual tiene un foco principalmente en Incendios Forestales y Aluviones. Esto se logra a través del uso del Motor para desarrollo de videojuegos Unity. Las principales incógnitas que este trabajo busca descubrir es la viabilidad del uso de comunicación entre Unity y un sistema externo durante su ejecución, en este caso comunicación con JavaScript a través de una página Web. De esta manera, el presente proyecto complementa el trabajo hecho por otros estudiantes de la Universidad Católica de Temuco, en donde se mostrará una base en la comunicación entre Unity y HTML y además un complemento del sistema de Unity, en donde principalmente se mostrarán la expansión de un terreno (añadiendo objetos nuevos al sistema tanto por el desarrollador como durante su ejecución), para finalizar realizando una serie de experimentos que evidencian el funcionamiento de la comunicación Unity-JS para el posterior uso interno dentro del proyecto FONDEF IT16i10096. Es preciso además destacar el uso de sistema de partículas implementadas a matrices y principalmente el trabajo con matrices de tamaños dinámicos utilizando 4 terrenos de forma simultánea para representar un incendio en un escenario 3D. Se utilizó herramientas como TerrainComposer para el trabajo visual de los terrenos y se trabajó principalmente con objetos del tipo Terrain en Unity los cuales son usualmente conocidos porque tienen una serie de métodos y variables útiles para la manipulación de terrenos en un espacio gráfico.

ABSTRACT

The present project seeks to give support and response to graphic solutions in the context of the FONDEF project IT16i10096 "Simulator based on video games for disaster response" (CEININA, 2016) which has a focus mainly on Forest Fires and Floods. This is achieved through the use of game engine Unity. The main unknowns that this work seeks to discover is the feasibility of using communication between Unity and an external system during its execution, in this case communication with JavaScript through a Web page. In this way, the present project complements the work done by other students of the Catholic University of Temuco, It will build a structured method for communication between Unity and HTML for use in future developments and also a complement of the graphics system focus on the terrain visualization (adding new objects to the system both by the developer and during its runtime), to finish by carrying out a series of experiments that demonstrate the functioning of the Unity-JS communication for the subsequent one within the FONDEF IT16i10096 project. It is also necessary to highlight the use of a system of particles implemented in matrices and mainly the work with dynamic size matrices using 4 terrains simultaneously to represent a fire in a 3D scenario. Tools such as TerrainComposer will be used for the visual work of the land, and work will be done mainly with Terrain objects in Unity, which are usually known because they have a series of methods and variables useful for the manipulation of land in a graphic space. The result is a WebGL plugin that shows the scenario in question as well as communicating with HTML through JS.

CAPÍTULO 1. INTRODUCCIÓN

1.1 Resumen del problema

En Chile existen distintas catástrofes que se han vivido de cerca durante estos últimos años para las cuales la población y autoridades se han visto sobrepasadas respecto a la toma de decisiones apropiadas durante la catástrofe. Para esto, FONDEF ha financiado el proyecto IT16i10096 “Simulador basado en video juegos para respuesta a desastres” (CEININA, 2016) a ser ejecutado 2017-2019, y del cual el trabajo propuesto en este documento será parte. Específicamente, en el presente trabajo se mostrará todo lo relacionado a lo que es la visualización de catástrofes en Unity, pero principalmente la comunicación que se realiza entre Unity y la página web, esto ya que Unity forma parte de los distintos componentes con los que contará el proyecto y, por ende, como parte de un sistema más grande, éste necesita estar en constante comunicación con el resto de módulos que están presentes en dicho sistema. Finalmente, para manejar el contenido que se ve en el lado de Unity a través de WebGL se busca establecer un módulo estándar en las comunicaciones entre Unity y el sistema externo para facilitar el trabajo entre sistemas incluso en etapas futuras del proyecto al momento de realizar llamadas o cambios en Unity para quien desee experimentar con la comunicación entre sistemas. Además, se espera validar dicha comunicación en base a experimentación de la comunicación a través de distintos ejercicios que evaluarán la efectividad del sistema en los que se involucran escenarios que representan catástrofes naturales como incendio forestal y aluvión.

Palabras clave: Simulación de catástrofe, Unity 3D, escenario 3d.

1.2 Descripción General

1.2.1 Descripción del problema

El principal problema es el desarrollo de un módulo que facilite las comunicaciones entre el sistema de Unity en su formato WebGL con un sistema Web utilizando la comunicación ya existente utilizada por Unity, en donde la comunicación será posible a través de JavaScript y métodos que permiten la comunicación entre ambos sistemas, por lo que será necesario establecer una forma "predeterminada" para que ambos sistemas puedan realizar llamadas a métodos entre sí utilizando una vía específica que tenga el control del flujo de llamadas entre sistemas, por lo que será necesario crear un sistema emisor-receptor tanto en Unity como en el sistema web con JavaScript.

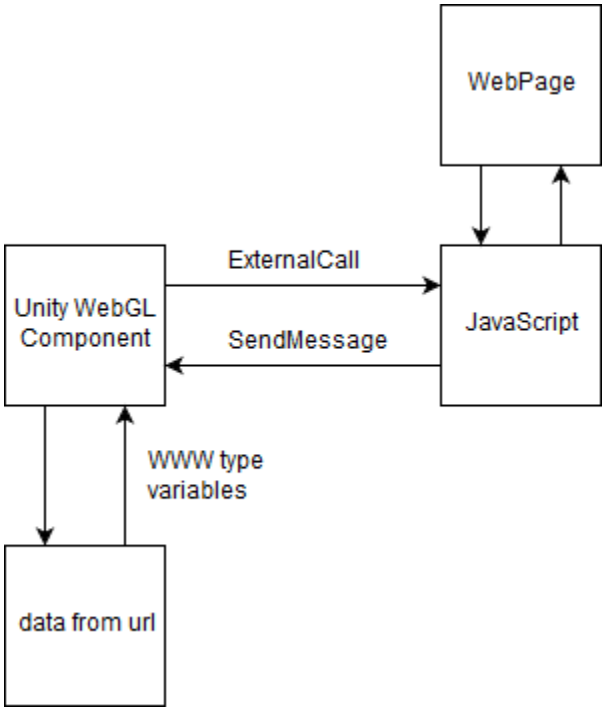

Figura 1. Esquema de comunicación Unity-JS.

En la figura 1 se muestra básicamente cómo la página web interactúa con Unity a través de JavaScript, además como complemento a eso se muestra que Unity puede extraer data desde cualquier URL. Esto puede ser de utilidad por ejemplo para extraer datos del clima desde algún servidor que diariamente refresca la data de una URL para indicar cómo está el clima. Esto resultaría en mostrar un día lluvioso en Unity o tal vez un día soleado, en un caso de uso para este proyecto, puede servir para enviar cantidades masivas de información (sin superar 1mb de datos) como podrían ser matrices de una dimensión moderada (256x256 por ejemplo). Como complemento a esto, se presenta como problema, la necesidad de mejorar la visualización de las catástrofes en las que se trabajará, por lo que será necesario trabajar con los escenarios agregando objetos, ambientación y buscando la manera de representar las catástrofes anteriormente mencionadas, como mostrar humo y/o fuego durante un incendio forestal, y mostrando propagación de barro o agua durante el aluvión.

1.2.2 Objetivo General

Diseñar e implementar aspectos de comunicación e interacción entre HTML, JavaScript y el motor de videojuegos Unity, para complementar la construcción de escenarios de videojuegos en el marco del proyecto FONDEF “Simulador basado en videojuegos para respuesta a desastres” (CEININA, 2016). Además, complementar la ambientación de escenarios de catástrofes como son incendio forestal y aluvión.

1.2.3 Objetivos Específicos

- Investigar acerca del uso del software Unity para el trabajo con datos externos a la plataforma, comunicación e Interacción entre plataformas Unity y HTML.

- Diseñar los aspectos técnicos y matemáticos para la interacción y comunicación entre el motor de videojuegos y la interfaz gráfica.

- Programar las funcionalidades necesarias para la comunicación entre HTML y Unity utilizando lenguaje JavaScript y C#.

- Realizar Test de funcionamiento del sistema para examinar posibles errores y mejorar la estabilidad del mismo.

- Agregar complejidad al terreno, agregando objetos como árboles, representar incendios y aluviones.

1.2.4 Resultados y producto esperado

Los resultados esperados para este proyecto son un módulo funcional que permita añadir profundidad a las capacidades de escalabilidad en escenarios dentro del proyecto FONDEF mencionado con anterioridad, y además la visualización de catástrofes a través de dicha comunicación. Dentro de dichas capacidades se espera complementar el uso del simulador utilizando la interacción y comunicación entre plataformas HTML y Unity establecidas como objetivo dentro del presente trabajo. El producto esperado será entonces una serie de funcionalidades que complementarán la interacción entre sistemas HTML y Unity, en donde a través de HTML se podrá utilizar API's como la de Google Maps que le darán una mayor

profundidad a la interacción entre sistemas, además del uso de algoritmos matemáticos para dar solución a problemáticas que se presentarán a lo largo del proyecto.

1.3 Antecedentes y Justificación

1.3.1 Antecedentes

Considerando la historia de Chile estos últimos 10 años podemos darnos cuenta que han pasado un sin fin de catástrofes que han afectado a la población, como se mencionó anteriormente está el caso de los incendios forestales, además del denominado terremoto del 27F (Aguevedo, 2010) y muchos otros más que los siguen. Entre otras catástrofes más tenemos los aluviones y derrumbes causados a partir de estos mismos. La técnica de Simular estos escenarios sería de gran utilidad para la población luego del historial de catástrofes que han afectado a Chile. Además, se sabe que actualmente no abundan sistemas que realicen este tipo de operación de simular una situación real de una catástrofe aquí en Chile. También, siendo Unity una plataforma en constante actualización es posible experimentar constantemente con nuevas tecnologías, entre ellas se han ido optimizando y han ido apareciendo nuevos formatos para trabajar con unity en su versión WebGL por lo que es actualmente se vive una etapa experimental de lo que es la comunicación entre Unity y HTML.

1.3.2 Estado de Arte

La simulación hoy en día se realiza de variadas maneras y la forma en que simulemos un sistema irá directamente relacionado con los resultados que esperamos obtener de ella, o sea, los datos de salida. Cuando queremos realizar una simulación enfocada a un público muy general lo ideal es evitar la complejidad del sistema para que este pueda ser interpretado correctamente por dicho público. Entonces la simulación que mejor se adapta a la necesidad de educar al público para responder ante una catástrofe es una simulación en un ambiente virtual gráfico en donde la persona maneje a un sujeto ficticio el cual se mueve y experimenta una situación de catástrofe dentro del mundo virtual. De esta manera, al plantear al usuario un ambiente que tiene un parecido a la realidad será mucho más fácil enseñarle y educarle, ya que se le está introduciendo a un ambiente del cual él ya tiene conocimientos en parte, el conocimiento restante sería el de cómo manejarse dentro de ese ambiente. Entonces, para la resolución del problema se realizará una simulación dentro de un videojuego, de esta manera se podrá aprender de forma interactiva como tomar decisiones ante una catástrofe natural o generada por el hombre, y la base para la resolución de este problema será el uso del motor gráfico Unity, que nos permite comenzar a trabajar en el simulador evitando crear el simulador desde cero utilizando alguna librería en C++ para la proyección de un mundo, por lo que a través de este método, se ahorrará mucho tiempo que se hubiera perdido si hubiésemos creado el motor gráfico desde cero por nuestros medios.

Las principales limitantes del problema será la falta de información acerca del problema en cuestión, ya que, a pesar de ser catástrofes conocidas de forma general, se desconoce particularmente las variables que afectan en ellas, por lo que para combatir dichas limitantes se deberá hacer una investigación que abarque la mayoría de las variables que pueden afectar

en estas catástrofes. Sin embargo, por la naturaleza de una simulación, a veces puede ocurrir que factores importantes sean pasados por alto haciendo que la simulación se pueda alejar cada vez más de la realidad, por lo que la investigación es un factor clave para el éxito del proyecto, de todas maneras, la metodología de trabajo de desarrollo en espiral permitirá una retroalimentación que puede cubrir ciertos aspectos que pudieron haber sido pasados por alto en etapas tempranas de desarrollo. Otras limitantes del proyecto será el tiempo del que se dispone para realizar las tareas programadas, ya que se tiene que cumplir con un tiempo límite para la entrega del sistema funcional. Finalmente, pueden existir limitantes referentes al uso del motor gráfico, ya que, al ser un software con sus propias reglas, hay que adaptarse a como estas funcionan para la correcta implementación del simulador dentro del motor gráfico Unity. No hay muchos proyectos de esta naturaleza, y el enfoque educativo es primordial para educar a la población ante catástrofe, más aún cuando se sabe que Chile es un país que ha sido afectado con diferentes catástrofes mayoritariamente esta última década. Por lo tanto, una herramienta que prepare a la población ante catástrofe será beneficiosa en el sentido del ahorro de los costos que puede generar un desastre natural. Tal como existe este proyecto, también existió un prototipo (CEININA, 2016) de lo que se esperaba realizar en este proyecto. Lo ideal es desarrollar un producto final en base a este prototipo. Esto se logrará cumpliendo una serie de funciones necesarias básicas para el correcto funcionamiento del producto final, dichas funciones le darán una diferenciación con respecto del prototipo y para el cumplimiento de ese objetivo cada integrante cumplirá un rol diferente en el proyecto. ¿Por qué el trabajo propuesto tendría que ser exitoso?, Porque se busca enseñar y dar seguridad a través de una simulación que simplifica los costos de llevar la operación a una escala real Y podrá ser utilizada por instituciones como el gobierno para educar a la ciudadanía. Dicho de otra forma, el proyecto tendría que ser exitoso por el objetivo principal,

que busca el beneficio para los habitantes de Chile, los cuales luego del uso de simulador deberán ser capaces de tomar decisiones más inteligentes ante una situación de emergencia para la cual ya han sido capacitados de forma virtual, por esto el éxito de este proyecto se verá reflejado en la supervivencia de la población y en la baja de costos después de una catástrofe.

Un nuevo método de realidad virtual basado en Unity3D

En la actualidad hay muchas herramientas a disposición de cualquier usuario para manipular gráficas de computadora, Hoy está de moda Unity3D, un motor gráfico que usualmente se trata como un motor de videojuegos. Este motor, debido a su gran dimensión, es capaz de emular distintos tipos de escenarios, lo que le da una versatilidad tremenda para crear una infinidad de escenarios dependiendo de la creatividad del usuario. A través de las herramientas que las grandes empresas proporcionan, se pueden realizar grandes cosas, en este caso, utilizando herramientas como 3dMax o Autocad, las cuales son herramientas de modelado 3D, se puede a través de un GPS crear un espacio 3D virtual utilizando parte del mapa, para dibujar el terreno, en este caso, se actualizará la capa de terreno también conocida en inglés como *Terrain Layer* la cual tiene por fin ser la "sabana" sobre la cual se ingresan los objetos dentro del motor. Dichos objetos son llamados *Assets*, los cuales complementan el escenario para que dé un aspecto más cercano a la realidad, si llevamos esto a la realidad, el *Terrain Layer* sería en este caso el suelo básico, sobre el cual está construido todo, o sobre el cual florece todo. Podría ser considerado la tierra. Mientras que los Assets son encontrados en capas como "Building Layer" o "Vegetation Layer". Este escenario se modela primeramente en un software de terceros como se mencionó anteriormente, ya sea 3dMax o

Autocad, y luego es importado a Unity para ser utilizado por este mismo. Unity permite con una tasa de refresco de 60 cuadros por segundo que el usuario al presionar una tecla para moverse permita ver este movimiento en acción justo en el momento después que la presiona, o casi inmediatamente, esto debido a la velocidad de computo interna de la CPU sobre la cual se utiliza la simulación, permitiendo una interacción en tiempo real con el entorno 3D que al usuario se le está presentando, de esta manera es ideal para mostrar simuladores en tiempo real que requieran una reacción rápida por parte del usuario para tomar decisiones ante catástrofes como ocurrirá en el proyecto al cual se dirige este trabajo de título.

Pensamiento Adaptativo y comando, videojuego de simulación para el entrenamiento de oficiales de fuerzas especiales

La resolución a problemas complejos que ocurren en tiempo real requiere una gran experiencia, la cual los militares o fuerzas especiales van adquiriendo a medida que se presentan eventos en los cuales deben proteger a su nación (Elaine Raybour, 2005). El simular estos eventos en la vida real puede ser muy costoso y en la actualidad los juegos de computadora crean un buen ambiente para estos propósitos. Por lo que utilizar la simulación para este tipo de ambientes es muy eficaz cuando se habla de entrenamiento de personas. Para la simulación de situaciones de riesgos, lo mejor es realizar una experiencia multi-jugador, en donde hay una serie de jugadores en distintos computadores que se conectan vía internet, LAN a un servidor para encontrarse en un servidor. En este servidor se pueden enfrentar de tal manera que al encontrar a un “enemigo” en el videojuego-simulador, este reaccionara dependiendo esta vez no como una máquina, sino como un ser humano. La toma de decisiones dentro de este tipo de simuladores puede resultar ser efectiva para un

enfrentamiento en la vida real hasta cierto punto, lo ideal es que el sujeto que es entrenado sea capaz de reconocer situaciones y relacionarlas con las que ha visto anteriormente durante la simulación para tomar decisiones que tomen el control de la situación y además tomarlas en poco tiempo, por temas de experiencia.

Interacción Unity-HTML

Unity como plataforma de desarrollo de videojuegos es muy versátil, esto se puede ver reflejado en la cantidad de plataformas en las que sus videojuegos pueden tener soporte. Unity trabaja con diversas plataformas las cuales son muy utilizadas hoy en día, incluso con consolas de videojuegos. En este caso particular, Unity tiene un plugin que permite ejecutar sus videojuegos en un navegador, esto ocurre gracias a las mejoras de HTML con la aparición de HTML5 y también con la existencia de un nuevo complemento llamado WebGL. Este complemento es una nueva tecnología que nace como una adaptación de OpenGL ES 2.0 para navegadores. OpenGL ES 2.0 es una adaptación de OpenGL 1.0 dirigida a múltiples plataformas, esencialmente a las consolas de videojuegos y móviles. Una de las características importantes que nos brinda Unity al ser utilizado con el Plugin WebGL es la posibilidad de interactuar con la página del navegador, a través de código JavaScript incrustado en el HTML y funciones dentro del plugin WebGL que están escritas en lenguaje C# las cuales en el proceso de compilación son traducidas a código C++ y luego compiladas para el plugin. Unity nos da dos funciones para la interacción entre el plugin WebGL y HTML, estas son “ExternalCall” la cual es utilizada desde el plugin para ejecutar una función en el HTML y la otra es “SendMessage” la cual es utilizada por el script en HTML para ejecutar una función en Unity. a través de esta información, Unity es capaz de enviar

información con HTML y viceversa. Un ejemplo, para demostrar la potencia de uso será el siguiente caso. Digamos que quieres probar el uso del envío de mensajes a Unity, estos llegan en una variable de tipo string, por lo tanto, se puede establecer una cadena de string que luego sea parseada por el plugin, con una cierta sintaxis que permita instanciar objetos dentro de Unity. Para la instancia de objetos podría utilizarse un objeto que no pasará por la etapa de renderización y quedará guardado en memoria, de esta forma, cada vez que se quiera instanciar un objeto simplemente se copia el que ya existe en memoria y luego se activa la renderización en el punto establecido. La sintaxis será "id, posX, posY, posZ" en donde id representa la id que identifica al tipo de objeto, en este caso podemos utilizar una id fija que sería "house" luego los tres parámetros siguientes son las posiciones que ocupa en el espacio tridimensional. Finalmente, esto se integra en un script en HTML y es probado para ver su funcionamiento. Finalmente, se concluye que Unity tiene un gran potencial para ser utilizado en distintas plataformas, además de tener un valor agregado dependiendo de cada plataforma, en el caso de WebGL, la interacción permite cambiar estados dentro del juego utilizando herramientas fuera de videojuego, expandiendo las posibilidades y creatividad de sus usuarios.

1.3.3 Justificación del trabajo en cuestión

Se quiere realizar un aporte al proyecto en cuestión expandiendo las capacidades del escenario dentro de la simulación. Este proyecto existe en el sentido de encontrar una metodología de preparar a la población ante catástrofes a través de un espacio virtual de simulación. La motivación personal para el desarrollo del trabajo es ampliar los conocimientos relacionados al trabajo con entornos gráficos para el desarrollo de videojuegos y/o simulaciones. Este trabajo tendrá un impacto importante ya que se intentará realizar una simulación de un problema que existe y es importante para la población. En caso de llevar a cabo el desarrollo del trabajo con éxito esta será una motivación para agregar mayores funcionalidades para dar más realismo a la simulación. Lo nuevo que mejorará el rendimiento del problema a abordar es un equipo de trabajo muy capaz que prestará ayuda necesaria de así serlo para mantener un buen flujo de trabajo.

1.4 Planificación y métodos

14.1 Requerimientos y Necesidades

Principalmente se requiere conocimientos moderados de lo que es programación en C# que es el lenguaje en el que se trabaja principalmente cuando se utiliza Unity, además conocimientos generales de Unity, lo cual se puede adquirir sin problemas con un par de lecturas introductorias al sistema, disponibles en su página web, además de muchos videos interactivos que ayudan a mejorar la comprensión del sistema.

