Universidad Central de Venezuela
Facultad de Ciencias
Escuela de Computación

Centro de Computación Gráfica

Generación procedimental de contenido en un videojuego del género MOBA

Trabajo Especial de Grado
presentado ante la Ilustre
Universidad Central de Venezuela
Por el Bachiller
Alejandro Cannizzo S.
para optar al título de
Licenciado en Computación

Tutor: Prof. Esmitt Ramírez

Caracas, 18 de Octubre de 2013

Universidad Central de Venezuela
Facultad de Ciencias
Escuela de Computación
Centro de Computación Gráfica

ACTA DEL VEREDICTO

Quienes suscriben, Miembros del Jurado designado por el Consejo de la Escuela de Computación para examinar el Trabajo Especial de Grado, presentado por el Bachiller Alejandro Cannizzo Stacconi C.I.: 20.219.975, con el título *Generación procedimental de contenido en un videojuego del género MOBA*, a los fines de cumplir con el requisito legal para optar al título de Licenciado en Computación, dejan constancia de lo siguiente:

Leído el trabajo por cada uno de los Miembros del Jurado, se fijó el día 28 de Octubre de 2013, a las 9:00 am, para que su autor lo defendiera en forma pública, en el Centro de Computación Gráfica, lo cual se realizó mediante una exposición oral de su contenido, y luego respondió satisfactoriamente a las preguntas que les fueron formuladas por el Jurado, todo ello conforme a lo dispuesto en la Ley de Universidades y demás normativas vigentes de la Universidad Central de Venezuela. Finalizada la defensa pública del Trabajo Especial de Grado, el jurado decidió aprobarlo.

En fe de lo cual se levanta la presente acta, en Caracas a los 28 días del mes de Octubre del año dos mil trece, dejándose también constancia de que actuó como Coordinador del Jurado el Profesor Esmitt Ramírez.

_____
Prof. Esmitt Ramírez

_____ _____
Prof. Francisco Sans Prof. Eugenio Scalise

Dedicatoria

A mi madre, quien ha dado todo para que yo pudiese tener un futuro. Sin su esfuerzo y dedicación como madre y como amiga no me hubiese sido posible llegar al punto en el que me encuentro hoy en día.

Agradecimientos

A mi madre quien ha sido un apoyo constante a lo largo de mi vida y un símbolo de que querer es poder.

A mi novia, quien se ha asegurado de que jamás me rinda y jamás ha dudado de mí.

A mis amigos, los cuales me han apoyado en todo momento y siempre han estado ahí cuando ha sido necesario.

A mi tutor, Prof. Esmitt Ramírez, por asistirme durante mi formación como profesional y durante la elaboración de este trabajo.

Resumen

La generación procedimental de contenido es una técnica que ha sido utilizada en computación para generar contenido a partir de un algoritmo. En los videojuegos, los algoritmos de PCG han jugado un papel importante a la hora de presentar contenido único a los jugadores mediante la permutación de contenido previamente creado por los desarrolladores. Sin embargo, el género de videojuegos MOBA actualmente no contempla la posibilidad de generar contenido procedimentalmente debido a la alta cohesión que presenta su diseño. Es por esto que en este trabajo se opta por elaborar un videojuego del género MOBA que pueda hacer uso efectivo de la generación procedimental de contenido sin desmejorar la calidad y objetivos del juego. Estos algoritmos de PCG se utilizan para generar dinámicamente el mapa y los personajes utilizados en el juego, los cuales se adaptan en tiempo real al estilo de juego del jugador.

Palabras claves: generación procedimental de contenido (PCG), Multiplayer Online Battle Arena (MOBA).

Índice

Introducción

La generación procedimental de contenido (PCG) tiene como finalidad incrementar la cantidad de contenido presente en una aplicación por medio de la utilización de algoritmos. Desde el comienzo de su utilización en los videojuegos, estas técnicas de PCG han demostrado ser una herramienta de utilidad que no solo aumenta la aleatoriedad del contenido presentado al jugador, evitando así la repetición de contenido, sino que también reduce el tiempo de desarrollo del videojuego.