1.4.2 Metodologías

La metodología de trabajo será la del desarrollo en espiral (Grijalva, 2017). Esta metodología utilizada por el proyecto Fondef IT16i10096 (propuesta por Boehm), es un proceso iterativo en donde el trabajo de investigación y desarrollo se produce mediante una iteración de fases las cuales cumplen un rol en específico distintos la una de la otra. De esta manera se minimizan los riesgos al momento de implementar el sistema.

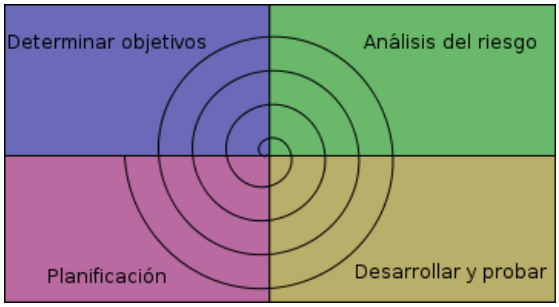

Figura 2. Modelo en espiral (Desarrollo en Espiral, s.f)

Como se puede ver en la Figura 2, el Modelo en Espiral funciona como un modelo iterativo en el cual se recorre un ciclo de etapas a lo largo del proyecto. Debido a la naturaleza del proyecto, será necesario la implementación de esta metodología de trabajo.

CAPÍTULO 2. MARCO TEÓRICO

2.1 OpenGL y Motor Gráfico

OpenGL es una librería gráfica de código abierto (*Open Graphics Library*) escrita en C (CEININA, 2016). Esta librería opera como una máquina de estados la cual son modificados por el usuario durante su ejecución. la implementación de OpenGL para un proyecto desde cero es muy compleja, ya que los requerimientos de comprensión básica de cómo opera el sistema son muy complejos, además de tener conocimientos de álgebra lineal para su correcto funcionamiento, hay muchas habilidades que debe poseer un programador para comprender como funciona la máquina de OpenGL. Por estas características, OpenGL se ha transformado en una herramienta que es añadida a un motor gráfico para su uso al momento de mostrar un espacio 3D o 2D. Entonces, usualmente, cuando un programador decide utilizar OpenGL de manera inteligente, este buscará siempre crear su propio motor gráfico implementando OpenGL en su motor como parte de un proyecto y en base a ese motor puede trabajar de forma particular para sus necesidades. Grandes empresas de videojuegos han desarrollado sus propios motores gráficos utilizando OpenGL como base para el desarrollo del motor, y más tarde lo han utilizado para la creación de videojuegos. Estos motores gráficos son privados, de tal manera que solo pueden existir dentro de la empresa y de ser utilizados por personas que no sean parte de la empresa puede significar en un problema legal por propiedad intelectual. A medida que han avanzado las tecnologías y las herramientas para la programación de gráficos de computadoras, ha nacido la necesidad de ayudar al programador promedio, o incluso usuario no programador a desarrollar su propio videojuego o virtualización de interfaz gráfica, de tal manera que nació el concepto del uso de motor gráficos gratuitos o de paga los cuales aíslan al usuario de la complejidad de OpenGL y su

máquina de estados para que este se preocupe de la programación básica y diseño de su programa con gráficas de computadora. Hoy en día existen varios motores gráficos, los cuales tienen buenas promociones para ser un negocio viable para sus creadores. Entre los motores gráficos más utilizados en la actualidad está Unity, Unreal Engine, CryEngine para juegos en 3D, y Construct 2, GameMaker, MonoGame para juegos en 2D (De la Rosa, 2017). Además, se puede mencionar que todos los motores gráficos en 3D tienen la capacidad de realizar videojuegos en 2D, sin embargo, la complejidad es mayor para lograr este objetivo. Por esto mismo los motores en 2D se han ganado su espacio como motores gráficos 2D dedicados. OpenGL es complejo, dicha complejidad ha sido añadida a través de los años en los que su desarrollo y actualización no ha cesado. Su primer lanzamiento fue en enero del año 1992 y hasta el día de hoy sigue se sigue actualizando, con la última versión 4.6 lanzada el 31 de Julio de este mismo año (2017). Como concepto básico de cómo opera OpenGL para transformar información en memoria a imagen visual en pantalla este realiza un proceso interno el cual pasa por un par de etapas en lo que se denomina como *graphics pipeline* que podría traducirse como tubo gráfico, este tubo es por donde el flujo de datos en memoria se transforma en la imagen en pantalla, por lo que este tubo tiene distintas etapas para procesar la imagen, de las cuales tres pueden ser intervenidas por un programador. Estas etapas de intervención tienen el sufijo *Shader*, las etapas por las que pasa una figura geométrica básica (triángulo) para ser mostrado en pantalla desde memoria son: *Vertex Data* (memoria) – *Vertex Shader* - *Shape Assembly* - *Geometry Shader* – *Test and Blending* – *Fragment Shader* – *Rasterization* (imagen en pantalla). La intervención que haga el programador en las etapas con Shaders lo realiza con la construcción de Shader propios, en los cuales se puede modificar como se ve todo el entorno. Por lo general para efectos visuales de entorno 3D se modifica el Vertex Shader, mientras que para efectos de post procesamiento se utiliza el

Fragment Shader, el cual mejora la imagen que se refresca múltiples veces en un segundo. Un motor gráfico puede ser muy útil para un programador ocupado que no tiene mucho tiempo libre para trabajar en su proyecto, o también para personas que deseen desarrollar un videojuego sin la necesidad de tener un conocimiento avanzado en los componentes más complejos que permiten la creación de un videojuego como es el manejo de memoria, manejo de eventos, ciclos por segundo, etc. Cuando uno decide comenzar un proyecto en donde se busca utilizar un espacio virtual en tres dimensiones o, mejor dicho, un espacio tridimensional, los resultados o primeros logros importantes en un motor gráfico son percibidos al comienzo del trabajo. Al tener incluso un espacio de muestra instantáneo donde se ve lo que se va realizando. Por otra parte, cuando se comienza un proyecto de cero utilizando OpenGL ocurre que los primeros resultados o logros son percibidos más adelante que con un motor gráfico. En un motor gráfico, mostrar un cubo en pantalla sin tener conocimientos del motor gráfico es una tarea fácil que en ocasiones el mismo motor inicia con una escena de esa forma, mientras que utilizando OpenGL desde cero para mostrar un cubo representa todo un desafío para alguien que recién está utilizando por primera vez la librería gráfica. Los beneficios y recompensas que ofrecen ambos caminos son indirectamente proporcionales a sus usos, mientras que el usar un motor gráfico para desarrollar un videojuego puede resultar más fácil, estos en ocasiones pueden limitar al usuario debido a las limitantes que pueden presentar en específico el motor gráfico para la naturaleza del proyecto del usuario, además, el uso de un motor gráfico pagado o gratuito pueden traer un costo monetario extra dependiendo de la cantidad generada por el juego en sí. Por otra parte, El desarrollo desde cero utilizando OpenGL representa un beneficio y recompensa a la dedicación del usuario, ya que al necesitar de mayor tiempo y conocimientos para su uso, este le permite crear su propio motor gráfico privado el cual puede utilizar más

adelante para la creación de sus videojuegos en donde el motor gráfico será creado en base a las necesidades del proyecto y no de forma genérica, en esta parte entonces se puede ahorrar en rendimiento y evitar tener módulos que no se usarán jamás, en otras palabras, el usuario tiene control total del motor gráfico. Además, las ganancias generadas por este mismo ya no pertenecen a la empresa que le proporcionó el motor gráfico, ya que el motor gráfico le pertenece a él. Un ejemplo es lo que pasó en el desarrollo de un videojuego que salió en el año 2015 titulado The Witcher 3. Dicho videojuego (independiente de la naturaleza del videojuego) tenía mucha complejidad en términos de ambiente y decisiones, y una de las ideas del director de dicho videojuego era incluir al videojuego múltiples finales. Estos requerimientos fueron pasados por alto en etapas tempranas del desarrollo del videojuego, el cual estaba siendo desarrollado en un motor gráfico comprado de uso público, este motor era Unity. Lo que ocurrió posteriormente en las etapas de programación del videojuego fue que, se llegó a una complejidad tal que el motor gráfico no soportaba las características que querían ser añadidas al videojuego de múltiples finales. Como consecuencia se tuvo que comenzar a desarrollar el videojuego desde cero utilizando un motor gráfico que tuvo que desarrollar la compañía para seguir con la producción del proyecto (Domínguez, 2017).

2.2 Unity

Unity es un motor gráfico desarrollado para el uso particular para estudios pequeños los cuales desean desarrollar videojuegos para su comercialización, una de las características de Unity es que es de uso gratuito, sin embargo, si se comercializa el videojuego, parte de las ganancias pertenecen a Unity, ya que se está prestando el motor gráfico se presta como servicio al desarrollador. Unity también tiene versiones profesionales las cuales son pagadas y cambian sus términos legales respecto a la publicación de videojuegos bajo la utilización del motor gráfico. Unity también trabaja con objetos dentro del juego y complementos, los cuales son también llamados Assets, estos objetos pueden ser desarrollados por uno mismo, pueden ser descargados y también obtenidos desde la Assets Store la cual es una tienda dentro de Unity que permite la compra de objetos para el posterior uso dentro de videojuego. Unity además permite desarrollar a múltiples plataformas como Android, HTML5, PlayStation, XBOX entre otras. Como se mencionó con anterioridad, Unity trabaja con la librería de OpenGL para su formato multiplataforma, en este caso particularmente para el desarrollo de aplicaciones web utiliza una extensión de OpenGL para páginas web llamado WebGL.

2.3 Matrices

Las matrices en Informática son arreglos de dos dimensiones, los cuales funcionan bien cuando se trabaja con espacios bidimensionales, o tridimensionales. Un ejemplo de esto, es la misma pantalla del computador. Una resolución de 1080x1920, por ejemplo, nos indica que tiene una matriz de 1080x1920 píxeles, y cada píxel almacena un dato de 32 bits de colores los cuales son distribuidos por 3 colores de 8 bits (Red, Green, Blue) y 8 bits

sobrantes. De esta forma, la pantalla del computador utiliza una matriz de 1080x1920 píxeles cada uno con 32bits de información. Gracias a esta matriz es posible para el ser humano ver e interactuar con la computadora de hoy en día. Las matrices además pueden ser de mucha utilidad cuando se trabaja un entorno 3D, en donde se utiliza una matriz de 4x4. Cuando se trabaja con estas matrices es posible transportar las partes de un modelo 3D a través de un mundo 3D evitando que haya partes del modelo que queden estancadas en el mismo lugar del mundo 3D.

2.4 Incendios Forestales

Los incendios forestales, son incendios que se manifiestan en zonas rurales de alta vegetación, estos amenazan con la destrucción de la flora y en ocasiones fauna, que les rodea. La gravedad de un incendio forestal varía dependiendo del control que pueda tener el ser humano dentro de la situación, en caso de pérdida total del control, las consecuencias de un incendio forestal pueden llegar a ser letales, como ocurrió en el caso de lo que ocurrió en febrero de 2017, en donde un pueblo se consumió a raíz de las llamas de un incendio forestal. "La localidad cuenta con una posta de primeros auxilios, un retén de Carabineros, un cuartel de Bomberos y un establecimiento de educación básica, el Liceo Rural Enrique Mac Iver, que impartía una modalidad científico humanista a una matrícula de más de 600 alumnos. Todo quedó destruido con el incendio." (Francisca Domínguez, 2017). Un incendio forestal puede ocurrir de diversas formas, entre ellas por obra del ser humano, en este caso el ser humano de forma descuidada puede provocar un incendio forestal debido a que utilizó algo inflamable o dejó encendido algo en un bosque que puede provocar el incendio, por ejemplo, al botar una colilla de cigarrillo dentro de una zona muy seca en un bosque puede provocar

un incendio forestal. Por otra parte, un ser humano puede iniciar un incendio forestal como acto terrorista, en el cual el ser humano está consciente de los actos que está haciendo con el fin de hacer algún mal a la zona que decide incendiar. Finalmente, un incendio forestal también puede ser provocado por la escasez de humedad y además la existencia de altas temperaturas en el ambiente. Se dice que en el caso de los incendios que tomaron lugar en la región del Bio-Bio el pasado mes de febrero de 2017 ocurrieron debido a las altas temperaturas registradas en la fecha, además de las sequías provocadas por la escasez de lluvia en meses anteriores. "Lo que se intenta al combatir un incendio forestal es romper la ecuación combustible (árboles) + comburente (oxígeno) + calor = fuego, eliminando al menos una de sus variantes."(Lopez, 2017). Cuando nos enfrentamos a un incendio forestal, hay factores claves que ayudarán a la extensión de este como, por ejemplo, humedecer la vegetación arrojando agua, aplicar algún retardante de fuego, construir zanjas corta fuegos para impedir la propagación del fuego e interrumpiendo la vegetación ya sea en sentido horizontal o vertical, cortando y sacándola del camino del fuego. Otro de los problemas que impiden la extinción del fuego es el calor y el humo, cuando llegan a un nivel insoportable es muy difícil a un ser humano extinguir el nivel del fuego, ya que estaría poniendo en riesgo su propia vida e incluso se vería incapaz de realizar un aporte a su 100% cuando se le está inhibiendo de visión y oxígeno para poder realizar la tarea de combatir un incendio forestal. Para esto existe equipo preparado de bomberos que intentan controlar la situación de fuego en áreas rurales.

2.5 Aluvión

Un aluvión es un flujo de barro el agua arrastra el material suelto por una ladera, quebrada o cauce. El aluvión puede viajar muchos kilómetros desde el lugar en que se origina, aumentando el tamaño a medida que avanza pendiente abajo, transportando hojas, ramas, árboles, rocas, y otros elementos, como pueden ser construcciones en caso de que este ocurra sobre un poblado. Este arrastre puede alcanzar gran velocidad por lo que es muy peligroso. El origen de los aluviones se genera principalmente por precipitaciones intensas en zonas de altas pendientes y quebradas. Ha ocurrido en el caso de Chile, que se han generado este tipo de flujo en sectores cordilleranos en donde haya existido un constante flujo de agua.

CAPÍTULO 3. DESARROLLO DE LA SOLUCIÓN PROPUESTA

3.1 Pasos en la metodología del como procede

En base a la metodología en la cual se está trabajando (Metodología en espiral) se ha hecho un continuo ciclo en el flujo de trabajo, de esta manera, se han ido realizando planificaciones, luego determinación de objetivos luego análisis de riesgos y desarrollar y probar. Una descripción de este proceso ocurre al momento en que se quiso realizar la comunicación Unity - HTML y viceversa, ya que ocurrió que se planteaba más de una solución al problema, ya se había planificado una tarea simple y se determinó como objetivo mostrar un texto en Unity cuando uno pulsara un botón en HTML. De esta forma, ocurrió que durante la investigación se descubrió que había más formas de hacer esto, por lo que se hizo un análisis en donde se enlistaron las posibles formas de realizar esto y finalmente se probaron mediante el desarrollo. El resultado fue que estas fallaron en su mayoría debido a problemas que estaban fuera del entendimiento del proyecto, finalmente la comunicación se limitó una comunicación a través de envío de mensajes entre sistemas, dicho mensaje es una variable de tipo String y es el sistema que se usaba en el prototipo anterior a este proyecto, por lo que fue mucho más fácil implementarlo y además este resultó ser muy efectivo. De esta forma se definió la comunicación a través de envío de mensajes de HTML a UNITY utilizando la función SendMessage y también al revés de Unity a HTML utilizando ExternalCall el cual envía mensajes de tipo String igual que SendMessage.

3.2 Requerimientos y Análisis

Para el proyecto se necesitó la instalación de Unity, en esta ocasión se utilizará Unity 2017.1.0f3 (64-bit). Para adquirir Unity en el computador basta con tener una conexión a internet e ingresar a su página web, desde ahí se descarga la última versión *community*, la cual es una versión dirigida a la comunidad. Esta versión es ideal para principiantes ya que tiene muchas herramientas y además es gratuita.

3.2.1 Requisitos mínimos para utilizar Unity

Para desarrollar en Unity se requiere de un ordenador que cuente con las siguientes

- Sistema Operativo, Windows 7 o mayor, y Mac OS X 10.9 o mayor.
- Tarjeta gráfica con compatibilidad de DX9 (modelo de shader 3.0) o DX11 con capacidades de funciones de nivel 9.3.
- El resto de componentes depende de la complejidad del proyecto. En este caso se utiliza una laptop con sistema operativo Windows 10, un procesador Intel i5 de tercera generación además de una tarjeta gráfica Nvidia GeForce GT 630m y 8 GB de RAM.

Durante pruebas del sistema Unity en el computador mencionado anteriormente se hizo un análisis del correcto funcionamiento del sistema mencionado. Una de las problemáticas que surgió fue los tiempos de compilación del trabajo, ya que este exporta a HTML el código en C# transformándolo a C++ y luego compilando todo el proyecto en una carpeta con un archivo Index ya que el plugin de Unity debe funcionar si o si en un navegador para cumplir las expectativas del proyecto FONDEF. Además de esto se realizaron un par de conclusiones respecto a la velocidad de compilación del sistema:

- La velocidad de compilación es de mínimo 15 minutos, por lo que se hace más difícil realizar debug a la página web debido a los tiempos de carga.
- Por suerte, si hay problemas de sintaxis estos pueden ser solucionados antes de compilar ya que Unity cuenta con una pantalla estilo *preview* para probar el videojuego en tiempo real, sin embargo, solo es posible probar el juego dentro de sistema Unity, por lo que si se quiere probar la comunicación si o si debe realizarse la compilación descrita anteriormente y hay que respetar los tiempos de compilación mencionados anteriormente.
- Una forma de solucionar los tiempos de compilación puede ser renovando el equipo, adquiriendo una CPU más actual con una mayor cantidad de núcleos y frecuencia, también utilizar un disco duro en estado sólido puede agilizar el proceso de compilación ya que la escritura de datos al disco es mucho más rápida además de los accesos al mismo.
- Debido a la complejidad del sistema Unity es recomendado o ideal trabajar con una pantalla en lo posible de una resolución 1080p, en este caso se trabaja con un sistema dividido de doble pantalla en la cual el sistema Unity utiliza toda la segunda pantalla (1080p) y el espacio de codificación utiliza la primera pantalla (768p).

3.3 Diseño del producto

En el diseño del producto se propone la prestación de servicios dentro de un proyecto en el cual se desarrolla el complemento funcional del mismo, donde se realiza un énfasis en el desarrollo en torno a la interacción entre sistemas, para eventualmente realizar un complemento en el diseño de la estructura de comunicación, de esta manera se busca no solo realizar un cumplimiento de tareas sino además entregar una correcta documentación además de un sistema bien estructurado para cumplir el rol de comunicación entre sistemas. Esta comunicación al ser bidireccional significa que el emisor también puede ser receptor, así y de esta misma forma el receptor es emisor también, un ejemplo que se podrá ver más adelante del producto es la clara comunicación bidireccional en donde se envía información desde Unity a HTML para actualizar la posición de jugador dentro del mapa, así mismo, en HTML se podrá cambiar la posición del jugador tan solo realizando un clic en la posición deseada, de esta forma existe una comunicación desde HTML a Unity, así ambos comunicadores cumplen ambos roles de emisor y receptor por lo que se realiza un diseño de emisor receptor tanto para HTML-JS como para Unity-C#.

3.4 Arquitectura del modelo

Atribuyendo a la arquitectura del modelo, este busca ser bien estructurado y documentado, ya que este será utilizado por distintas personas a lo largo de proyecto, se busca generar una estructura que sea comprensible y utilice una metodología simple evitando complejidad a no ser que estas sean estrictamente necesarias para cumplir la tarea de comunicación. La complejidad es justificada cuando esta da lugar a situaciones más simples, por ejemplo, una

función que sea capaz de procesar distintos tipos de datos será más compleja si se realiza una conversión que pueda aceptar distintos tipos de datos a la vez, pero esto añade simplicidad al sistema ya que se iría la preocupación de añadir un convertidor personal para cada función que quiera llamar a esta, ya que esta misma realizaría el trabajo de conversión de datos. Este tipo de problemas puede ocurrir por lo que se evalúa a fondo como es la arquitectura del modelo, para evitar problemas que puedan ocurrir en etapas tempranas de programación o que puedan ser ignorados y afectar en etapas más avanzadas de programación en donde ya sería más difícil realizar cambios en el trabajo. Además, cada evento desarrollado es descrito en un documento con ejemplos y este es almacenado en una plataforma que tiene un Kanban (sistema de información que controla la fabricación de productos). También, bajo esta misma plataforma es posible realizar un seguimiento de los trabajos desarrollados por los participantes del proyecto, de tal manera que agiliza la toma de decisiones en el quehacer luego de finalizadas las tareas de desarrollo para luego asignar nuevas tareas de desarrollo.

3.5 Diagramas de Trabajo

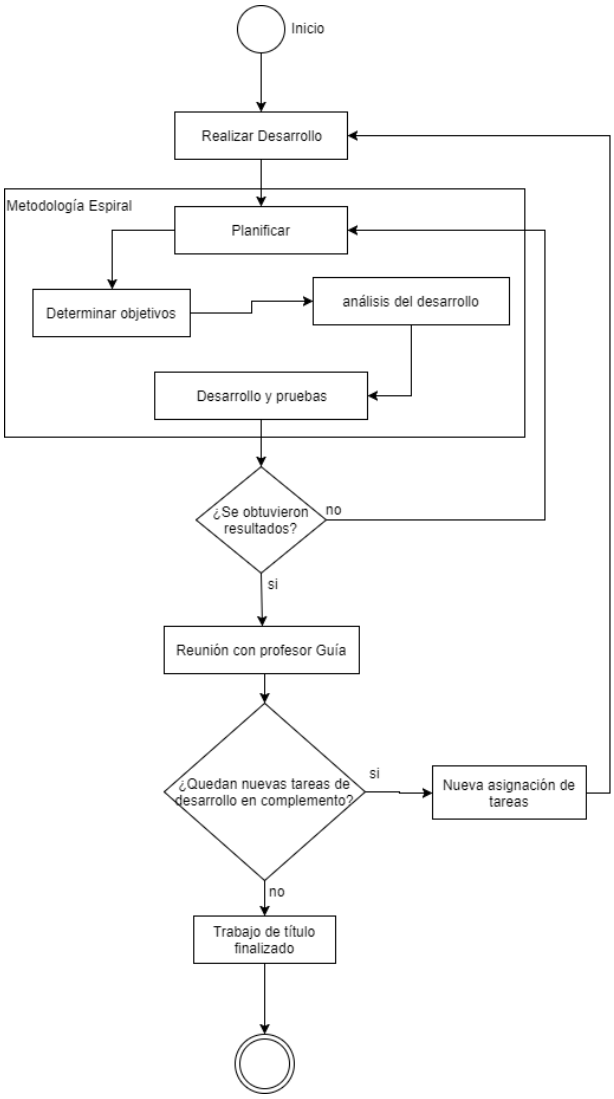

Figura 3. Diagrama de flujo y diagrama de metodología.

El diagrama de flujo de la figura 3 es del trabajo personal y resume las actividades individuales que se desarrollaran a lo largo del proyecto.

Los siguientes diagramas son propiedad del proyecto FONDEF IT16i10096 “Simulador basado en video juegos para respuesta a desastres”.

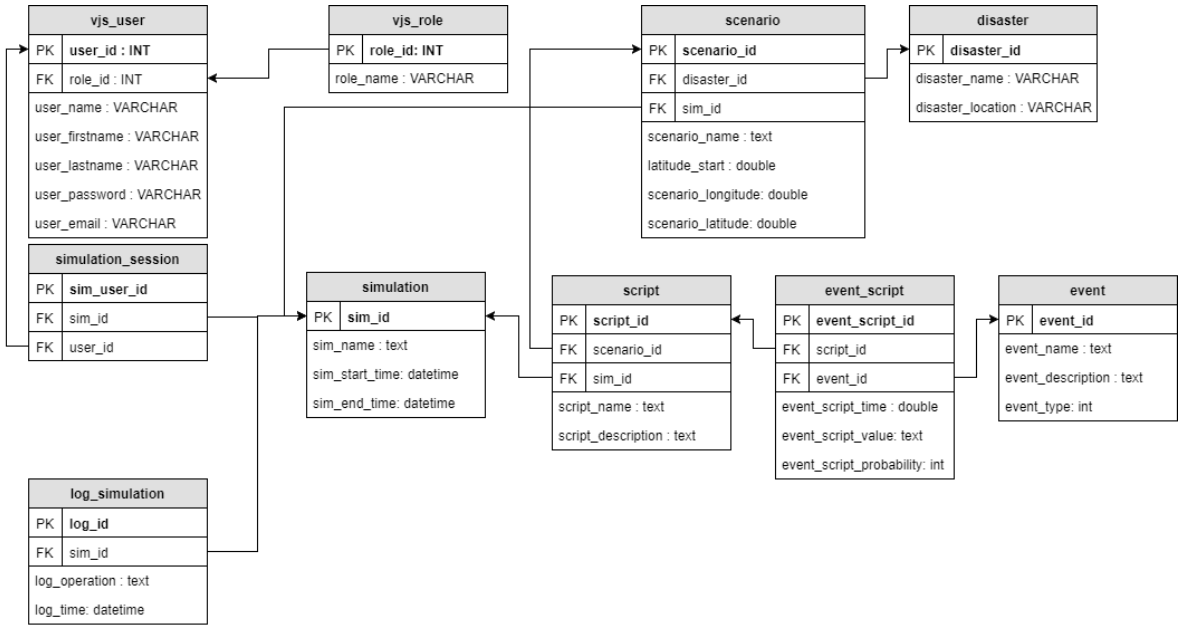

Figura 4. Diagrama de la base de datos FONDEF IT6I10096.

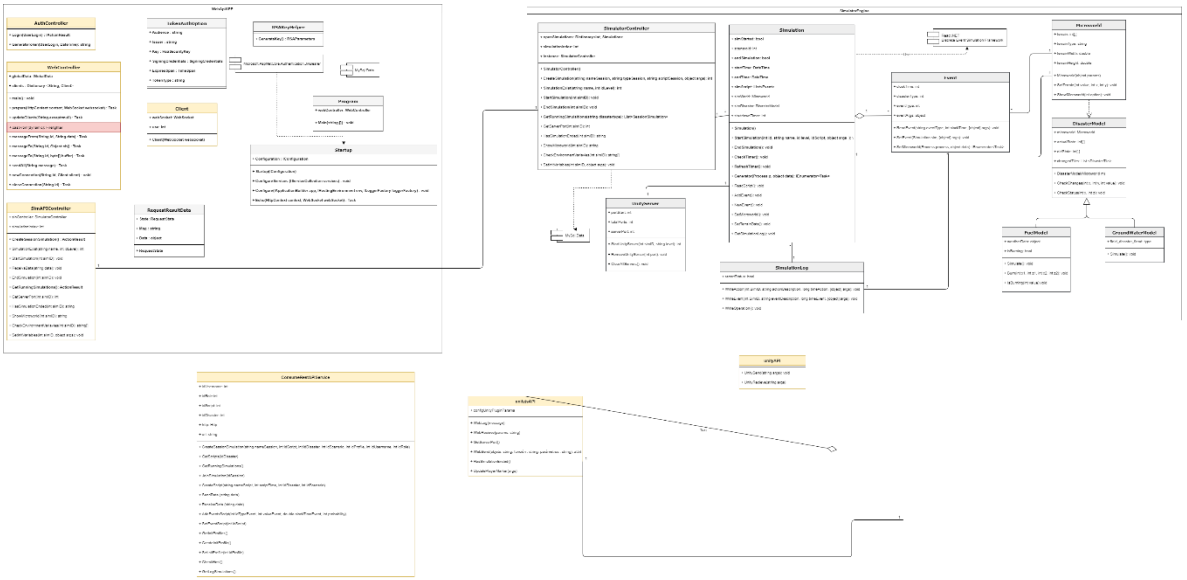

Figura 5. Diagrama de clases FONDEF IT6I10096.

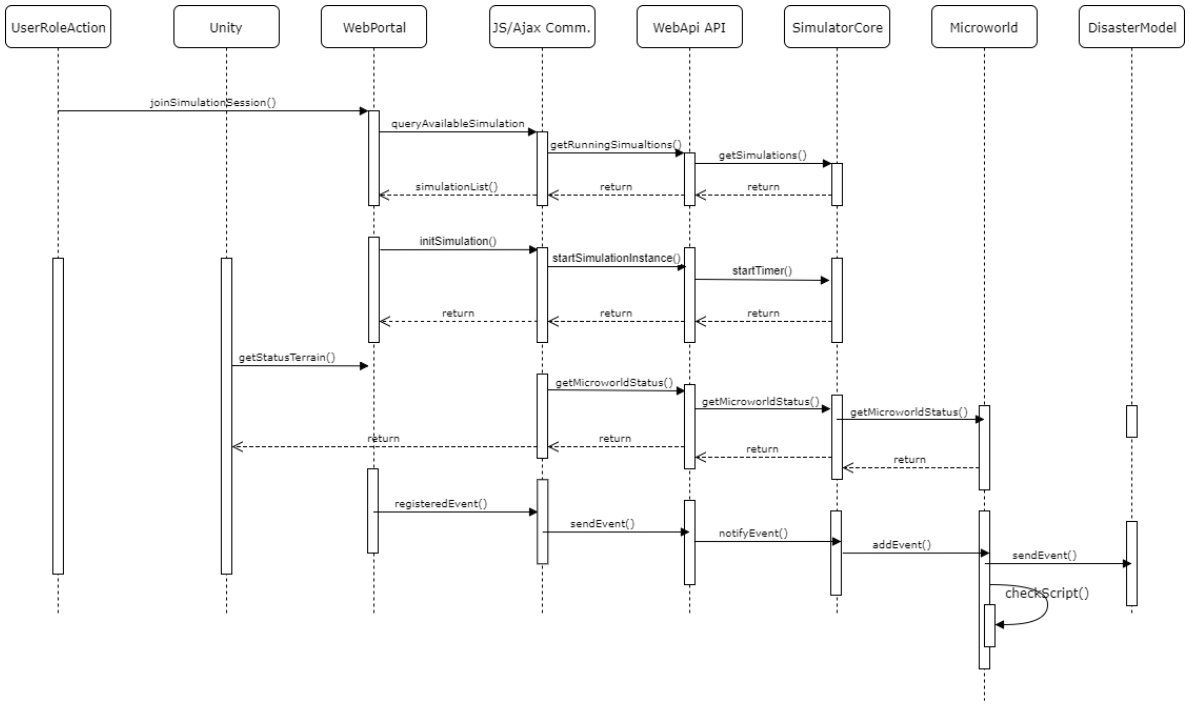

Figura 6. Diagrama de secuencia FONDEF IT6I10096.

Los diagramas de las figuras 4, 5 y 6 son parte del proyecto FONDEF IT16I10096. Estos diagramas demuestran la complejidad de desarrollo del proyecto “Respuesta a desastres a través de videojuegos serios”.

3.6 Diseño de experimentos y simulaciones

Unity es una herramienta de trabajo para el desarrollo de videojuegos, como es primera vez que se utiliza (en mi caso personal), es conveniente conocer su funcionamiento básico por lo que a continuación se realiza una prueba o simulación del sistema para tener todo en orden.

Para comenzar a utilizar Unity, se inicia el programa y se crea un nuevo proyecto, es opcional crearse una nueva cuenta para trabajar en proyecto personales, en este caso se utilizará de manera independiente sin cuentas asociadas, a no ser que sea necesario. Como es primera vez que se inicia, es de utilidad comenzar a conocer cómo funciona el espacio virtual de Unity. En la figura 7 se muestra como es la pantalla principal del editor de Unity.

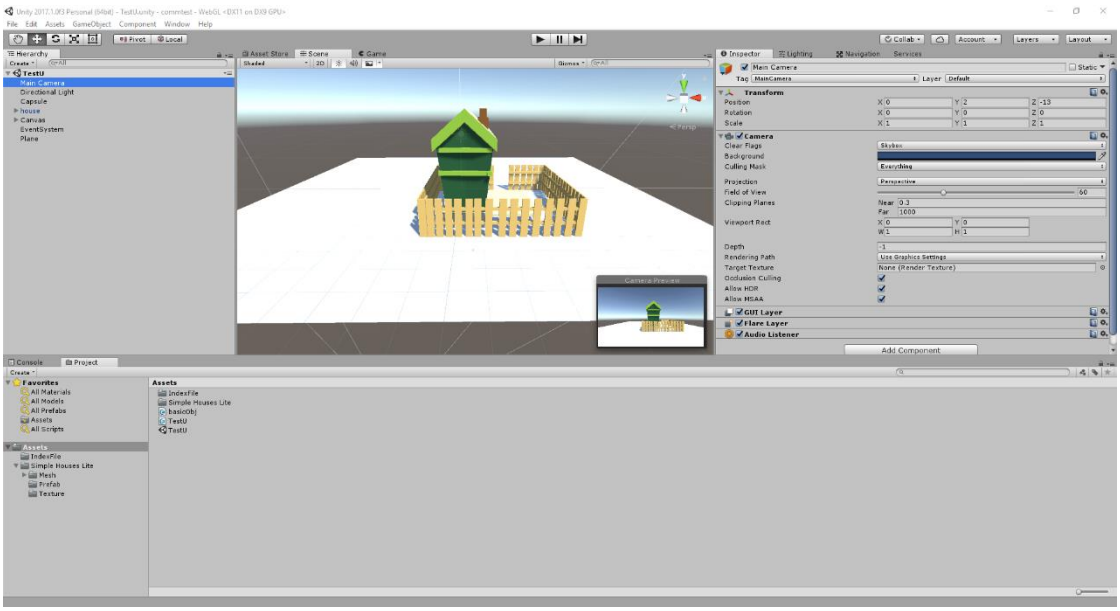

Figura 7. Unity Software

Al costado izquierdo del programa está la zona de objetos, estos son denominados *GameObjects* y se pueden incorporar más si se agregan al escenario que está en la zona central del software. Además, en la barra de navegación hay una zona dedicada a los

GameObjects. Desde ahí pueden incorporarse los GameObjects necesarios para utilizarse en el software. Esta Zona de objetos es denominada Jerarquía, en donde se muestra la jerarquía de objetos, de este modo se pueden agregar objetos dentro de otros objetos. Una forma de explicar esto es que existen coordenadas en el espacio tridimensional, y cada objeto tiene coordenadas en el espacio tridimensional, estas coordenadas son conocidas como coordenadas del *world Space* (del espacio virtual). Significa que cada objeto tiene una coordenada especifica en este espacio tridimensional, sin embargo, si se imagina un auto como objeto, se sabe que la complejidad del auto es distinta a la de una esfera, ya que el auto tiene componentes como las ruedas, los asientos etc. En este caso si se mueve el auto, se deberían mover todos los componentes que lo constituyen y no solo el parabrisas o las ruedas, para esto cada componente del auto tienen una coordenada específica en su espacio local, también llamado *local Space*. Y a través de una matriz (llamada *model matrix*) esta coordenada local es transformada a una coordenada global del *world space*. Entonces, en jerarquía, cuando se añade un GameObject dentro de otro las coordenadas que se apliquen a este serán realizadas en *local space* del GameObject pariente en lugar de ser aplicadas directamente al *world space*. Al costado derecho se encuentra el inspector de objetos. Este inspector muestra las opciones de cada GameObject una vez que es seleccionado. Todo GameObjects tiene un nombre único y además este puede incluir un Tag o estar en una capa específica (*Layer*). Los *Tag* y *Layers* pueden ser útiles cuando se necesitan agrupar objetos de cierto tipo para que cumplan un propósito particular. Por ejemplo, en un videojuego en el que se utilice la gravedad y plataformas, podría existir una capa llamada suelo, y dar una orden a través de un *script* que para cada objeto que esté en la capa llamada suelo sea sólido para el jugador, de manera que este no pueda atravesar dicho objeto y sirva como suelo para que este camine sobre él. Luego, cada objeto puede tener componentes, estos componentes

pueden ser creados por el usuario, como son los *scripts* que describen la lógica de videojuego y otros que son prefabricados por Unity. Por ejemplo, hay componentes que detectan colisión, estos son ideales para ejecutar eventos dentro de un videojuego a medida que se avanza. Por lo que finalmente en el inspector se pueden ver tanto las definiciones básicas del objeto como sus componentes. Al centro del software está la escena del videojuego, la cual aparece inicialmente con una cámara y una luz direccional. En esta zona se ve todo lo que tiene referencia al espacio virtual del videojuego. Se puede ver de forma libre la escena y además tiene pestañas para probar el videojuego y otra para comprar Assets los cuales son objetos que pueden ser añadidos al videojuego. Es posible navegar en esta escena utilizando el cursor para mover la cámara arrastrándola y realizando acciones de acercar y alejar. Finalmente, en la zona de abajo de Unity está la consola que puede ser utilizada para realizar Debug del videojuego y también la carpeta del proyecto en donde podemos interactuar con los elementos que contiene el proyecto además de crear más carpetas y scripts, los cuales pueden ser generados en formato JavaScript y en formato C#. Para este caso se trabajará con scripts escritos en el lenguaje C#. Además, cada Asset obtenido desde la Assets Store (tienda de objetos) serán añadidos a la carpeta del proyecto. De esta manera podemos arrastrar desde la carpeta hasta el escenario o la jerarquía de objetos para agregar más objetos a la escena del proyecto. Para comenzar entonces, se necesitará crear un script en C# para mostrar un mensaje en pantalla. Para realizar esto se utilizará una llamada externa desde HTML para ejecutar una función en Unity. Para la comunicación entre HTML y Unity se utiliza “ExternalCall()” desde Unity a HTML y “SendMessage()” desde HTML a Unity. Estas funciones llaman a una función dentro de cada sistema, dependiendo cual sea. En este ejemplo que se presentará a continuación se realizará una llamada sendMessage de HTML a Unity para verificar que exista comunicación entre sistemas, por lo que en HTML existirá un

botón que permite llamar a una función dentro de Unity, en este caso podría ser mostrar un texto en Unity, para esto se mostrará un ejemplo hecho para ir un poco más lejos y ver las posibilidades de la herramienta SendMessage.

3.7 Resultados Preliminares

Se realizará una prueba de funcionamiento entre sistemas utilizando SendMessage, a continuación, se realiza una descripción de lo que ocurrió durante las pruebas de la interacción entre sistemas. la función SendMessage es una función integrada y reservada para el envío de mensajes desde HTML a Unity. Como dice la función, esta se encarga de enviar mensajes a Unity, mensajes en forma de String (cadena de caracteres). De esta forma el mensaje se envía a una función existente en Unity creada por el usuario que como parámetro pueda recibir exactamente una variable del tipo String, por lo que puede resultar un poco limitante para la interacción de sistemas, por lo que depende del usuario cómo explotar esta función para sus necesidades. En este caso, para demostrar las cualidades de SendMessage se hará un pequeño programa en Unity el cual permitirá instanciar un objeto en distintas zonas de un espacio tridimensional en tiempo real utilizando SendMessage para enviar las coordenadas y el objeto en un formato de escritura en el cual se separarán mediante comas, además se integrará el ";" para separar los objetos, de esta manera se podría instanciar en un futuro más de un objeto, de manera de instanciar una lista de objetos, con el formato "objeto1, posicionx1, posiciony1, posicionz1;objeto2, posicionx2, posiciony2, posicionz2;.....;objeton, posicionxn, posicionyn, posicionzn; " De este modo, se pueden instanciar múltiples objetos utilizando muy poca memoria entre objetos por lo que puede resultar muy útil y no sobrecarga el canal de flujo de información entre los sistemas.

Una descripción general de lo que pasa en Unity es que se crea un objeto casa que es guardado en memoria y no pasa por la etapa de renderizado. Como se explicaba en el marco teórico, al no pasar por la línea de renderizado esta casa se mantiene en memoria, por lo que se quedará ahí por ahora. Luego se creó la función *spawnObject* en la cual se realiza el proceso de realizar un *parse* de la cadena recibida que ha sido enviada por HTML, entonces, luego realiza la tarea de instanciar un objeto en pantalla. Para esto, al realizar el *parse* se obtiene la primera cadena, la cual pertenece al nombre clave del objeto, luego se busca ese objeto y se realiza una copia del objeto que está en memoria, y luego se modifican las coordenadas del objeto nuevo reemplazándolas por las tres siguientes cadenas, que son transformadas a formato float y luego son ingresadas reemplazando su valor en "transform.position" por un nuevo objeto del tipo "Vector3" el cual tiene contenido adentro la posicionx, posiciony y posicionz del objeto. Finalmente, se aplica la función HideObject dentro del objeto copiado, para revelar nuevamente el objeto, dicha función ha sido creada por el usuario y cumple la función de hacer que el objeto realice una alternación en su etapa de renderizado, significa que, en cada ejecución de esta función, el objeto será renderizado si es que esta solamente en memoria, o será oculto y guardado en memoria si esta renderizado actualmente.

Para el funcionamiento correcto del Script que realiza todas estas acciones, es necesario que este Script sea asignado a un objeto que no vaya a interferir directamente con el entorno en el cual se realizan todos los trabajos, o sea que no interfiera con el entorno virtual. De esta forma su interferencia será sólo a través de la lógica del código escrito en el Script. Finalmente, al probar el programa se debería poder instanciar casas en un terreno en tiempo real utilizando HTML como instanciandor en lugar de utilizar directamente Unity a continuación en la figura 8 se ven los resultados de dos casas instanciadas.

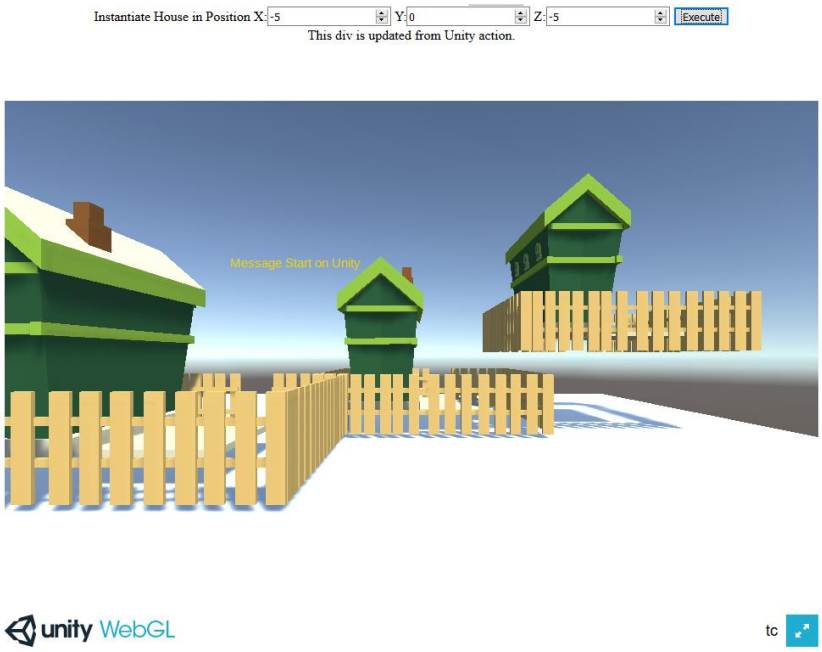

Figura 8. Ejemplo de Instanciación de objetos.