Hoy en día las ventajas de la utilización de los algoritmos de PCG en los videojuegos es evidente, sin embargo, existen escenarios donde la viabilidad de la utilización de estas técnicas se reduce debido a la complejidad del diseño del videojuego. Este es el caso de los videojuegos del género MOBA, los cuales implementan una gran cantidad de contenido para traer variedad al juego y esto lo hacen por medio de los desarrolladores.

En este documento se describen las técnicas y decisiones de diseño tomadas para poder crear un videojuego del género MOBA que haga uso extensivo de algoritmos de PCG para poder crear su contenido. El mapa donde se desenvuelven los jugadores en el juego, el cual suele ser estático, es creado en tiempo real como se describe en el Cap. 3. Además de esto, los personajes han sido rediseñados para adaptarse mejor a un concepto de PCG, donde el personaje es capaz de adaptar sus características según el estilo de juego del jugador (Cap. 4).

Se implementa un sistema de red completo para controlar el pase de mensajes entre cliente y servidor para poder implementar el aspecto multi-jugador de los MOBA. Haciendo uso de este sistema, se realizan pruebas para demostrar el funcionamiento del sistema como un todo y la viabilidad de la utilización de PCG en este género (Cap. 6).

Capítulo 1 – Descripción del problema

Los videojuegos se dividen en diversos géneros, los cuales vienen dados por las características del juego. Los videojuegos del género MOBA (Multiplayer Online Battle Arena) son el resultado de la mezcla entre el género de estrategia y el de acción. En estos juegos se ofrece una gran cantidad de contenido al jugador en la forma de distintos personajes, que presentan características únicas entre sí. Sin embargo, Esto implica que el diseño de estos juegos está altamente cohesionado, lo que significa que agregar contenido nuevo usualmente viene acompañado de ajustes al contenido ya existente.

Debido a las características de los juegos MOBA, estos representan problemas de balance que son directamente proporcionales a la cantidad de personajes existentes dentro del juego. Los desarrolladores de estos juegos crean nuevos personajes frecuentemente y deben asegurar que estos no tengan capacidades notoriamente mayores a las de los personajes que ya existían. Es debido a esta problemática de diseño, que este tipo de juegos implementa frecuentemente actualizaciones de *balance* para corregir desigualdades en las capacidades de los personajes.

Además de los personajes, el mapa en el cual se desenvuelven los jugadores en un MOBA tiene un diseño estratégico que presenta ciertas restricciones para los jugadores, la cuales no pueden ser modificadas sin impactar la manera en la que el juego funciona.

1.1 – Motivación

Los videojuegos del género MOBA constan de contenido estático y que se encuentra altamente cohesionado, lo que significa que agregar contenido se vuelve una tarea incrementalmente compleja y que consume una gran cantidad de tiempo para los desarrolladores. Es por esto que se optó por crear **Casmage**, un MOBA que explora la posibilidad de utilizar herramientas como la PCG (Generación Procedimental de Contenido) como apoyo para creación de contenido nuevo y dinámico, incrementando así la variedad de contenido presentada al jugador

1.2 – Objetivos generales

Desarrollar un videojuego del género MOBA que haga uso activo de algoritmos de PCG en la creación del mapa de navegación y en la evolución de los personajes durante el juego.

1.3 – Objetivos específicos

- Definir el conjunto de reglas que deben seguirse para lograr crear procedimentalmente un mapa que cumpla con todos los requisitos necesarios para ser utilizado en un videojuego del género MOBA.
- Desarrollar un sistema capaz de generar en tiempo real un mapa.
- Implementar un sistema que permita adaptar dinámicamente las características de un personaje con la manera de jugar del jugador.
- Implementar un sistema de red capaz de soportar el esquema de juego de un videojuego del género MOBA.

1.4 – Solución propuesta

Para la generación procedimental del mapa, se hace uso de un algoritmo evolutivo, el cual es guiado por un conjunto de normas establecidas para lograr que el mapa resultante tenga las características principales de un mapa común en un MOBA.