(este objeto casa fue obtenido desde la Asset Store de Unity, la cual es un objeto gratuito que puede ser obtenido por cualquier usuario que pueda tener acceso a la tienda).

CAPÍTULO IV. RESULTADO Y DISCUSIÓN

(algunos de los trabajos realizados, como la población de árboles están basados en tesis anteriores como "Modelo de desarrollo de escenarios para la simulación de incendios forestales mediante tecnología de videojuegos serios" de Matías Moreno, disponible en las referencias)

4.1 Integración al proyecto Utilizando Google maps y Unity en HTML para mover un auto dentro de Unity

Luego de las pruebas anteriormente realizadas es momento de comenzar a alinear los objetivos del trabajo de título con los requerimientos en el proyecto FONDEF IT16i10096 "Simulador basado en video juegos para respuesta a desastres", para esto se realizará un programa en el cual se integrará el uso de la API de Google Maps, la cual está hecha para trabajar con JavaScript dentro de una página web HTML. El único requisito para trabajar con la API de Google maps es obtener una API key para trabajar con el entorno. La API Key puede ser obtenida a través de la página dedicada para desarrolladores Google. Para trabajar con esta basta con instanciar dentro de un Script en el HTML y agregar la API key para que sea válido su uso. Una vez que se cumplen los requisitos se puede empezar a trabajar. A continuación, se realizará un sistema que buscará realizar una interacción entre los sistemas:

- Google Maps
- HTML
- Unity

Por lo que uno de los objetivos principales en esta sección es evidenciar la interacción de Google Maps y Unity, estos siendo como intermediario HTML. Para esto el sistema realizará lo siguiente:

- Mostrar un mapa y el plugin WebGL de Unity.
- El mapa está en un lugar específico (el lugar no es de importancia para lo que se realizará), además el lugar está bloqueado, de tal manera que no es posible mover el mapa ni realizar acciones de zoom en el mapa.
- Dentro del mapa se podrán poner dos *Markers* (pines que apuntan a una dirección en el mapa), estos *Markers* se pueden poner en cualquier zona del mapa de tal manera que se creará un punto inicial y un punto final.
- Al colocar el segundo *Marker* que representa el punto final, ocurrirá una acción en el HTML que enviará un mensaje a Unity indicando los puntos anteriormente señalados.
- En el lado de Unity, este tendrá una escena que mostrará un auto, este auto recorrerá la trayectoria creada por el usuario en Google Maps.

Al realizar todos estos puntos el resultado final será que el usuario al registrar dos puntos en Google maps se formará una trayectoria en el plugin de Unity en el cual el auto se moverá a una velocidad lenta para mostrar la trayectoria. Esta parte del trabajo se dividirá en dos secciones, la primera cubrirá todo lo referente a la API de Google Maps y HTML. La segunda parte cubrirá toda la parte de Unity, Finalmente se mostrará el resultado del trabajo.

4.1.1 Google Maps y HTML

Para comenzar, lo primero es autenticar la conexión a Google maps con la API key privada obtenida desde Google developers. Luego a través de código JavaScript incrustado en HTML se trabajará con la primera parte. Lo primero será utilizar la función reservada por Google maps “initMap()”, dentro de esta función se creará por el usuario un mapa utilizando un div reservado desde el HTML. Luego a este mapa se le asignan un par de reglas, entre ellas el “zoom” específico, evitar que se pueda realizar más acciones de “zoom”, no mostrar la UI (los controles de usuario) y evitar que el mapa pueda ser arrastrado para que muestre las ubicaciones adyacentes. Una vez dados los parámetros al nuevo mapa, se agrega una función para que se ejecute cuando se realiza un clic en el mapa, para esto utilizamos la función “addListener('click', function)” para llamar una función que realice algún tipo de acción cuando exista un clic en el mapa, en este caso, la función llama a otra función creada por el usuario llamada “states()” la cual se le pasa como parámetro la latitud y longitud del mapa en la zona que se hizo clic. Entonces la función “states()” realizará una evaluación para contar la cantidad de marcadores que existen en pantalla, esta evaluación dice que si la cantidad de *markers* es menor a dos se realicen acciones, en este caso si es uno se añade un marcador al mapa, en caso de que sea dos se añade el otro marcador al mapa y seguido a esto se ejecutan dos funciones nuevas, una es calcular y enviar información y la otra es borrar los marcadores. En la función calcular y enviar información se transforma los datos y luego se envían a Unity a través de la función SendMessage exclusiva para la interacción HTML - Unity. Google Maps trabaja con un formato distinto al usual utilizado para representar un vector x, y en la pantalla, tal como un mapa, Google Maps trabaja con latitud y longitud, el problema es cómo transformar latitud y longitud a puntos x e y para enviarlos a Unity. La solución es utilizar la

técnica de Normalización, para llevar dichos valores a un intervalo [0,1] en el cual podremos operar sin problemas posteriormente en Unity. Entonces para este proceso primero en el HTML se necesitará obtener los bordes o esquinas del mapa actual en Google Maps, para obtenerlos se utiliza la función "getBounds()". Una vez obtenidos las esquinas, se procede a obtener el punto inferior izquierdo, que representará el punto 0,0 en el plano cartesiano, y el punto superior derecho que representará el punto 1,1. Al obtener estos puntos se tendrán números distintos a los anteriormente mencionados, a modo ejemplo, los números deberían ser más o menos 40.0058, 56.068 para la zona inferior izquierda y 50.0058, 66.068 para la zona superior derecha. ahora lo siguiente que se realizará será obtener los dos puntos que representan los *Markers* realizados por el usuario anteriormente y luego realizar la técnica de normalización. Para esto, se aplica la siguiente fórmula:

$$Norm = \frac{X_n - X_{min}}{X_{max} - X_{min}}$$

nota: dado la naturaleza de esta fórmula, se asume que xmin debe ser siempre menor (y no igual a max)

dada esa fórmula podemos llevar un punto por ejemplo 43.008, 59.345 a un número entre 0 y 1 siguiendo los limites anteriormente señalados, dando como resultado 0.30022, 0.3277. Estos puntos son más flexibles para ser utilizados en Unity, por lo tanto, se realiza la técnica de normalización anteriormente señalada para los dos *Markers* y luego se ordenan y transforman en caracteres para ser enviados en una variable tipo String hacia Unity a través de SendMessage. Esto finaliza la sección de Google Maps y HTML.

4.1.2 Implementación en Unity

1. recepción de datos

En Unity los scripts funcionan como clases, las cuales tienen métodos reservados, como son “Start()” y “Update()”. El método “Start()” sirve para inicializar las variables en cuanto el script sea ejecutado, y luego no son ejecutadas nuevamente, luego el método “Update()” funciona como un “loop”, el cual es actualizado cada cuadro de avance en el juego. Asumiendo que el videojuego que se está construyendo tiene una tasa de refresco de 60 cuadros por segundo significa que este método “Update()” se actualizará o realizará un “loop” 60 veces en un segundo. Con estos datos en mente se sabe que en un momento se utilizarán ambos métodos para mostrar en pantalla al auto moviéndose de un punto (x1, z1) a otro punto (x2, z2), más adelante se explicará por qué se utiliza (x1, z1) en lugar de utilizar (x1, y1). Luego de lo realizado en HTML se ejecutará la acción “SendMessage()” que llamará a la función “recieveData()” en Unity, esta función realizará lo siguiente.

- separará la información recibida, como espera recibir 4 puntos separados por comas los separará y los ingresará a un arreglo de String los cuales para cada uno se aplicará la técnica de *parsear,* la cual transformará el tipo de variable de String a float la cual es el tipo de información que es de interés para su posterior uso.

- Luego, ya que estos datos han sido normalizados anteriormente ahora se debe configurar el nuevo espacio por el que se moverá el auto, el cual debe ser cuadrado igual que el mapa. Para esto se creará un plano nuevo en Unity el cual será un objeto poligonal parecido a una hoja o manto cuadrado que representará el suelo sobre el cual

el auto se moverá, luego obtenemos la posición de cada esquina en los ejes x,y,z. Sin embargo, para esta prueba se sabe que los autos no vuelan o no tienden a tener un movimiento en el eje y si la superficie es plana, por lo tanto, el auto se moverá en el espacio x, z ya que estos representan el movimiento horizontal. Entonces se obtienen las posiciones x, z de las cuatro esquinas del plano y son asignadas como x mínimo, x máximo, z mínimo, z máximo y además posición en eje y las cuales serán estáticas. y Finalmente se adaptan los valores normalizados a los bordes del plano, esto se realiza mediante un cálculo del rango de valores que se tiene en X y el rango de valores de Z y luego se multiplican los valores normalizados por dichos rangos y son sumados al valor mínimo de cada Eje, de esta manera los valores han sido transformados desde longitud latitud a normalización y de normalización a coordenadas dentro del espacio del videojuego.

- Para el movimiento, aplicando conocimientos del teorema de Pitágoras se debe calcular el triángulo rectángulo que representa la trayectoria del auto, dicho esto, se sabe que el auto irá de un punto inicial a un punto final, dicha dirección puede ser descompuesta en un vector (x,y) formando un triángulo rectángulo.
- Una vez obtenido estos datos se sabe la dirección en la que el auto irá, sin embargo, si el auto se mueve en esta dirección bastará que con un 1/60 de segundo llegue del punto inicial al punto final, lo que no sería un resultado muy bueno ya que se quiere mostrar una trayectoria a una velocidad normal que el auto realice, para mostrar movimiento e interacción entre las plataformas. Para esto se utilizará una técnica de cálculo avanzado llamada el vector unitario. El vector unitario es un vector que permite el movimiento de un objeto en exactamente una unidad en cualquier dirección. Por ejemplo, (1,0) es un vector unitario ya que se mueve exactamente una unidad, (0,1)

también lo es, sin embargo (1,1) no es vector unitario ya que se está moviendo más allá de una unidad. Para transformar un vector a vector unitario basta con tomar cada elemento del vector como dos catetos de un triángulo rectángulo y luego calcular la hipotenusa, luego el vector unitario será el mismo vector multiplicado por un escalar de la hipotenusa elevado a la menos uno. A continuación, se muestra la fórmula de lo explicado anteriormente.

$$c^2 = \sqrt{a^2 + b^2}$$

de esta forma, el vector unitario sería el vector por el escalar de la hipotenusa elevado a la menos uno.

$$(a, b) * \frac{1}{c} = \left(\frac{a}{c}, \frac{b}{c}\right)$$

Finalmente, como se puede ver estas expresiones representan la función de seno y coseno.

$$\left(\frac{a}{c}, \frac{b}{c}\right) = (Cos\theta, Sen\theta)$$

Siendo θ el ángulo adyacente al punto inicial y a, b la dirección x, z.

Entonces en base a lo dicho anteriormente se puede utilizar el vector unitario para que el auto se mueva en cualquier dirección exactamente en una sola unidad vectorial. Una vez hecho esto las variables son guardadas de forma global y se procede a realizar el movimiento, para esto, las últimas ordenes de la función que recibe la información es buscar el objeto auto, transformar la posición de dicho objeto a la del punto inicial y cambiar de estado de reposo a moviéndose, esto a través de una variable de tipo “bool” que pasa de un estado falso a verdadero. Como se dijo anteriormente, los scripts de Unity tienen los métodos ya

mencionados “Start()” y “Update()”, entonces en el método Start() se definieron una variables para configuración del objeto las cuales se describen brevemente.

- **isMoving:** una variable del tipo bool comienza en estado falso, y será utilizada para dar el pase a que el auto pase de un estado de reposo a en movimiento.
- **vel:** esta variable representa la velocidad a la que se mueve el auto, se dijo anteriormente que se utilizaría el vector unitario, el vector unitario es como una normalización muy útil que lleva cualquier dirección a su unidad, de esta forma al operar con la unidad se puede multiplicar por escalares que permiten alterar un valor que siempre será uno en todas las direcciones, de esta forma si el auto se moviera a la velocidad de 1 en 1/60 de segundo, este seguiría siendo muy rápido, por lo que dicho vector unitario será multiplicado por el escalar que es representado por esta variable vel. Su valor es 0.025f, esto significa que se moverá 1.5 en un segundo en cualquier dirección.
- **variables xDir, zDir**: estas serán variables que guardarán el vector unitario que generará la función que recibe los datos desde HTML.
- **variables xFinal, zFinal**: estas variables guardarán el punto final que es recibido por los datos desde HTML. Ya que el punto inicial es utilizado sólo para poner el punto inicial del auto este no necesita estar guardado en una variable, por otra parte, el punto final debe ser guardado en una variable global, ya que una vez termina la ejecución del método que hace la recepción de los datos desde HTML estas variables al ser locales se destruyen al finalizar el método. Luego en el método “Update()” existe una condición que solo se ejecuta si el auto está en movimiento, en caso de que el auto no esté en movimiento el método “Update()” no realizará nada. Dicho método hace lo siguiente.

- Encontrar el objeto auto.
- Transformar su posición sumando a su posición actual el vector unitario (x, 0, z) multiplicado por el escalar de velocidad vel.
- Transformar la variable *"isMoving"* de su estado verdadero a falso, esto lo hace mediante una condición que pregunta, si el auto está en la posición final entonces no debe seguir moviéndose. De esta manera finalizaría la ejecución del movimiento del auto.

Hasta este punto al realizar pruebas el auto efectivamente se mueve en la dirección que se requiere, sin embargo, falta un pequeño detalle, que es que no importando la dirección en que el auto se mueva al probar el programa, el auto siempre mira hacia el norte. Esto ocurre porque a pesar de alterar la posición del auto, no se alteró su ángulo de rotación. De esta forma el auto siempre mirará en una dirección fija, por lo que hay que corregir esto girando el auto a través del uso de Seno y Coseno.

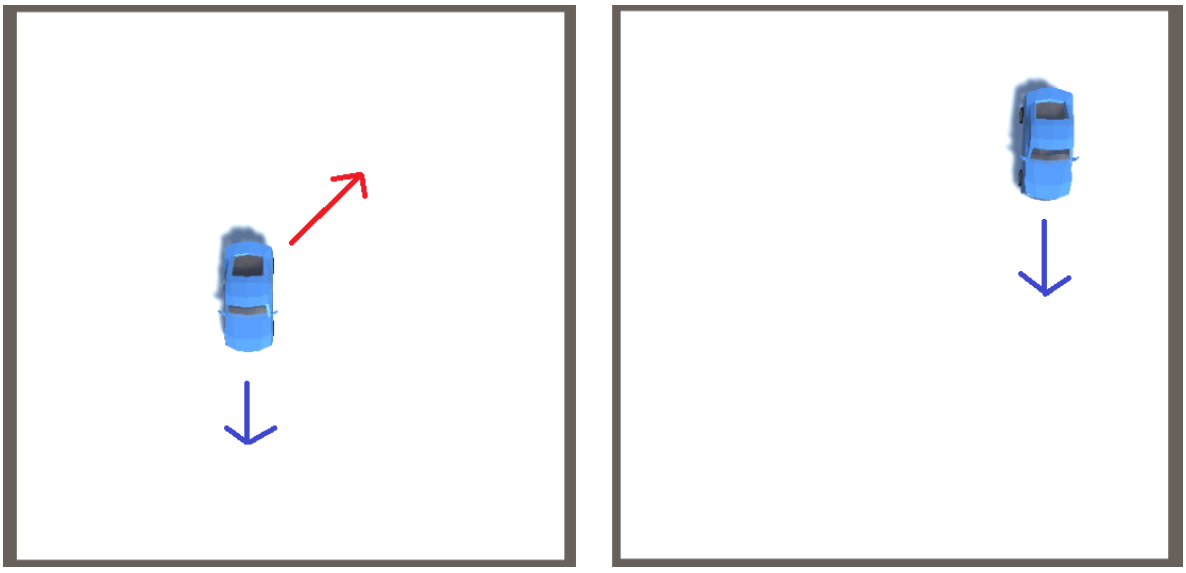

Figura 9. problema de dirección del objeto.

Como se puede ver en la imagen de la figura 9, el auto a pesar de moverse en la dirección correcta sigue mirando hacia el sur. La solución al problema que se ha presentado será rotar el objeto utilizando el valor de movimiento en el que se mueve en el eje X. En términos de rotación, Unity puede trabajar en radianes como también en ángulos, en este caso se utilizará ángulos. Entonces, si se sabía con anterioridad que xDir es el vector unitario, también se sabe que este representaba la función seno del ángulo de movimiento. Con esta información para obtener el ángulo original en radianes se puede utilizar el Arcoseno de xDir a través de la función de Unity "Mathf.Acos(xDir)" Además luego debe multiplicarse por el valor que transforma de radianes a ángulos, este valor es una variable de la librería Mathf, llamado "Mathf.Rad2Deg" (*Radians to Degree*, que significa radianes a ángulos). Sin embargo, el auto debería estar mirando hacia el norte para que esto funcionase, por lo que se le debe agregar una rotación de 180 grados para corregir la dirección del auto (en caso contrario, se vería que el auto está retrocediendo). Esto se aplica para cuando el auto se mueve por encima del eje X, para cuando el auto se mueva por debajo del eje X se debe aplicar la técnica del arco coseno y además agregar una corrección de 90 grados, con esta corrección el auto toma una dirección acorde a su movimiento.

Figura 10. Corrección de rotación.

Finalmente, como se pudo ver en la figura 10, se efectuó una corrección en su rotación que provocó que el auto se viera de manera más realista mientras recorre el camino trazado por Google Maps.

4.1.3 Conclusiones respecto al ejercicio realizado

Para finalizar, se pudo evidenciar una de las tantas posibilidades que ofrece el plugin de Unity, además de aprender un poco más de cómo funcionan los componentes de Unity tanto dentro del espacio tridimensional como por fuera cuando el plugin de Unity es utilizado a través de la interacción con el código HTML. Así como Google Maps Api interactúa con HTML y HTML con Unity, es posible utilizar más API's que hagan uso de JavaScript para continuar experimentando con la interacción entre sistemas.

4.2 Estandarización de los métodos de comunicación y organización de las funcionalidades de Unity

4.2.1 Desarrollo de la estandarización, consideraciones importantes.

Anteriormente se logró evidenciar la comunicación total entre HTML y Unity utilizando elementos de HTML5 como lo es la Api de Google Maps. Si bien el experimento tuvo buenos resultados, la finalidad para la incorporación de esta comunicación en el proyecto FONDEF IT16i10096 "Simulador basado en video juegos para respuesta a desastres" es que esta sea simple y pueda ser utilizada por más métodos, ya que existirá una gran interacción entre sistemas. Es por esto que, ya luego realizado las pruebas de conexión anteriores es momento de establecer una metodología de comunicación final que debería ser estándar, 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 el cual aplicar reglas similares, a continuación, se lista lo siguiente:

- De HTML a Unity al utilizar la función SendMessage para enviar el mensaje esta 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.
- Tanto en Unity como HTML cuando se hace un llamado al otro sistema se hace llamando a una función dentro de otro sistema y además enviando un String, de modo 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 entonces 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)” (UnityReceiver es la función contenida en el script UnityAPI que recibe la información). dicha línea de código será entonces la que enviará la información finalmente a Unity. Por ahora se dejará esta línea intacta, 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 que sirve para separar un texto mediante un delimitador. De esta forma, se propone una sintaxis general para el mensaje que se envía durante la comunicación. Ya que se busca realizar la mejor estructura posible del sistema se propone que al enviar un mensaje, este mensaje 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 decide que este será el formato final para el envío de mensajes. Bajo lo propuesto en los párrafos anteriores ya es momento de comenzar a desarrollar los métodos finales para la comunicación de HTML a Unity y de Unity a HTML. En HTML, se crea la función “WebSender”, dicha función 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.

Ya 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 ha terminado de realizar toda la concatenación y la ha guardado en una variable mensaje, ejecuta la función SendMessage que se armó anteriormente y se envía el mensaje en Unity. Ahora en Unity esta información deberá ser recibida. Por lo tanto, en Unity debe haber una función lista para recibir los datos y trabajarlos. Esta función llamada "UnityReciever" recibe entonces los datos en forma de String y por lo tanto 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 un algoritmo SWITCH-CASE el cual busca el caso en que la función se aplica, y una vez encontrado se ejecuta la función pasándole como parámetros lo que está al lado derecho del delimitador mencionado anteriormente. Así concluye la comunicación o envío de datos desde HTML a Unity. Para que esta comunicación sea efectiva es necesario que sea bidireccional y hasta el momento la comunicación ha sido establecida solo desde HTML a Unity, por lo que ahora se realizará el proceso inverso de Unity a HTML para establecer la bidireccionalidad en la comunicación de ambos sistemas, de modo que ahora Unity en lugar de actuar como un receptor esta vez será el emisor de los datos y HTML el receptor de datos. Entonces, ya que sea trabajó y conoció a fondo cómo funciona la comunicación de un lado, el proceso inverso es similar con unas pequeñas variantes debido a que el código es diferente. La función que enviará información a HTML

desde Unity es UnitySender y esta recibe como parámetro una variable de tipo String y luego una cantidad variable de argumentos del tipo “object”. El hacer que los argumentos sean de tipo object permite mayor dinamismo de los datos, ya que un arreglo del tipo object puede tener datos en binario, mientras que también tiene un texto o también una variable de tipo float por lo que da 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, este es finalmente enviado 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.

4.2.2 Conclusiones parciales del trabajo realizado.

Así concluye la comunicación bidireccional entre Unity y HTML. Adicional a esto se establece que para el orden de la estructura de la información, se crea un nuevo Script en Unity, esta vez el Script tiene como objetivo contener todos los métodos o en su mayoría métodos de uso general o de gran importancia para el proyecto, el Script llevaría por nombre “UnityGlobals” y cada función llamada en el SWITCH-CASE de Unity redireccionaría a una función contenida en “UnityGlobals”, además de esto, “UnityGlobals” contendrá información importante para el proyecto como por ejemplo 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 en la cual se realizará un proceso más gráfico donde se trabajará con parte de proyecto realizado por otro desarrollador dentro de proyecto FONDEF IT16i10096 "Simulador basado en video juegos para respuesta a desastres", en el cual se realizó el manejo de cámaras y objetos para moverse por un mapa que representa una situación de desastre de incendio forestal ubicado en Santa Olga, región del Maule. Es posible ver la codificación de la comunicación entre Unity y HTML en el Anexo 1 para conocer la comunicación desde el lado de Unity. En el Anexo 2 para ver funcionalidades complementarias para la comunicación por el lado Unity, como la transformación de dimensiones desde HTML a Unity. En el Anexo 3 para ver la comunicación por el lado de HTML utilizando Javascript.

4.3 Google Maps como GPS para Unity, Utilización de los Estándares de Comunicación bidireccional entre Unity y HTML

En esta sección se realizará un experimento el cual consiste en reflejar la ubicación del usuario de Unity en el mapa de Google Maps dentro del navegador, de modo que cada vez que este se mueva se actualice la posición en Google Maps. Para realizar esto se necesita obtener la posición del usuario en el mapa de Unity y luego enviarla de manera normalizada hasta Unity (revisar ecuaciones de normalización en sección anterior en el apartado de Google Maps y HTML). 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 Google Maps simulando como si este fuera un GPS. Durante el desarrollo de esta sección se utilizará el proyecto desarrollado por uno de los alumnos que ha trabajado para el proyecto FONDEF IT16i10096 "Simulador basado en video juegos para respuesta a desastres" en donde se puede experimentar con movimiento dentro de un mapa utilizando distintas perspectivas, entre ellas está la vista en primera persona, satélite y un modo de exploración a través de un helicóptero. Para este ejemplo se utilizarán 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, esta ha sido reconstruida utilizando complementos de Unity como es "Terrain Composer" y" World Composer" que permite reconstruir imágenes satelitales. El mapa tiene una dimensión de 7970x7970. 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 creará un nuevo script el cual será agregado al objeto que se mueva en el mapa. Este script ejecutará una función de UnityGlobals que realizará una llamada a HTML a través de la Api

de Unity para la comunicación la cual fue desarrollada en secciones anteriores. Para la llamada hay que establecer puntos importantes, y es que esta 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 hace falta tan solo 60 *frames* para hacer un segundo de juego, o sea que significaría 60 llamadas a HTML por segundo. Por tanto, considerando la dimensión del mapa, se establece un umbral de movimiento, de modo que si el jugador se mueve muy poco no realizará ninguna llamada, hasta el punto en que el jugador se mueva lo suficiente para registrar un movimiento significativo y entonces realizar una llamada a HTML avisando que se debe actualizar la posición del jugador. De este mismo modo si el jugador está en reposo no se realizará nunca una llamada lo cual puede resultar muy favorable para la comunicación. Entonces una vez superado el umbral 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 este es mirado desde arriba hacia abajo o mejor dicho con 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 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 que actualmente se ve, el cual está posicionado en la localidad de Santa Olga. 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” envía 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 este no existe entonces lo crea en la posición indicada, en caso de que ya existía solamente actualiza su posición a la indicada por el parámetro que se le pasó al momento de ejecutar la función. Luego se realiza la compilación del sistema en Unity y se lanza con el respectivo archivo HTML que contiene el container de Google Maps. Los resultados se evidencian a continuación:

Figura 11. Vista Satelital.

Como es posible ver en la figura 11, se ubica la cámara en un punto estratégico del río para evaluar si la cámara está exactamente donde debería estar, luego se realiza un desplazamiento de forma manual por el usuario para ver si el mapa de Google se actualiza. Además, se realizan pruebas con otros elementos de la cámara como es la cámara del helicóptero, como

se puede ver en la figura 12, el nuevo punto refrescado de Google Maps ha sido refrescado en base a la nueva posición del Helicóptero en el web player de Unity.

Figura 12. Vista desde Helicóptero.

(Durante las pruebas de funcionamiento se adaptó el umbral de movimiento con fin de reflejar el mínimo necesario para que la actualización del mapa fuera un cambio visible.) Esto permite verificar el buen funcionamiento del sistema. Para probar la comunicación de forma inversa (mover algo en Unity desde HTML) se realizará un proceso más el cual será cambiar de lugar el helicóptero trasladándolo al lugar que se desee haciendo un clic en el mapa de Google Maps. De esta forma se evidenciará totalmente la comunicación bidireccional dentro del experimento dando lugar al cumplimiento de objetivo principal de esta sección. Entonces lo primero será agregar un "Listener" en el mapa que realice una acción cada vez que se haga un clic en el mapa. Esta acción será ejecutar nuevamente la función UpdatePlayer para actualizar el marcador para que apunte directamente donde se

hizo el clic, la otra función será ejecutar la función UpdateUnityPosition la cual tiene por finalidad actualizar la posición del jugado en Unity a través de la comunicación HTML-Unity. Entonces la función UpdateUnityPosition recibirá la longitud, latitud del lugar al que se ha realizado el clic, luego mediante la normalización transforma estos datos a valores decimales entre 0 y 1. Finalmente se establece el nombre del objeto al que se le modificará la posición en Unity, en este caso será "HelicopterModel" que corresponde al objeto del helicóptero dentro de Unity. Una vez hecho esto se ejecuta la función WebSender para enviar la información a Unity. Una vez en Unity se llama a la función changePosition y se aplica la normalización inversa para que los valores entre 1 y 0 sean expandidos hasta los bordes del mapa. Luego se busca el objeto al que se le quiere modificar la posición (en este caso, el helicóptero) y se guarda su posición Y ya que sólo se modificará la posición X, Z y finalmente se actualiza la posición con los nuevos valores. A continuación en las figuras 13 y 14, se ven los resultados, en donde se presionó el clic en zonas estratégicas del mapa para notar el cambio. En este caso el helicóptero se trasladó a la pequeña isla ubicada en medio del río y luego se posición en la esquina superior derecha del mapa.

Figura 13. Helicóptero en la isla.

Luego de estar posicionado en el río el usuario al hacer clic en la zona superior derecha del mapa puede ver reflejado una traslación del helicóptero hacía la esquina del mapa.

Figura 14. Helicóptero en el límite superior derecho del mapa.

Conclusiones del experimento

Como se pudo evidenciar anteriormente, la comunicación bidireccional se concretó a través de un ejemplo que permitió simular el uso de GPS en un helicóptero para una localidad real de Chile. Gracias a la estructura del código y de la comunicación de Unity las funcionalidades que hicieron uso de la comunicación no tuvieron mayor complicación de desarrollo debido a que la complejidad de comunicación quedó aislada en el script UnityAPI. De esta forma la comunicación bidireccional es efectiva y práctica para su uso durante el desarrollo del proyecto. A través del ejemplo del helicóptero se espera que esto sea utilizado posteriormente para simular otro tipo de vehículos o tal vez brigadas de bomberos que se desplazan hacia algún lugar determinado en el mapa recibiendo órdenes del mismo usuario que realiza acciones en el navegador para ver como estas se desenvuelven dentro del plugin de Unity.

4.4 Complemento de Escenarios, adición de capas filtro, ambientación del terreno

Por necesidades del proyecto FONDEF IT16i10096 "Simulador basado en video juegos para respuesta a desastres" se ha requerido como complemento al trabajo presentado una guía experimental para la adición de objetos a escenarios dentro del editor, estos objetos le darán mayor vida al escenario y además tendrán influencia en lo que sería más adelante la interacción de escenarios con los desastres naturales. Parte de esta guía está basada en los conocimientos entregados por Matías Moreno en su tesis "Modelo de desarrollo de escenarios para la simulación de incendios forestales mediante tecnología de videojuegos serios" presentada en la Universidad Católica de Temuco en el año 2014. Para complementar la comprensión de lo que a continuación se hará se recomienda una lectura previa al trabajo de escenarios con TerrainComposer de esa tesis.

Para comenzar se utilizará el mapa de Santa Olga en el cual se hicieron las pruebas de comunicación en el experimento anterior. Lo ideal será verificar que la ventana de TerrainComposer está activa. TerrainComposer es una herramienta complemento de Unity la cual permite trabajar con superficies y agregar capas filtro, por ejemplo agregar árboles, rocas, entre otros, a un escenario utilizando métodos avanzados para que estos sean agregados de forma aleatoria, además de la posibilidad de trabajar con máscaras las cuales permiten hacer un recorte de espacios en donde no se quiera agregar vegetación debido a que esta pudiera ser inexistente en la zona (por ejemplo, un árbol en medio de un río, pasto en el desierto, etc.). A continuación, en la Figura 15 se muestra como referencia el mapa actual además del complemento terraincomposer.

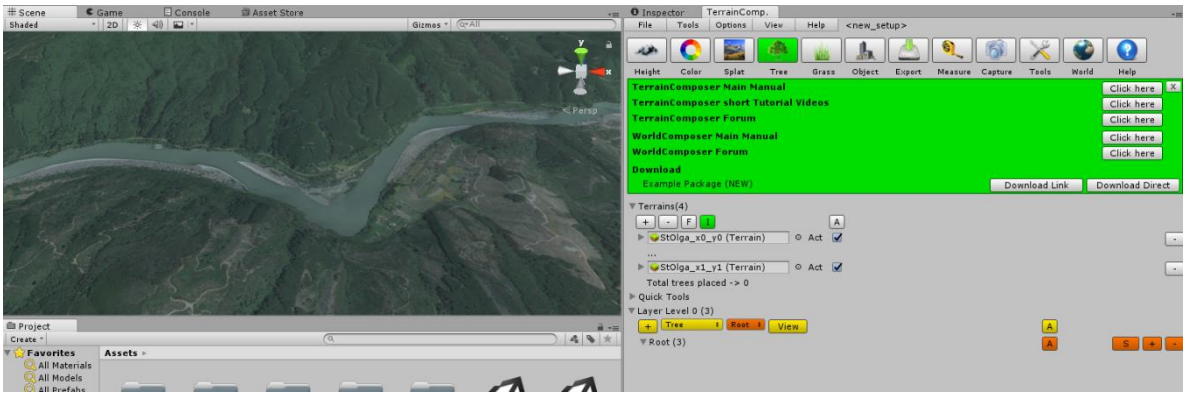

Figura 15. Mapa sin objetos y TerrainComposer.

Una vez que esté todo en orden se puede comenzar con el proceso de la instanciación de objetos en el escenario. Para esto primero hay que establecer los objetos que serán tratados como árboles en el escenario. En este caso será necesario obtener árboles, posiblemente cualquier modelo de árbol genérico puede servir como los incluidos en el paquete de Terrenos de Unity, el cual puede implementarse dentro del proyecto a través de Assets>Import Package>Environment(Dentro de todas las características de importación se importará sólo la descrita como Environment). Una vez incluidos se puede comenzar a trabajar en la forestación del escenario. Lo primero es ubicar el terreno sobre el cual se trabajará en terrainComposer en la pestaña “Height”, como se puede ver cada terreno tiene sus pestañas con opciones que son únicas a ese terreno (“Local Area”, “Size”, “Resolutions”, “Grass/Details”, “Trees” entre otros), en este caso se irá a la pestaña Trees del terreno (no confundir con la pestaña Trees que sirve para trabajar directamente con las capas de árboles). En la figura 16 se puede ver la intefaz gráfica del complemento “Terrain Composer”.

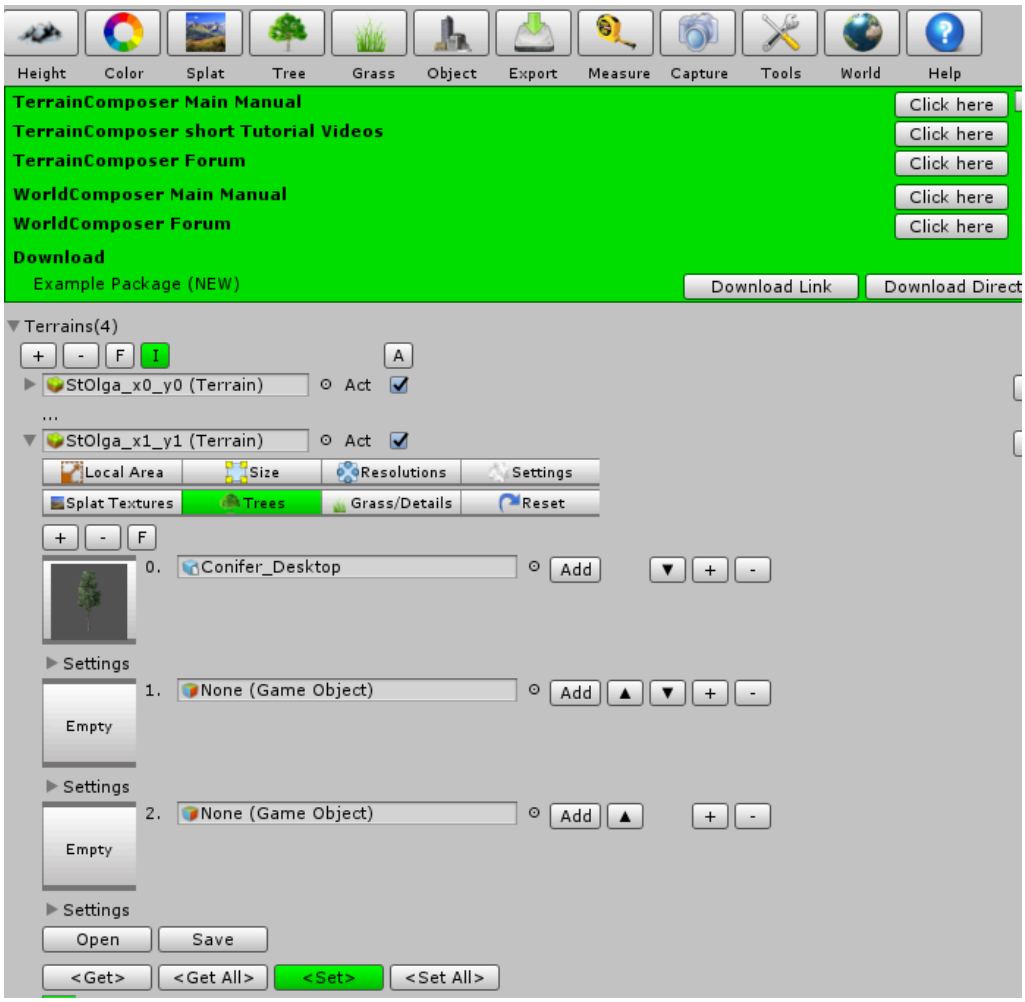

Figura 16. Interfaz terreno para asignación de modelo de árboles.

Se pueden asignar múltiples árboles a un terreno, de esta forma estos serán puesta de forma aleatoria tanto de lugar como de tipo de árbol. Una vez se han escogido los árboles (presionando en el circulo pequeño al lado derecho del Textbox del objeto) se procede a presionar el botón shift y clickear en el botón "<Set>" para aplicar los cambios y los árboles queden registrados para el terreno indicado. En caso de que haya múltiples terrenos que tengan el mismo tipo de árbol es posible aplicar la asignación de árboles a los múltiples terrenos repitiendo el proceso anteriormente descrito pero esta vez presionando en el botón

“<Set All>”. Ahora se debe cambiar a la capa de Árboles, en la pestaña Tree (se asume de manera deductiva que el cambio de pestaña sólo cambia el tipo de capa en el que se trabaja por lo que lo realizado anteriormente podría haber sido realizado en la mayoría de las pestañas y no solo en la pestaña Height). En la capa árboles se debe presionar en el botón “+” en la zona de Layer Level. Esto agregará una nueva capa a Root (que está debajo de Layer level), como se puede ver hay una especie de jerarquía (tipo árbol en informática). Esta capa comúnmente llamada Layer n(tree) (siendo n un número autoincrementable) contiene bastantes opciones las cuales sirven para la forestación con árboles al escenario. Dentro de la jerarquía de Layer se deben agregar los árboles en la zona de árboles, en donde al presionar “+” deberían aparecer los árboles que pertenecen al escenario (en caso de que esto no ocurra es porque se ignoró algún paso anterior a lo que se lleva de este experimento). Una vez aparezcan los árboles se va a “Filter (1)” y se expande el árbol “Tree Select0”, aquí se muestra el filtro de como se hace el llenado de árboles sobre el escenario. Para este caso se cambia el Input a Random. Luego más abajo en la zona de máscaras se agregan dos máscaras. Las máscaras sirven tanto como para llenar como para borrar árboles del escenario, esta técnica funciona a través de distintos métodos. En este caso y ya que da mayor facilidad al trabajar, se trabajará con máscaras en base a imágenes, en donde al utilizar una imagen con colores blanco y negro por ejemplo se puede restringir el mapa a que solo se instancien árboles en las zonas en que la imagen tenga color blanco. Este mismo proceso será repetido más tarde por la instanciación de pasto en el escenario. El procedimiento de forestación a través de máscaras esta vez será un poco distinto a como se realizó originalmente en la tesis de Matias Moreno (2014). Esta vez la primera máscara tendrá como Input el valor Random, mientras que el Output será Max. Bajo estas reglas el escenario completo será llenado de árboles. Luego se crea otra máscara más que será la que permita sacar los árboles de lugares donde

no debería haber, por ejemplo, ríos, zonas secas, etc. Entonces para esta máscara se utilizará como Input una imagen y para Output la opción Subtract que se traduciría como “borrar árboles”, entonces, la primera capa llenará de árboles el escenario mientras que la segunda borrará o cortará todos los árboles que está en zonas que no corresponden. Ya que se está trabajando con un mapa de 2x2, podemos utilizar distintas imágenes para cada zona, en este caso se recortarán árboles solo en una zona del mapa, que justamente es la zona que recorre el helicóptero en el experimento anterior. Entonces, las imágenes que se utilizan para el recortado son imágenes editadas con Photoshop por un usuario para establecer los lugares a los cuales se le quiere cortar la vegetación, en este caso todo lo negro será lo que se recortará del escenario, mientras que lo blanco será la zona que tendrá árboles. Finalmente para indicar a TerrainComposer cuál es el color que se quiere sustraer del mapa hay que agregar al “Color Range” un nuevo objeto presionando el botón “+” y verificar que el color sea negro. A continuación, en la figura 17 se muestra la capa a utilizar para la “deforestación” del escenario.

Figura 17. Máscara de Recorte.

A continuación, en la figura 18 y figura 19 se muestra el trabajo realizado en las máscaras y en las figuras 20 y 21 se muestran los resultados.

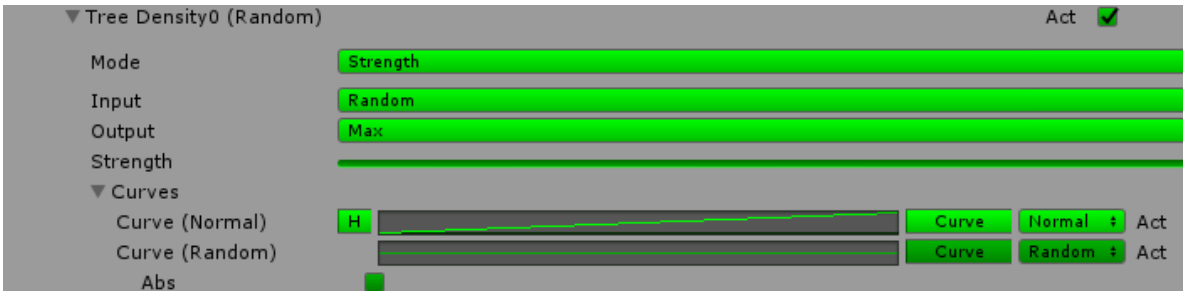

Figura 18. Configuración Máscara de Forestación.