Crear un personaje capaz de adaptarse al estilo de juego de un jugador es una tarea que necesariamente debe hacer uso de un algoritmo de PCG. Para lograr implementar este tipo de comportamiento en un personaje, es necesario crear un sistema capaz de interpretar las intenciones de un jugador y posteriormente realizar los cambios necesarios en su personaje para fomentar su estilo de juego.

Para poder permitir a los jugadores interactuar entre sí por medio de una red, es necesario implementar un sistema capaz de manejar todos los mensajes de red pertinentes a los eventos que se desarrollan en un MOBA, haciendo uso de un servidor autoritativo.

Capítulo 2 – Marco teórico

La Generación Procedimental de Contenido (PCG, por sus siglas en inglés) puede ser definida como la creación de contenido aleatorio o pseudoaleatorio durante la ejecución de una aplicación.

Este tipo de algoritmos se observó por primera vez en los fractales. Un fractal es un conjunto matemático que posee la característica de la "auto-similaridad" a cualquier escala, es decir, es posible escalar el conjunto y siempre obtener elementos que siguen un patrón específico. Por ejemplo, algoritmos como el del conjunto de Mandelbrot[1] generan un contenido *infinito* (ver Fig. 1) y pueden ser considerados algoritmos de PCG.

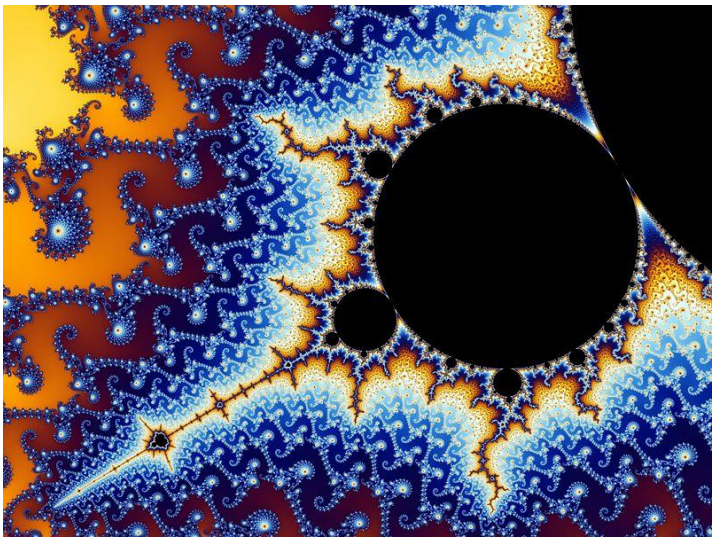

Figura 1: Una representación del conjunto de Mandelbrot observado a diferentes escalas de acercamiento.

Estos algoritmos usualmente generan contenido basándose en entradas que comúnmente varían lo suficiente como para producir resultados que muestran cierta aleatoriedad. Un ejemplo de ello ocurre en la aplicación SpeedTree[2], la cual al hacer uso de algoritmos de PCG, es capaz de crear árboles que difieren mucho en sus características, como son la altura y número de ramas. Dicha aplicación también es utilizada para crear bosques enteros generando resultados similares a los que se observan en bosques reales.

Existe un gran número de sistemas que hace uso de contenido generado aleatorio o pseudoaleatoriamente y es para estos sistemas que los algoritmos PCG generan contenido. Las aplicaciones pueden ir desde simulaciones climáticas, hasta creación de objetos muy específicos según su ambiente. Estos algoritmos incrementan la capacidad de las aplicaciones para generar contenido que usualmente tomaría mucho tiempo si no se utilizaran.

Si bien la PCG representa un gran aporte a campos como las ciencias, en el área de los videojuegos está incrementando constantemente su uso. En dicha área no solo ayudan a disminuir el tiempo de desarrollo, sino que también pueden ser utilizados como herramientas creativas para ayudar a los diseñadores o incluso para aumentar el tiempo de jugabilidad.