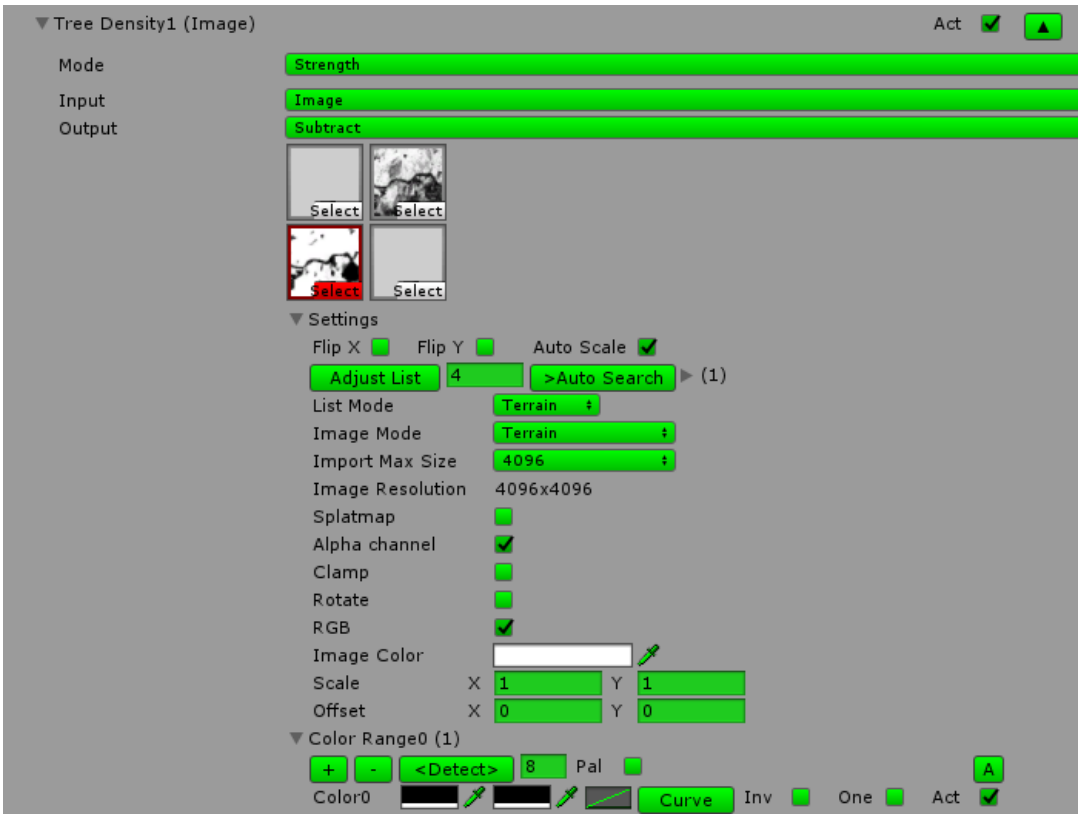

Figura 19. Configuración Máscara de Recorte.

Finalmente, y una vez realizada la configuración anteriormente señalada, se presiona el botón "Generate" en la esquina inferior izquierda del panel para generar árboles. Además, mediante

el mismo proceso será posible agregar pasto al escenario. A diferencia de los árboles, la resolución de pasto se ve en la pestaña “Resolutions” en lugar de “Local Area”.

Resultados y Conclusiones

La forestación de árboles al generar lleva tiempo, dependiendo de la cantidad de árboles que se quiere plantar en el escenario (además del tamaño del escenario). Al realizar el experimento, la generación de árboles demoró 3.24 segundos, sin embargo, los resultados no fueron tan buenos como se esperaba, ya que los árboles estaban muy alejados los unos de los otros. La solución a este problema fue corregir la cantidad de árboles en la pestaña Local Area de los escenarios. En este caso se aumentó la población de árboles cambiando el valor de Tree Resolution desde 128 a 512 causando también que la generación de los escenarios tomara un poco más de tiempo en ejecutarse (48.5 segundos). A continuación en las figuras 20 y 21, se muestra los resultados del experimento mostrando una comparación de árboles con resolución de 128 contra árboles con resolución de 512.

Figura 20. Árboles con resolución de 128.

Figura 21. Árboles con resolución de 512.

Cuando es mayor la resolución de los árboles se puede notar una mayor población, esto da un impacto visual mayor a la anterior, ya que se nota que existe un bosque poblado de árboles, en cambio en la otra figura parece más como un campo con un par de árboles esparcidos alrededor.

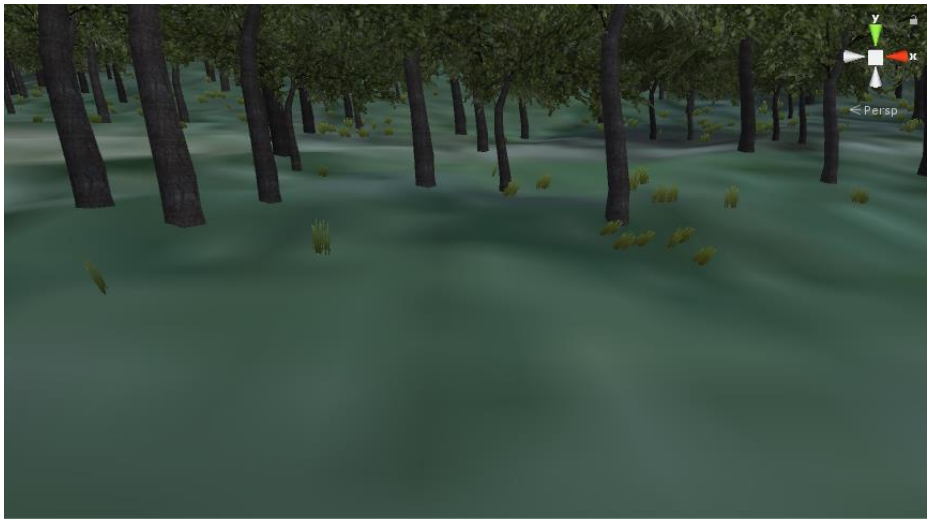

Figura 22. pasto en escenario.

También se ha agregado pasto al escenario como se muestra en la figura 22, sin embargo se considera que el impacto visual del pasto no es significativo para lo que se quiere mostrar en el proyecto. Finalmente, y como complemento al experimento, esto permitirá más adelante realizar una representación de incendios forestales, de modo que se pueda ver como los árboles se van quemando de manera gradual, de esta forma se buscará con el impacto visual dar mayor realismo a lo que el usuario ve durante la toma de decisiones. Además, en base a este procedimiento será posible más adelante realizar algo similar en territorios como Tierra Amarilla en donde es posible que en lugar de muchos árboles haya pocos, y además se vea la presencia de otros objetos como rocas (siendo este un terreno más desértico).

4.5 Simulación de un aluvión utilizando matrices

Nuevamente por necesidades del proyecto FONDEF IT16i10096 "Simulador basado en video juegos para respuesta a desastres" se ha requerido la experimentación en nuevas formas de representar desastres naturales, en este caso un aluvión, en donde se intentará simular el escenario ocurrido en Tierra Amarilla en donde el río se desbordo y causó un aluvión en la ciudad. Además, antes de empezar es necesario destacar que el objetivo de este experimento es mostrar una simulación visual para que el usuario comprenda el problema y pueda tomar decisiones, por lo que el realismo no es la prioridad principal.

4.5.1 Resumen del problema

Para este problema, se tiene que realizar un foco en la localidad de Tierra Amarilla, por lo que es necesario definir una matriz que cubra toda la zona de Tierra Amarilla. Luego esta misma Matriz debe tener un río, o sea, una línea de zonas con agua que representen el río. Esta matriz en algún momento debe llenarse de agua y también secarse. Por esto mismo es necesario que el río esté siempre con agua independiente de que exista un crecimiento o decrecimiento de la cantidad de agua. Entre otros puntos entonces se considera que:

- La matriz debe considerar toda la ciudad.
- Debido a la naturaleza de la matriz, el agua será plana, tal como debería ser la ciudad. A los costados de la ciudad se subirá el terreno para que la matriz no se vea cuadrada.
- debe haber una máscara aparte, con la que se pueda marcar lugares específicos en los que nunca dejará de haber agua.

- para la facilitación del trabajo de la zona de agua permanente, será necesario implementar el uso de imágenes con escala de grises para demarcar las máscaras.
- Para que el crecimiento del agua sea más realista, este será en base a probabilidades, determinadas por las celdas adyacentes a las celdas que no tienen agua.

Todos los puntos anteriores serán realizados en un script en Unity. El cual tiene la capacidad de cumplir con dichas tareas.

4.5.2 Creación de Matriz

La matriz que se generará se hará en el mundo virtual. Por lo tanto, primero se debe crear un objeto virtual que represente un charco de agua. Para esto se creará un GameObject Nuevo y vacío. Como componentes se le agregará un "MeshFilter" que representa un plano de 1x1 en el espacio 3D, y también se le agregará como componente un MeshRenderer el cual representa la textura del plano. En este caso la textura tendrá un material con transparencia, eso quiere decir que el color se verá parcialmente, en este caso, utilizando un color azul con una opacidad del 50% es suficiente. Cada plano 3D intencionalmente excederá los límites de su matriz, de tal manera que se superpongan unos con otros. Esto provocará un efecto de profundidad o condensación del agua, en donde se verá que el agua está más concentrada al centro que a los bordes. A continuación en la figura 23 se muestra gráficamente lo descrito anteriormente.

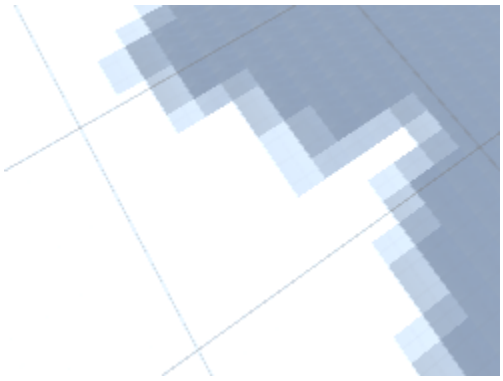

Figura 23. Matriz de “agua/barro”.

Entonces, el componente “MeshRenderer” será desactivado (no confundir con eliminado) con el fin de que el plano no sea renderizado durante la ejecución, sino más bien guardado en memoria solamente. El plano entonces es guardado en memoria para que luego al momento de iniciar el programa este plano sea copiado e instanciado las veces necesarias en la matriz. Si tenemos entonces, por ejemplo, una matriz de MxN elementos, habrán MxN instancias del plano con agua todos desactivados. Una vez realizado todo este procedimiento, la matriz se crea mediante código, y se agrega un margen de un plano a toda la matriz en la que no se puede llenar con agua, esto para evitar problemas de “out of bounds” durante se recorre la matriz (*out of bounds*: en español “fuera de los bordes” es un problema que se puede experimentar al buscar elementos fuera de los bordes de un objeto; como no “existen” objetos instanciados entonces se encontrarían objetos de tipo *null*, que podrían generar errores en la correcta ejecución del código).

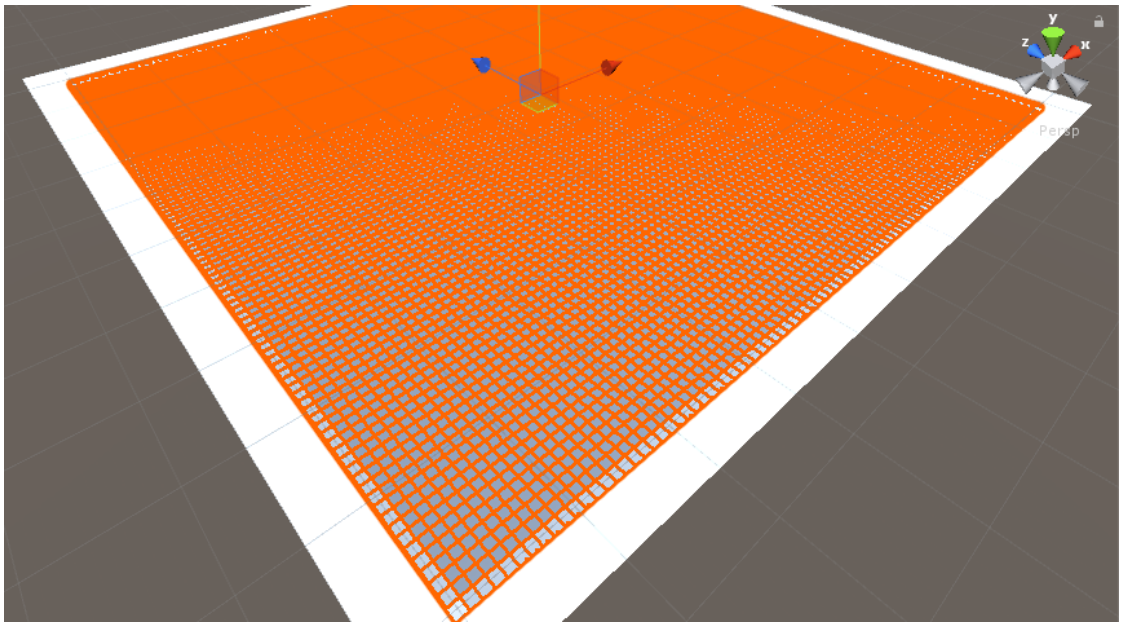

Figura 24. Matriz llena de agua.

Finalmente, en la figura 24 se puede ver la matriz llena de agua, en donde está cada uno de los elementos de la matriz siendo marcados por un contorno naranjo lo que indica por el editor Unity que son objetos.

4.5.3 Algoritmo de llenado de matriz

Como se dijo anteriormente, la matriz inicialmente existe sólo en memoria, ya que se había desactivado su etapa de renderizado a través del componente “Mesh Renderer” por lo que la técnica para ir llenando la matriz consiste en ir activando este componente en cada objeto. La forma en que se llenará entonces la matriz será a través de probabilidades. En donde, la probabilidad de que un plano desactivado se active estará determinado de un peso total, el cual será la suma del peso de sus planos vecinos. A continuación en la figura 25 se describen

los pesos de cada uno de los vecinos, y además un listado con las probabilidades para cada uno de los pesos totales.

Pesos(w)		
1	3	1
3	0	3
1	3	1

Figura 25. Pesos de vecinos.

Entonces, si la suma de los vecinos es:

- 2, las probabilidades de llenado son de 10%.
- 3, las probabilidades de llenado son de 30%.
- 4, las probabilidades de llenado son de 50%.
- 5, las probabilidades de llenado son de 75%.
- entre 6 y 8, las probabilidades de llenado son de 85%.
- 9 o más, las probabilidades de llenado son de 100%.

Estos porcentajes fueron escogidos en base a pensamiento lógico personal, por lo que no sería raro que más adelante sean ajustados dichos porcentajes, pero por el momento funcionan para lo que se requiere. Los resultados bajo este algoritmo son evidenciados a través de las figuras 26 y 27, en donde se puede apreciar como la capa de agua/barro “avanza” hacia la derecha, llenándose cuadro a cuadro dependiendo de las probabilidades descritas anteriormente.

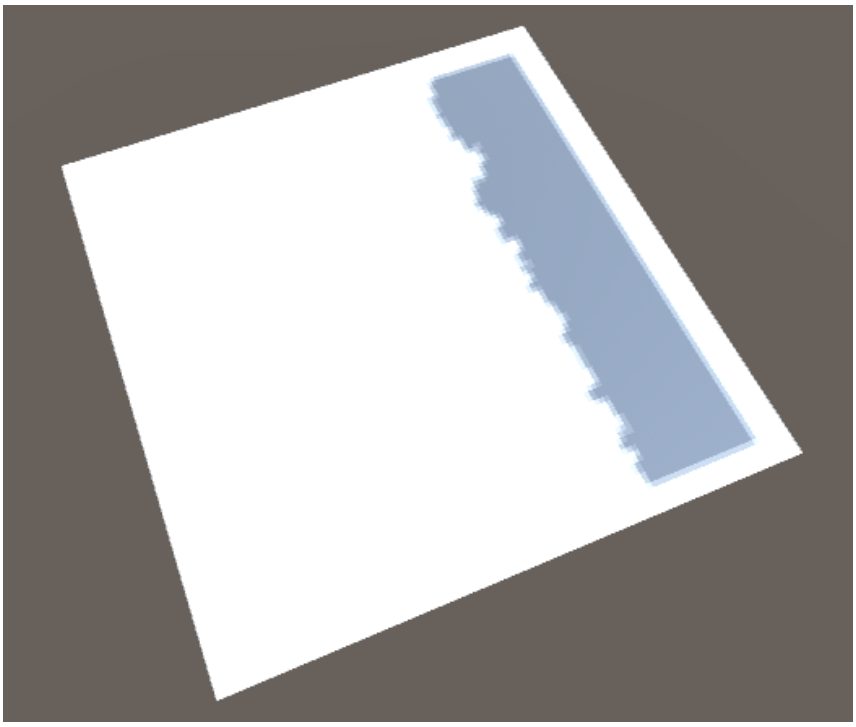

Figura 26. propagación de agua 1.

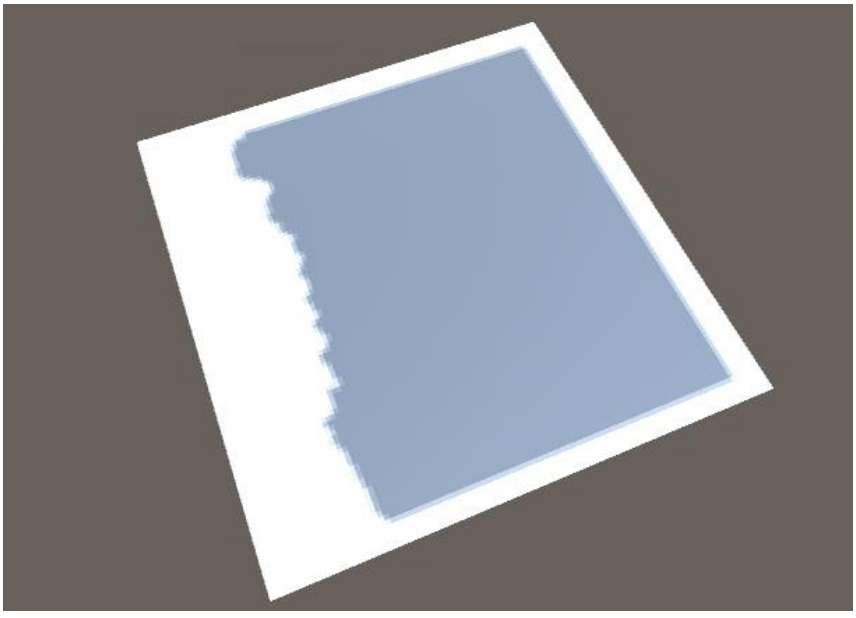

Figura 27. propagación de agua 2.

Como se puede ver, se crea un efecto de llenado de agua uniforme, lo que le da mayor realismo a la simulación. En el paso final se agregará esta matriz al mapa de tierra amarilla y además se agregará la máscara en donde siempre existirá agua. (Este algortimo puede revisarse en el Anexo 4 – Algoritmo de propagación de agua).

4.5.4 Máscara de río, Implementación en Tierra Amarilla

La idea principal de la matriz es implementarla en el territorio de Tierra Amarilla, para esto, se aplicará un ajuste al mapa de Tierra Amarilla que ha sido obtenido por el proyecto FONDEF IT16i10096 "Simulador basado en video juegos para respuesta a desastres". En donde se ajustará toda la localidad de tierra amarilla a la misma altura, para evitar errores y trabajar con la matriz más fácilmente. Luego ante la situación de ubicar la matriz en la localidad de Tierra Amarilla será necesario primero, crear una corriente de agua que representará el río de Tierra Amarilla. Para esto se utilizará una variable del tipo Texture2D la cual recibirá una imagen en escala de grises, en donde la zona negra será las partes de la matriz en que se llenará con agua y además se bloqueará esa parte para que siempre tenga agua independiente de las condiciones. A continuación en la figura 28, se muestra una imagen de cómo es la máscara.

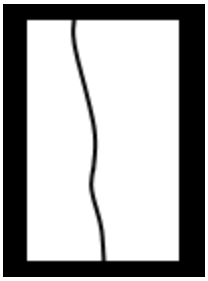

Figura 28. máscara río.

Una vez realizado este proceso, se indica en el script a través de instrucciones que la zona negra nunca dejará de tener agua independiente de las condiciones. Luego de esto, se instancia en la localidad de Tierra Amarilla y se ajusta para que esta se vea acorde al mapa. Finalmente, el algoritmo de propagación se hará en relación al origen del agua, en este caso el río. A continuación en las figuras 29, 30 y 31, se muestran imágenes evidenciando el experimento.

Figura 29. Mapa inicial con máscara de río.

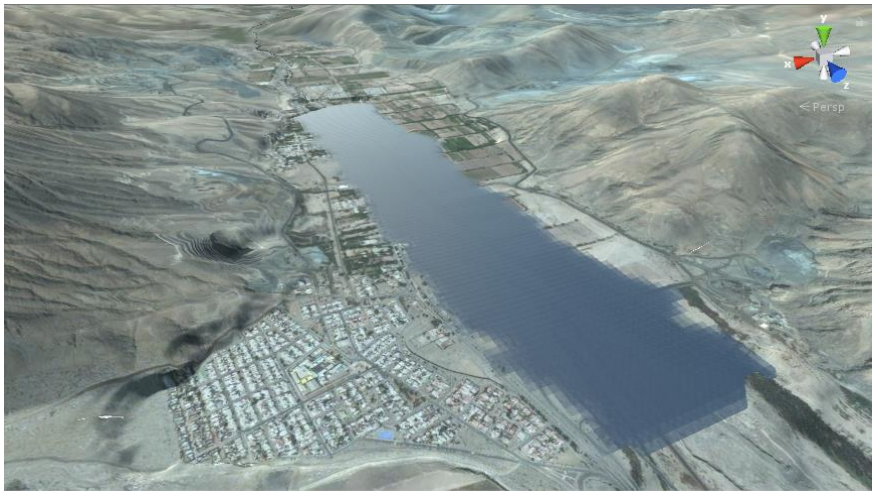

Figura 30. aluvión a través del algoritmo.

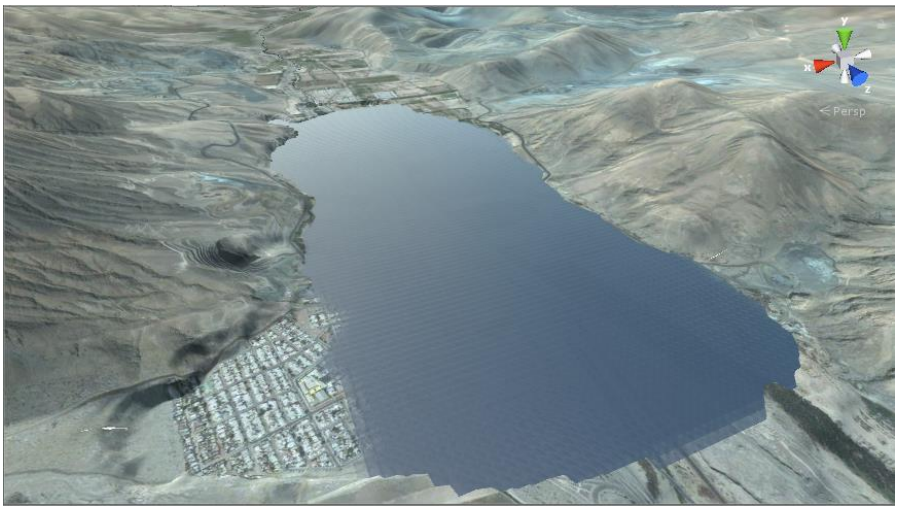

Figura 31. Aluvión más avanzado.

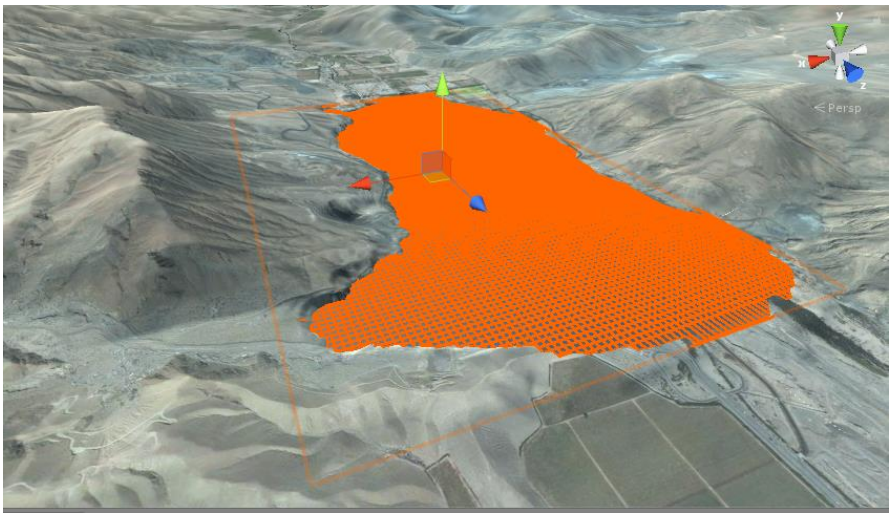

Figura 32. Matriz de texturas de agua con contorno naranjo.

Como se pudo ver en la figura 32, la matriz es un rectángulo, sin embargo, éste se ve oculto en algunas zonas, esto da la sensación de que el aluvión no es cuadrado, sino que se adapta a la zona plana y en las curvas en donde comienza a subir la altura éste no alcanza a llegar.

4.5.5 Conclusiones acerca de la propagación y posterior implementación

Para finalizar, cabe destacar que la forma en que se logró el resultado de esta sección fue en base a mucha experimentación tanto de texturas y algoritmos, en donde hubo pruebas de ensayo y error, en donde finalmente se pudo obtener un resultado aceptable en lo que concluye el experimento. Además, para relacionar este complemento de trabajo con los objetivos específicos de la tesis se agregó una funcionalidad extra en donde se realiza una comunicación entre HTML y Unity en donde a través de HTML se puede comenzar el aluvión, detener, retroceder el aluvión y además modificar la velocidad con la que este se genera, lo cual más tarde puede simular como ocurre un aluvión en horas o en un par de minutos.

Figura 33. Tierra Amarilla HTML-Unity.

Como se puede ver en la figura 33, la integración de comunicaciones entre HTML y Unity ahora permite controlar el Aluvión desde la misma página web, por lo que los objetivos del experimento han sido cumplidos.

4.6 Alternativas en la visualización de la propagación de aluvión para mejorar el rendimiento

4.6.1 Investigación previa al ejercicio

Cuando se trabaja con terrenos en Unity se tiene una serie de métodos y variables reservadas para modificar dicho terreno tanto durante su edición como en "run time" (mientras el sistema se está ejecutando). Una de las principales ventajas del terreno es tener múltiples texturas en un solo terreno ya que éstas existen internamente en memoria y son parte del terreno, o sea que no significa un gran uso de recursos al momento en que estas requieren ser utilizadas ya que las texturas están "precargadas" en memoria. La forma en cómo operan estas texturas es mediante una matriz activa de 128X128 en el terreno. Esta podría considerarse como la resolución del terreno, eso si no se debería confundir que el terreno solo soporte una imagen de 128x128 como textura de suelo, sino más bien que el tamaño más pequeño posible en que se puede cambiar parte de la textura del terreno es de 1/128. Por cada punto en la matriz, las texturas tienen un nivel que va desde 0.0 a 1.0; como bien se sabe, cada textura tiene 3 canales de colores, los cuales son usualmente representados como canales RGB (RED, GREEN, BLUE). Entonces el algoritmo para calcular el color final del punto en la matriz es sumando cada textura la cual a su vez es multiplicada por el escalar mencionado anteriormente que va desde 0.0 a 1.0. Una forma de representarlo es la siguiente:

$$Textura\ Final\ =\ text1.RGB\ *\ escalar1\ +\ text2.RGB\ *\ escalar2$$

Frente a esto pueden ocurrir casos importantes que podrían afectar la visualización correcta de terreno. Como se mostró en la ecuación anterior, si uno deja como escalar 1.0 a ambas texturas lo que ocurrirá será que se obtendrá un terreno de un color mucho más blanco o

brillante de lo normal, ya que los valores al son sumados simplemente en lugar de obtener un promedio de ambos, obteniendo así valores RGB más altos de los posiblemente deseados. Este sistema es interno de Unity por lo tanto no se puede manipular, sin embargo, lo mejor que se puede hacer es manipularlo manualmente a través de los escalares. En este caso la técnica utilizada es el recorrido de la matriz y activar y desactivar texturas alternando entre escalares de 0.0f o 1.0f. Otra de las variables y métodos accesibles de terreno en Unity es la matriz de altura de terrenos, esta se denomina usualmente como "HeightMap". Esta matriz es de resolución 1025x1025 la cual es mucho mayor que la anterior matriz de texturas. Esto es debido a que se necesita una mayor precisión/resolución al momento de mostrar terrenos en ya que generalmente no son completamente planos, sino que bastante variables.

4.6.2 Implementación en base a la investigación anterior

En base a lo anterior, se procede a programar los algoritmos necesarios, re-implementando la misma forma de propagación del experimento anterior, esta vez la gran ventaja es tener una mejora en el rendimiento y, además, el tener un alcance de todo el mapa. Este factor es importante ya que puede hacer que las situaciones que se generen a partir del aluvión sean más dinámicas en lugar de estáticas, de esta forma el usuario final no siempre se enfrentará a una misma situación, sino que se presentará distintas situaciones a las que tendrá que enfrentarse posiblemente en cualquier parte del mapa y no sólo en la zona central donde está el poblado de Tierra Amarilla. Un punto importante es la transformación de puntos de Heightmap a TextureMap, dicho de otra forma, realizar el cálculo para que cada 1 punto de textura cambiada se cambie (1025/128) x (1025/128) puntos de altura de suelo. El propósito de cambiar la altura del suelo es simular que hay un mayor volumen en las zonas que tienen

agua, levantando el terreno un poco más que el resto da la sensación de que hay una capa de agua encima, en lugar de solo cambiar la textura resultando en un efecto visual ligeramente más realista.

Los resultados son los siguientes:

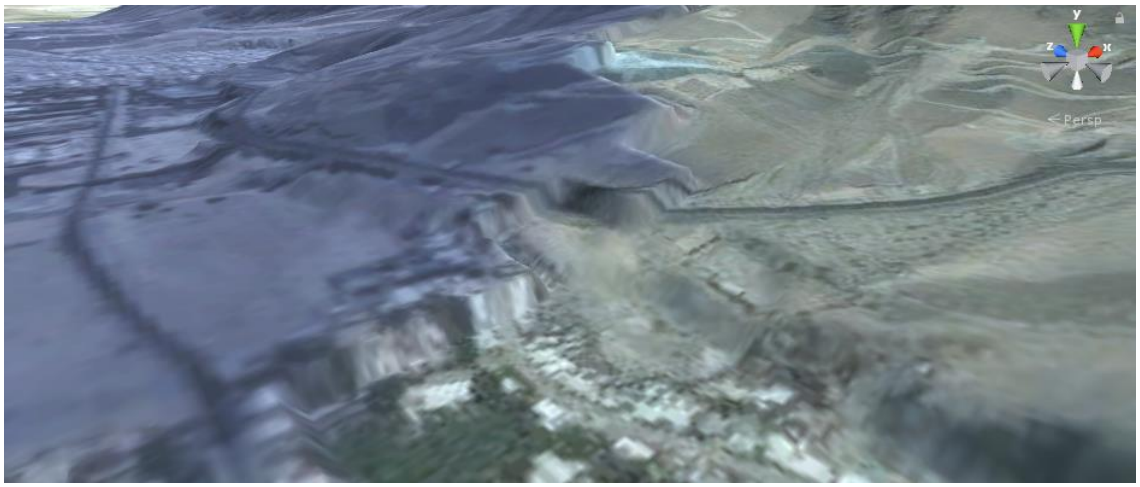

Figura 34. Tierra Amarilla propagación con Mapa de Texturas 1.

Se puede ver la “ola” de agua sobre el poblado (en este caso experimental es “agua” cuando en un aluvión debería verse un color más como barro que arrasa con el poblado)

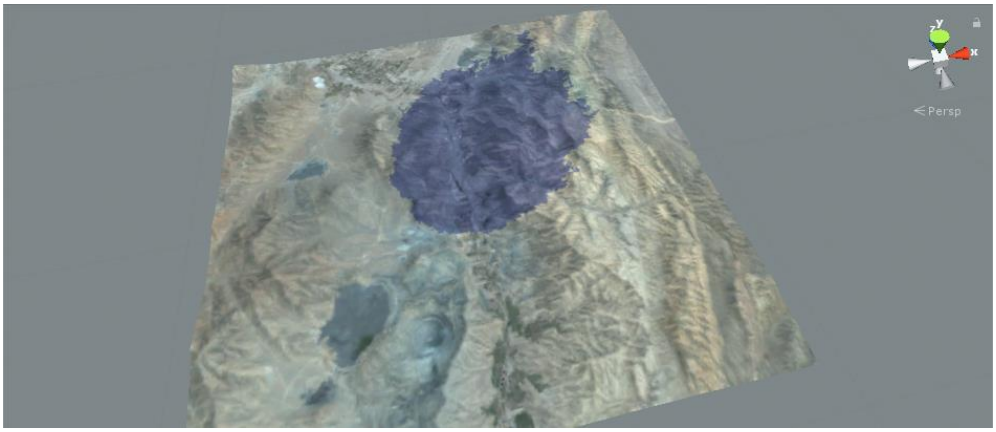

Figura 35. Tierra Amarilla propagación con Mapa de Texturas 2.

Como se puede ver en las figuras 34 y 35 se demuestra que tiene un alcance total del mapa, en lugar de solo la parte central como el experimento anterior, además si se logra ver de cerca se puede ver el relieve generado por la capa de agua.

4.6.3 Conclusiones del experimento

Finalmente, este ejercicio es una buena alternativa para la propagación del aluvión, dando más libertad que el ejercicio anterior, además tiene más sentido para el proyecto ya que este se basa en el envío de matrices a través de comunicación HTML-UNITY. La diferencia es que antes solo se manipulaba el tiempo en que tardaba en propagarse el aluvión generando objetos 3d que representaban el agua, y ahora se modifica el suelo en cualquier parte del mapa tanto en textura y alturas. Además, para evitar problemas se agregó una variable que guarda y cargar los valores iniciales del mapa para evitar que estos sean modificados de forma permanente después de su uso. Los beneficios finales del ejercicio es un impacto directo en el rendimiento ya que evita la creación de nuevos objetos 3D lo cual hacía que el sistema se hiciera muy lento.

4.7 Uso de Partículas. Desarrollo de sistema de partículas en Unity e Instanciación de partículas a través de comunicación HTML-Unity

4.7.1 Creación de un Sistema de Partículas, Análisis de su uso

En Unity los sistemas de partículas dan la capacidad de crear efectos sin la utilización de polígonos. La idea es principalmente eso, que un entorno 3D no este hecho en base a solo polígonos, ya que el impacto visual no es tan grande. La gran particularidad del sistema de partículas es que está hecho para representar objetos como humo, fugas de agua, fuego, etc. Dichos elementos no tendrían mucho sentido al ser trabajados como una serie de polígonos, ya que los haría poco realistas, y ese es el propósito del uso de sistema de partículas.

En este experimento se utilizará entonces un sistema de partículas para representar el fuego y el humo del incendio. El sistema de partícula básicamente consta de una textura que se repite múltiples veces simultáneamente para representar algo; en este caso se utilizará una especie de esfera con bordes opacados, la cual al instanciarse múltiples veces dará la representación de una nube de humo.

Unity cuenta con su propio sistema de partículas modificable por el usuario para que él pueda representar lo que guste. A continuación, se describirá de forma breve las propiedades que se consideraron más importantes del sistema de partículas para la creación del humo y fuego del escenario.

- **startLifetime** es el tiempo que la partícula existe en el mundo 3D.
- **startSpeed** velocidad de la partícula (la velocidad de movimiento del fuego es rápida, mientras que la del humo es más lenta).

- **gravity Modifier** es ideal para representar el peso de la partícula, por ejemplo, si se le añade mucho peso y la partícula sale disparada en dirección vertical hacía arriba esta caerá en un punto, esto sería ideal como para representar el chorro de agua en una pileta. En este caso, se le aplica poca gravedad para que el humo se desplace hacía arriba.

- **Emission** es la cantidad de partículas que son lanzadas simultáneamente, reducir este valor puede aumentar el rendimiento en caso de que haya muchas partículas en el escenario, aunque la sobre reducción puede conducir a efectos visuales no deseados.

- **Shape** sirve para configurar la "caja" que contiene a las partículas, ideal para manejar las restricciones de espacio de las partículas.

- **velocity over lifetime** que significa velocidad con el tiempo. a través de esta se puede por ejemplo reducir la velocidad de las partículas de fuego al final de su ciclo, para que exista una transición más coherente entre el fuego y el humo

- **color over lifetime** significa color con el tiempo. Puede servir igualmente para hacer la transición de color fuego a humo.

- **size over lifetime** regula el tamaño con el tiempo. Ideal para el humo, ya que este aumenta su tamaño con el tiempo.

- **External forces** esta opción es ideal para trabajar con cajas de viento en Unity. Significa que si uno agrega una caja de viento en Unity para representar que hay viento corriendo en una dirección el sistema de partículas será afectado por dicho viento y se moverá en paralelo a su dirección, lo que le da más realismo al sistema.

en base a los parámetros anteriores se creó un sistema de partículas para el fuego y otro para el humo el cual se incluye como un sistema anidado al de fuego. El resultado se puede apreciar en la figura 36 a continuación.

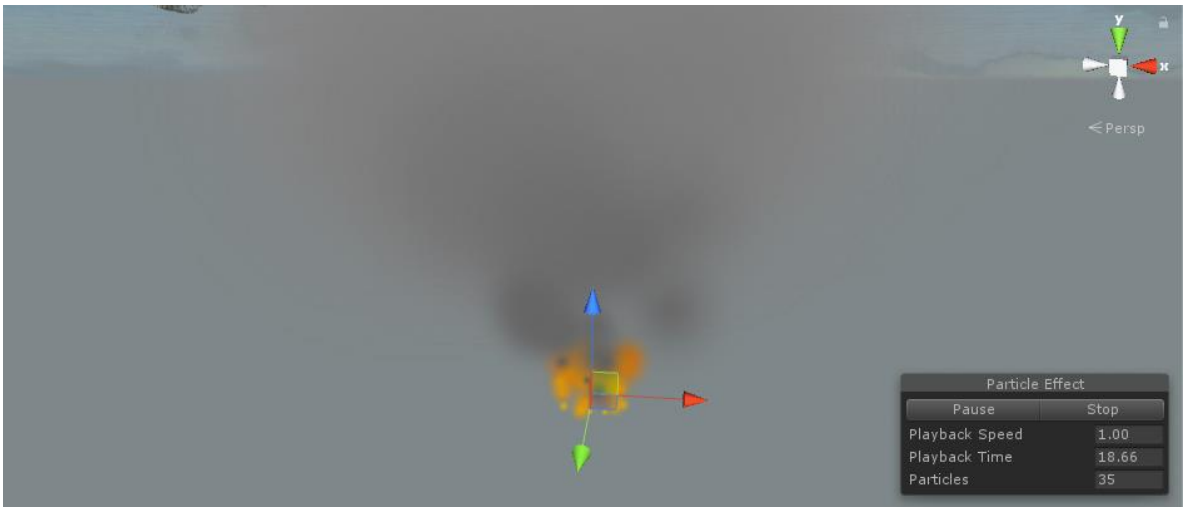

Figura 36. Sistema de partículas Fuego-Humo

4.7.2 Instanciación de partículas utilizando HTML

Como se demostró en fases preliminares del proyecto, es posible instanciar objetos a través de HTML en base a ordenes enviadas desde HTML a Unity. Como ya está listo el sistema de partículas y se tiene una matriz hecha para el incendio en Santa Olga en este experimento final se enviará una matriz utilizando HTML para ser recibida por Unity y luego se mostrarán focos de incendios dentro del terreno en Unity. Para esto sólo se debe desarrollar el formato en que se recibirá la data. Se utilizará el mismo método del Aluvión en donde se utilizará una textura secundaria para pintar el suelo como si se estuviera quemando y en base a esos lugares en el suelo se instanciará una nueva partícula de humo y fuego para el mapa el cual actuara

como “foco de incendio”. Dado que cada terreno tiene una resolución interna de 128x128 para el mínimo de cambio de textura, significa que teniendo 4 terrenos entonces se tiene un espacio de 256x256 para llenar con instancias. Ahora para tener en consideración, en el peor de los casos en que todo el mapa se esté quemando, se instanciaría 65.536 veces el sistema de partículas, resultando en un consumo de memoria grande que incluso forzaría a un sobrecargo de memoria y problema del sistema. Por lo que se decidió limitar la matriz con un algoritmo bastante simple. Lo que hace este algoritmo es que el usuario (a nivel de desarrollador) determina una cantidad n máxima de partículas, en este caso digamos n=1000 sistemas de partículas de forma simultánea (que no generan una sobrecarga del sistema), luego se calcula el valor entero de la raíz cuadrada de n el cual resulta en 31. Luego se calcula el valor de salto dividiendo la resolución total (256) por la generada (31) tomando nuevamente el valor entero, el cual resulta ser 4. Significa que en la matriz real de 256x256 se realizarán saltos de 4x4 para instanciar las partículas de humo, de esta forma el total no excederá el máximo de partículas de humo que se ha descrito con anterioridad. Además, antes de realizar el proceso de instanciación se evalúa si las zonas de fuego exceden el número determinado por usuario (n=1000) y sólo si lo exceden ejecutarán el algoritmo. ya que el algoritmo está hecho, solo falta determinar cómo recibirá la data Unity. Para esto se decidió utilizar un formato de texto simple de la forma “xxxyyyz” significa que xxx representa el eje x desde 000 a 256, yyy representa el eje y desde 000 hasta 256 y z representa el estado, el cual puede ser normal, o quemándose. Este formato se repite 65.536 veces en un archivo de texto para describir a toda la matriz. esto resulta en un archivo de un peso alrededor de 486 kilobytes. El cual es más que suficiente para ser utilizado como prueba en el sistema.