Debido al notable incremento del uso de la PCG en los videojuegos, a continuación se expondrán las consecuencias que trae la utilización de estos algoritmos y como pueden ser usados para impulsar la experiencia del jugador en un videojuego.

2.1 - PCG en Videojuegos

Los algoritmos PCG representan una gran ayuda para los desarrolladores y diseñadores de videojuegos, ya que pueden ser utilizados para disminuir el tiempo de desarrollo de diversos aspectos, como por ejemplo los niveles. Debe tomarse en cuenta que estos algoritmos deben conservar aspectos de gran importancia para poder generar contenido que sea considerado parte del juego.

Para este trabajo se optó por definir las siguientes características como los aspectos que deben cumplir los algoritmos PCG:

- Jugabilidad: El contenido generado no entorpece de manera no intencional el desarrollo del juego. Por ejemplo, el camino que debe seguir el jugador no está obstruido por paredes.
- Alcanzabilidad: El contenido generado debe poder ser alcanzable por el jugador en caso de que dicho contenido necesite que el jugador este a cierta distancia, como mínimo, para poder ejecutar su funcionalidad. Por ejemplo, si el jugador debe activar un botón, debe tener la posibilidad de acercarse lo suficiente para hacerlo.
- Coherencia: Es necesario que el contenido generado conserve relación con su medio ambiente para que se considere parte del mismo. Por ejemplo, en un ambiente desértico no hay nieve.
- Balance: Se debe garantizar que el contenido generado no ponga al jugador en una posición demasiado o muy poco ventajosa. Por ejemplo, el jugador no debe ser invencible y no debe ser excesivamente frágil.

Existe un gran número de géneros de videojuegos y por esto se clasificarán, para este trabajo, en géneros primarios y secundarios. Los géneros primarios son aquellos que presentan un estilo de juego único, esto quiere decir que no comparten características con otros géneros. Los secundarios son géneros compuestos que mezclan características de 2 o más primarios.

Los géneros primarios más relevantes para este trabajo son:
- Estrategia: El jugador debe tomar decisiones estratégicas sobre cómo distribuir sus recursos y manejar a las unidades que están bajo su mando para poder lograr el objetivo planteado en el juego.
- RPG (*Roll Playing Game*): El jugador debe ganar experiencia y con esto aumentar su nivel lo suficiente como para cumplir con los objetivos que se presentan a lo largo del juego. En este tipo de juegos normalmente se le da al jugador la posibilidad de elegir qué características se adaptan mejor a su estilo de juego, o que aspectos de su personaje desea incrementar para lograr un objetivo.

En este trabajo de investigación se considera que solo es necesario definir un tipo de género secundario, el cual es un género llamado MOBA (*Multiplayer Online Battle Arena*). Este género es una composición entre estrategia y RPG. El jugador controla un único personaje que es capaz de ganar experiencia y subir de niveles, además de administrar sus recursos y actuar de manera estratégica junto con 4 jugadores aliados para poder cumplir el objetivo final del juego que consiste en vencer al equipo contrario.

2.2 - Tipos de PCG según el contenido generado

Existen diversas clasificaciones que se pueden utilizar para categorizar los distintos PCG. Se tomarán en cuenta 2 de las propuestas por Doull[3]:

2.2.1 - Generación aleatoria del nivel en tiempo de ejecución

Son algoritmos de PCG cuya función es generar uno o más niveles (que incluye el terreno y todos los elementos necesarios para hacer un nivel completo del juego) en tiempo real o al comienzo del juego. Un ejemplo de esto son los juegos de estilo *Rogue* como Diablo[4], en el cual el mundo es generado procedimentalmente cada vez que el jugador comienza el juego.

Los juegos tipo *Rogue* caben dentro de la categoría de los mundos persistentes dentro de este tipo de algoritmos. Existen también los algoritmos que generan mundos no persistentes los cuales son generados cada vez que el jugador los visita, de manera que cuando cambie de nivel, siempre entre a uno totalmente nuevo, a diferencia de generar el mundo únicamente cuando se comienza el juego desde cero. Un ejemplo de esto se puede apreciar en la Fig. 2 en el juego Spelunky[5].