Luego en Unity se crea una variable del tipo WWW que es el tipo de variables que se usa cuando se requiere descargar contenido desde internet. Entonces, el archivo de texto descrito con anterioridad será guardado en una página de internet (en este caso se utilizó el servidor pillan de la escuela de ingeniería en informática destinado al uso por alumnos) y a Unity se le dará la instrucción a través del estándar de comunicación desarrollado en este mismo proyecto. La instrucción enviada será “sendMatrix” y como parámetro se agrega la URL perteneciente al archivo de texto. Para este ejercicio se creará un botón que automáticamente ejecutará esa acción en el script de HTML e internamente, Unity realizará la llamada a través de la variable WWW para extraer el texto, “parsear” y luego instanciar las partículas en el escenario. A continuación, se muestran los resultados en las figuras 37 y 38:

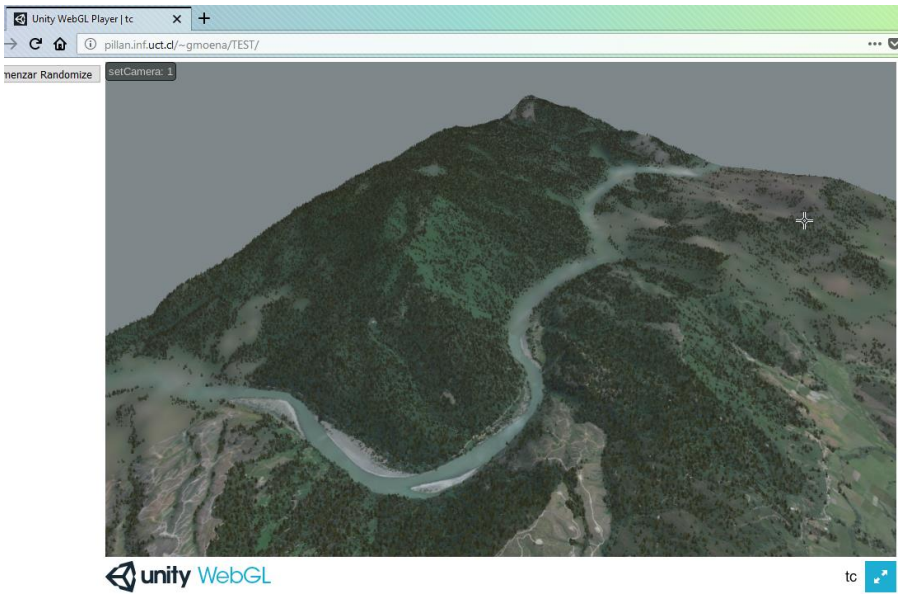

Figura 37. Mapa completo antes de realizar instancia de fuego

Figura 38. Fuego instanciado en el Mapa

Todas esas zonas de fuego han sido obtenidas desde un archivo localizado en un servidor de internet al cual Unity ha sido capaz de entrar a través del tipo de variables WWW.

Es posible revisar parte del trabajo realizado en el Anexo 5 – Algoritmo de Instanciación de partículas (Extracto).

5. CONCLUSIÓN FINAL

Finalmente, el experimento permitió mostrar la potencia de la comunicación entre Unity y HTML, y como ambos sistemas pueden complementarse para el desarrollo de un videojuego serio y complejo con fines educativos. Lo que se realizó en este último experimento es muy similar a lo que sería la forma final que se implementaría en el proyecto FONDEF IT16I10096 en donde se utilizará un sistema desarrollado por otros alumnos en la plataforma web que configuraría o prepararía los datos de la matriz para ser enviados a este proyecto en Unity, en donde ejecutará todo lo referente a las partículas y el incendio forestal. Como observación personal, creo que, desarrollando un sistema desde su raíz utilizando un terreno creado desde cero sin dar una visualización tan compleja podría servir también para la simulación de terrenos de catástrofes, ya que se podría enseñar de alguna manera mejor el concepto. Aun así, eso desviaría completamente el foco del proyecto principal, ya que el sistema de Unity no es el foco principal del proyecto, sino más bien un complemento a las situaciones que se presentarán a lo largo del proyecto web final.

6. BIBLIOGRAFÍA

- 24Horas Tvn. (2017). Al menos cuatro desparecidos dejan aluviones en la zona central. 14-08-2017, de 24horas.cl Sitio web: http://www.24horas.cl/nacional/al-menos-cuatro-desaparecidos-dejan-aluviones-en-la-zona-central-2312371
- Aguevedo, E. (2010). Chile: terremoto del 27 de Febrero de 2010. 14-08-2017, de Aquevedo wordpress Sitio web: https://aquevedo.wordpress.com/2010/03/08/chile-terremoto-del-27-de-febrero-de-2010/
- Andee. (2009). Do you use Unity for simulation?. 14-08-2017, de Unity Forums Sitio web: https://forum.unity3d.com/threads/do-you-use-unity-for-simulation.30735/
- Andrioti, Z. (2015) "Web 3D gaming over HTML5 and web-based communication"(Master's thesis, Technological Educational Institute of Crete, Greece).
- Bennis, W.G., & Thomas, R.J. (2002). Geeks & geezers: How era, values, and defining moments shape leaders. Harvard Business School Press, Boston, MA.
- Blazona, B., and Mihajlovic, Z. (2007). Visualization service based on web services. Journal of Computing and Information Technology 15, 4, 339.
- CEININA. (2016). Adjuficación proyecto IT16I10096. 20-08-2017, de Centro de Innovación de Investigación Aplicada Sitio web: http://ceinina.cl/?q=node/290
- Computer Simulation. (Sin fecha). En Wikipedia. Recuperado el 14 de Octubre de 2017 de https://en.wikipedia.org/wiki/Computer_simulation
- De La Rosa, I. (2017). Recrearon las condiciones de la tragedia de Tur Bus en un simulador y el resultado fue el mismo. 12-08-2017, de LosAndes Sitio web:

http://www.losandes.com.ar/article/al-menos-15-muertos-tras-el-vuelco-de-un-micro-en-horcones

- Desarrollo en Espiral. (Sin fecha). En Wikipedia. Recuperado el 05 de 11 de 2017 de https://es.wikipedia.org/wiki/Desarrollo_en_espiral
- Domínguez, F. (2017). Santa Olga, la historia del pueblo de la comuna de Constitución que fue destruido por las llamas. 14-08-2017, de Emol Sitio web: http://www.emol.com/noticias/Nacional/2017/01/26/841943/Santa-Olga-la-localidad-forestal-que-desaparecio-con-el-incendio.html
- Fette, I. Google Inc. Melnikov, A. Isode Ltd. (2011). The WebSocket Protocol. 2017, de Internet Engineering Task Force Sitio web: https://tools.ietf.org/html/rfc6455
- Gao, J. Huth, A. Lescroart, M. & Gallant, J. (2015). "Pycortex: an interactive surace visualizer for fMRI" Front Neuroinform 2015. Pp 9-23, September 2015.
- Grijalva, N. (2017). Modelo Espiral. 20-08-2017, de Nathaly Grijalva Sitio web: http://sofware1nathalygrijalva.blogspot.cl/2012/10/modelo-espiral.html
- Lavoué, G. Chevalier, L. & Dupont, F.(2013) "Streaming Compressed 3D Data on the Web using JavaScript and WebGL" ACM. International Conference on 3D WebTechnology(Web3D), Spain.
- Lopez, A. (2017). ¿Por qué están difícil combatir incendios forestales?. 28 de Noviembre de 2017. Sitio web: https://www.eldefinido.cl/actualidad/pais/8065/Por-que-es-tan-dificil-combatir-incendios-forestales/
- Luttecke, C. (2014). Sabes que es unity? Descubrelo aqui. 14-08-2017, de Deideaapp Sitio web: https://deideaaapp.org/sabes-que-es-unity-descubrelo-aqui/
- Moreno, M. (2014) "Modelo de Desarrollo de escenarios para la simulación de incendios forestales mediante tecnología de videojuegos serios", 2014.

- Oak, J. & Bae, J. (2013). “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.
- ONEMI. (2017). “Aluviones”, de ONEMI Sitio web: http://www.onemi.cl/aluviones/
- Ortiz, M. (2017). Alerta roja por incendio forestal descontrolado que quemó una casa y avanza en Quillón. 14-08-2017, de BioBioChile Sitio web: http://www.biobiochile.cl/noticias/nacional/region-del-bio-bio/2017/02/22/incendio-forestal-se-reactiva-afectando-a-sector-el-arenal-en-la-comuna-de-quillon.shtml
- Raybour, E. Heneghan, J. (2005). Adaptive Thinking & Leadership Simulation Game Training For Special Forces Officers. 2017, de I/ITSEC Sitio web: http://www.sandia.gov/adaptive-training-systems/Raybourn%20et.%20al.%20ITSEC%202370b.pdf
- Raybourn, E. M. (2001) Designing an emergent culture of negotiation in collaborative virtual communities: The DomeCityMOO Simulation. In E. Churchill, D.Snowden, & A. Munro (Eds.) Collaborative Virtual Environments: Digital Places and Spaces for Interaction, Springer, London UK, 247-64.
- Raybourn, E. M. (2003) Design cycle usability and evaluations of an intercultural virtual simulation game for collaborative virtual learning. In C. Ghaoui(Ed.), Usability Evaluation of Online Learning Programs, Information Science Publishing, 233 -53.
- Raybourn, E. M., & Waern, A. (2004) Social learning through gaming. In Extended Abstracts of CHI Proceedings 2004, ACM Press, 1733.
- Recatalà, O. (2017). “Real world to 3D terrain in Unity” Edited by Universitat Jaume I, 2017.

- SputnikNews. (2017). Un terremo te magnitud 5,7 se registró en las regiones de Atacama y Coquimbo Chile. 14-08-2017, de mundo.Sputniknews Sitio web: https://mundo.sputniknews.com/americalatina/201708121071523599-america-latina-sismo/
- T13. (2017). 38 incendios fueron controlados y solo 3 estan en combate. 14-08-2107, de t13 Sitio web: http://www.t13.cl/noticia/nacional/minuto-a-minuto/Sigue-el-minuto-a-minuto-de-los-incendios-forestales-en-Chile
- Thorn, A. (2016). How to Cheat in Unity 5: Tips and Tricks for Game Development. Google Books: Focal Press.
- Wang, S. Mao, Z. Zeng, C. Gong, H. Li, S. Chen, B. (2010). A new method of virtual reality based on Unity3D. 2017, de IEEE Sitio web: http://ieeexplore.ieee.org/abstract/document/5567608/
- Wong, L. (2004). Developing adaptive leaders: the crucible experience of Operation Iraqi Freedom. Strategic Studies Institute, Carlisle Barracks, PA.

ANEXOS

Anexo 1 – Script de Comunicación UNITY-HTML por el lado de Unity utilizando lenguaje C#.

UnityApi.cs

```
public class UnityAPI : MonoBehaviour {


    // Sugerencia de implementacion:
    // seria bueno crear una clase gameobject vacio en unity y dar nombre
    // significativo, como "UnityCOM", luego en el html-js cada vez que
    // se quiera ejecutar una funcion del tipo sendmessage la sintaxis seria:
    // gameInstance.SendMessage("UnityCOM", "UnityReceiver", msgToUnity);
    // se propone lo siguiente:
    // En Unity: UnitySender(), UnityReceiver()
    // En HTML5: WebSender(), WebReceiver()

    // clase estandar para la comunicacion entre plugin de unity y html-js
    // este es la puerta de entrada salida al sistema unity

    // sintaxis:
    // "metodo: parametro1, parametro2, ..., parametron"

    /* UnitySender recibe como parametro un string que sera el mensaje
     que enviara a javascript a traves de externalcall
     el primer parametro es obligatorio, este sera la el nombre del metodo,
     los parametros seguidos a este son opcionales y representan a los
     parametros de la funcion llamada en el primer parametro.
```

```
 Los parametros opcionales pueden venir en cualquier tipo de variable,
 pero se debe asumir que esta luego sera transformada automaticamente a
 tipo string ya que seran concatenadas en una sola variable tipo string
 que sera la que se enviara mas tarde.

*/
public static void UnitySender(string data, params object[] values)
{
    data += ":";
    for(int i=0; i < values.Length; i++)
    {
        data += values[i].ToString();
        // si i no es el ultimo del array
        if(i+1 < values.Length)
        {
            data +=",";
        }
    }
    // la variable data ha sido procesada, ahora se envia a html.js
    Application.ExternalCall("WebReceiver", data);
}

/* UnityReceiver recibe como parametro un string que sera enviado
por javascript

data.Split separara un parametro de la forma mostrada anteriormente
a un array de la forma ["metodo", "parametro1, parametro2, ..., parametron"]
```

```
    */
    public void UnityReceiver(string data)
    {
        string[] dContent = data.Split(':');

        switch (dContent[0])
        {
            case "moverAuto":
                //mainScript ms = GetComponent<mainScript>();
                //ms.SendMessage("recieveData", dContent[1]);
                break;

            case "changePosition":
                UnityGlobals.changePosition(dContent[1]);
                break;

        }
    }

}
```

Anexo 2 – Scripts de Funcionalidades complementarias en la comunicación por parte de Unity.

UnityGlobals.cs

```
public class UnityGlobals : MonoBehaviour {

    public static float x1 = 0;
```

```
public static float y1 = 0;
public static float x2 = 7970;
public static float y2 = 7970;
        // Use this for initialization



// updateMiniMap:
// este metodo envia la informacion a js-html utilizando UnityAPI
// recibe la informacion, luego utiliza el metodo de normalizacion y
// finalmente envia los datos.
public static void updateMiniMap(float x, float y)
{
    Vector3 pos = normalizeData(x, y);
    UnityAPI.UnitySender("UpdateFromUnity", pos.x, pos.y );

}



// normalizeData:
// este metodo utiliza los bordes del mapa para transformar valores
// dentro de estos bordes a numeros entre 1 y 0.
// se utiliza la misma tecnica de normalizacion usada en otros ejemplos
// del modo n = (n - min) / (max - min). Como en este caso el min es 0
// entonces este resulta en data1 = a / b .
// luego se retorna el valor normalizado utilizando una variable de tipo
// vector3 en la cual se utilizan los primeros dos valores y el otro
// es rellenado con 0.
public static Vector3 normalizeData(float a, float b)
```

```
{
    float data1 = a / x2;
    float data2 = b / y2;

    Vector3 data = new Vector3(data1, data2, 0.0f);
    return data;
}


// changePosition:
// esta clase deberia recibe un string con siguiente formato
// "nombreobjeto, posicionx, posiciony"
// esta info viene desde google maps, por lo tanto hay que ajustar
// el formato a Unity en terminos del vector de posicion
// por tanto aqui se aplicara una transformacion en eje x, z
// utilizando un vector en el cual la inforamcion se distribuira entre
// x,z de la forma: Vector3(posx, 0, posY)

public static void changePosition(string data)
{
    string[] dContent = data.Split(',');

    string objName = dContent[0];
    float posX = float.Parse(dContent[1]);
    float posZ = float.Parse(dContent[2]);

    // normalizacion a mapa Unity
    posX *= x2;
```

```
        posZ *= y2;

        GameObject obj = GameObject.Find(objName);
        float posY = obj.transform.position.y;

        obj.transform.position = new Vector3(posX, posY, posZ);

    }
}
```

Anexo 3 - Comunicación HTML-JS por el lado de HTML utilizando Javascript

```
// ------------------COMUNICACION JS - UNITY ----------------------
// funcion que se propone como funcion final para recibir datos
// desde Unity en Js.
// funciona de manera similar a la que existe en Unity.
// se recibe informacion y esta es segmentada para ser enviada
// a la funcion correspondiente a traves de un switch

function WebReceiver(args)
{
  var data = args.split(":");
  //if(data[0] == "UpdateFromUnity")
  //{
  //  updateFromUnity(data[1]);
  //}

  switch (data[0])
  {
```

```
    case "UpdateFromUnity":
      updateFromUnity(data[1]);
  }

}



// WebSender
// segun la definicion de argumentos variables de JS dice lo sgte:
// es posible definir una cantidad variable de argumentos siempre y
// cuando sean llamados a traves de la palabra reservada arguments
// function foo(){
//   for (var i=0; i < arguments.length, i++){
//     //do something argument[i];
//   }
// }

// la zona de if if for if finalmente realiza lo sgte:
// si hay argumentos relizar envio.
// si hay mas de un argumento preparar la cadena para mas argumentos.
// preparar la cadena concatenando a traves de un for mediante ','
// que separan cada valor.
// como se agrega la ',' al final de cada valor, si este es el ultimo
// valor de la lista entonces no se agrega nada.
// finalmente esto es enviado de la forma 'metodo: param1, param2, paramx'
function WebSender()
{
  // si hay argumentos
```

```
  if(arguments.length > 0)
  {
   var msgToUnity = arguments[0]+":";
   // si hay mas de un argumento
   if(arguments.length > 1)
   {

    // empieza de 1 ya que 0 ya se incluyo
    for(var i = 1; i < arguments.length; i++)
    {
     msgToUnity += arguments[i].toString();
     // si i no es el valor final del array
     if(i+1 < arguments.length)
     {
      msgToUnity += ",";
     }
    }
   }
   console.log(msgToUnity);
   gameInstance.SendMessage("UnityCOM", "UnityReceiver", msgToUnity);
  }
}

// ----------FIN COMUNICACION JS - UNITY ----------
```

Anexo 4 - Algoritmo de propagación de agua (extracto)

```
//-- Importante: algoritmo de propagación de agua en tierra Amarilla---
//-- se asume que se tiene ya una matriz, por lo que se mostrará
```

```
//-- solo el algoritmo de llenado de matriz
    void waterPropagation()
  {
      // sow representara la suma de pesos

      int sow;

      // se almacenan los datos en una matriz temporal, de este modo
      // se evita el problema de que se llene mas hacia un lado que al otro
      // debido a posibles llenados mientras aun se leen nuevos datos.

      int[][] oldWaterData = new int[waterData.Length][];
      for(int y=0; y < waterData.Length; y++)
      {
         oldWaterData[y] = new int[waterData[y].Length];
         for(int x=0; x < waterData[y].Length; x++)
         {
            oldWaterData[y][x] =
               bti(waterData[y][x].GetComponent<basics>().getRend());
         }
      }

      // mientras mas peso tengan los vecinos de una celda vacia
      // mas probabilidad de que esta se llene
      // los vecinos adyacentes tiene peso 3, los diagonales peso 1
      for(int y=0; y < col; y++)
      {
         for(int x=0; x < row; x++)
```

```
{
    // accedo al script del "tile" actual para ver si esta siendo
    // renderizado o no.
    basics s = waterData[y][x].GetComponent<basics>();

    // checkear si no esta renderizado
    if(!s.getRend())
    {
        // si no esta renderizado entonces ejecutar codigo
        sow = 0; // se empieza con 0

        // para realizar la suma se calcula el peso de los vecinos
        // * se deja un margen de 1 tile a la matriz, esto debido a
        // que para manejar los bordes de la matriz se necesitan 8
        // casos puntuales en los que se debe calcular de forma
        // diferente para no salirse de los limites
        // de la matriz. Por lo que esta metodologia es mas simple.
        if((x > 0 && x + 1 < row) && (y > 0 && y + 1  < col))
        {
            // el numero final son los pesos
            sow += oldWaterData[y+1][x-1] * 1;
            sow += oldWaterData[y+1][x] * 3;
            sow += oldWaterData[y+1][x+1] * 1;
            sow += oldWaterData[y][x-1] * 3;
            sow += oldWaterData[y][x+1] * 3;
            sow += oldWaterData[y-1][x-1] * 1;
            sow += oldWaterData[y-1][x] * 3;
            sow += oldWaterData[y-1][x+1] * 1;
```

```
            }

            // Finalmente se genera la propagacion, esta propagacion es
            // simplemente una probabilidad en que se saca un numero al
            // azar de una distribucion uniforme entre 0 y 11, y si esta
            // es menor al peso entonces se muestra el tile
            // ej: sow = 3 -> probabilidad de random
            // menor a 3 = 3/11 = 27.27%
            if(Random.Range(0, 11) < sow)
            {
               waterData[y][x].GetComponent<basics>().HideObject();
            }

         }
      }
   }
}
```

Anexo 5 - Algoritmo de Instanciación de partículas(Extracto)

```
// proceso de generación de una partícula
void InstantiateParticle(int y, int x)
{
   // si la partícula sobre la que está el
   if(textureMap[y, x, 1] > 0.5f && particleCounter < maxParticles)
   {
      // copiar el objeto escondido del escenario
      GameObject temp = Instantiate(GameObject.Find("ForestFire"));
```

```
        // se agrega un offset para corregir posición
        int offset = 0;
        Vector3 pos = new Vector3(-7970 + x * 62 + offset, 1000f, -7970 + y * 62f +
offset);



        // se asigna la posición con un eje y más arriba de lo normal
        temp.transform.localPosition = pos;

        // luego que el objeto ha sido posicionado más arriba del nivel normal
        // se utiliza un Raycast,
        // lo que esto hace en pocas palabras es "lanzar" un rayo hacia abajo
        // que busca una colisión y al encontrarla se obtiene información de esa colisión
        // en este caso al encontrar el suelo se realiza un calculo de distancia y se ajusta la
partícula
        // para que esta esté a la altura del suelo
        RaycastHit hit;
        if(Physics.Raycast(temp.transform.position, -Vector3.up, out hit))
        {
           pos.y -= hit.distance;
        }

        // se asigna la posición corregida.
        temp.transform.localPosition = pos;
        // asignar nombre / identificador
        temp.name = "smoke_"+x+"_"+y;

        // agruparlo en el directorio Particles
        temp.transform.SetParent(GameObject.Find("Particles").transform);
```

```
    temp.transform.localScale = new Vector3(3f, 3f, 3f);

    // registrar partícula
    particleCounter += 1;
}
```