Figura 2: Primer nivel en el juego Spelunky generado 4 veces.

2.2.2 - Contenido intervenido por el usuario

La principal diferencia que poseen ante los tipos de PCG previamente mencionados, es la alta importancia de la participación del usuario. El usuario gana mucho más control sobre el resultado del algoritmo y gracias a esto los resultados serán siempre los que el jugador ha preferido sin importar las razones.
Un ejemplo de ello es el juego Spore[6] donde se permite al usuario cambiar el contenido según las decisiones que éste toma. Igualmente, existe un sistema guiado por el usuario para la PCG llamado Dryad[7], el cual fue desarrollado en la Universidad de Stanford[8] para permitir a los usuarios explorar las posibilidades de un mundo virtual.

2.3 – Tipos de PCG según sus características

Para explicar de mejor manera los diferentes tipos de algoritmos que existen, se utilizará la clasificación descrita en el artículo *Search-based Procedural Content Generation*, presentado por J. Togelius, G. Yannakakis, K. Stanley y C. Browne[9], la cual nombra las diferentes características de los algoritmos PCG, diferenciándolos de acuerdo a la manera en que funcionan y son implementados.

2.3.1 – En Línea y Fuera de Línea

Los algoritmos que son utilizados durante el desarrollo del juego, para poder crear contenido rápidamente que luego será modificado por los desarrolladores para el producto final, tienen la característica de ser fuera de línea.

Los algoritmos que trabajan mientras el juego está en curso y generan el contenido en tiempo real son en línea.

2.3.2 – Necesario y Opcional

Se considera que el contenido generado por el algoritmo PCG es de contenido necesario cuando el jugador depende de este contenido para poder avanzar en el juego, mientras que el de contenido opcional es aquel que puede ser ignorado por el jugador sin afectar su progreso.

La principal diferencia entre estos 2 es que el algoritmo de contenido necesario está obligado a conservar la Jugabilidad del juego, mientras que el de contenido opcional no tiene esta restricción debido a que le es imposible generar contenido que pueda entorpecer el progreso del jugador.

2.3.3 – Semillas Aleatorias y Vectores de Parámetros

Existen diversas maneras de crear las entradas que utilizaran los algoritmos PCG para generar el contenido del juego. Se puede utilizar una semilla aleatoria para generar todo el contenido o se puede utilizar un vector de parámetros, el cual contiene un conjunto de valores donde cada valor representa un aspecto específico del contenido que generará el algoritmo.

2.3.4 – Generación Estocástica y Determinista

En un extremo se tienen los algoritmos que siempre generan resultados diferentes sin importar si se utilizan las mismas entradas, los cuales denominaremos estocásticos, en el otro extremo se tienen los algoritmos deterministas, los cuales generaran el mismo contenido si se utiliza la misma entrada.

Esta diferenciación es un tanto objetiva ya que se debe tomar en cuenta hasta que punto un algoritmo es completamente estocástico o determinista. Es por esto que es posible encontrar algoritmos que están en algún punto entre los 2 extremos antes mencionados.

2.3.5 – Constructivo y Genera-y-Evalúa

Los algoritmos de PCG Constructivos son aquellos que en su fase de creación de contenido constantemente realizan pruebas para determinar si el contenido sigue siendo apto o no.

Por otro lado los algoritmos de genera-y-evalúa realizan una tarea similar a los anteriores, pero con la diferencia de que las evaluaciones para determinar si el contenido es apto o no se realizan cuando el algoritmo termina de generar el contenido, si el candidato de contenido no es aceptado, el algoritmo busca otro y el proceso se repite hasta que un candidato pase la prueba.

Existe un caso especial para los algoritmos genera-y-evalúa, el cual es llamado PCG basado en búsqueda (*Search-based Procedural Content Generation*). La particularidad de estos algoritmos está en que utilizan computación evolutiva para elegir al mejor candidato de contenido.

Comúnmente los algoritmos evolutivos (y por ende los algoritmos de PCG basados en búsqueda) siguen los siguientes pasos para llegar al mejor candidato:

1- Se genera un conjunto de candidatos de contenido.
2- Se evalúa con una función de aptitud que tan *aceptable* es cada candidato.
3- Se generan nuevos candidatos de contenido, pero esta vez existe mayor probabilidad (según los resultados de la función de aptitud) de que se presenten las características de los candidatos más aptos evaluados previamente.
4- Se repiten el paso 2 y 3 hasta que se encuentre un candidato de contenido cuya función de aptitud determine que es un candidato aceptable.

2.4 – Trabajos previos

Existe una gran variedad de videojuegos y trabajos que involucran la utilización de PCG. Todos estos implementan estos algoritmos de maneras muy diversas, según sus necesidades. Sin embargo, aun cuando hay una gran variedad en los distintos algoritmos de PCG existentes, se hará un enfoque en el llamado Algoritmo de PCG basado en búsqueda, debido a la flexibilidad que proporciona.

2.4.1 – PCG basado en Búsqueda

Los algoritmos PCG basados en búsqueda son en su mayoría estocásticos debido a la manera en la que seleccionan a sus candidatos de contenido. Debido a esto el nivel de aleatoriedad con el que se genera el contenido es alto, lo cual es deseable en los algoritmos de PCG.

2.4.1.1 – Funcionamiento de los algoritmos PCG basados en Búsqueda

Los algoritmos de PCG basados en búsqueda utilizan computación evolutiva para evaluar un conjunto de candidatos de contenido y poder llegar a escoger el más apto.

Al momento de iniciar su ejecución, el algoritmo genera aleatoriamente un conjunto de candidatos de contenido, los cuales difieren entre sí en distintos aspectos. A cada candidato de este conjunto se le aplicará una función que determinará qué tan aceptable es un candidato. Esta función es llamada función de aptitud o función de *fitness*.

Luego de que la función de aptitud es aplicada sobre el conjunto de candidatos, el algoritmo procede a crear un nuevo conjunto, el cual contendrá candidatos de contenido que muy probablemente poseerán las características más resaltantes de los individuos que los precedieron. Así, se asegura que mientras más conjuntos nuevos de candidatos se creen, más se aumenta la posibilidad de encontrar un candidato ideal.

2.4.1.2 – Desventajas de los algoritmos PCG basados en búsqueda

Los algoritmos utilizados en el área de computación evolutiva son en su mayoría iterativos. Este es el caso de los PCG basados en búsqueda. Cuando se denomina a un algoritmo iterativo, significa que para poder llegar a un resultado aceptable se debe ciclar entre una cantidad de candidatos hasta llegar al resultado. El problema que trae esto es que no se puede saber a priori cuantas veces es necesario iterar, por lo cual muchas veces se opta por aplicar diferentes estrategias para asegurar que el algoritmo no itere una cantidad excesiva de veces.

Además de las iteraciones, otro problema que enfrenta este tipo de algoritmo es que los candidatos no pueden tener una gran cantidad de parámetros modificables (características del candidato que pueden cambiar conforme itera el algoritmo). Esto se debe a que mientras más parámetros estén sujetos a modificaciones, la función de aptitud deberá contemplarlos y a su vez evaluarlos, lo cual probablemente suponga una carga mayor al procesamiento y un incremento en el tiempo de ejecución del algoritmo. Por ello, el tiempo de ejecución de estos algoritmos está atado a la complejidad de la función de aptitud y la cantidad de parámetros que se evaluaran por cada candidato.

2.4.1.3 – Limitaciones de los algoritmos PCG

Los algoritmos PCG son herramientas utilizadas para producir contenido para un juego, pero con la tecnología y métodos actuales para crearlos, no es posible emular efectivamente el proceso de pensamiento humano en cuanto a lo que puede hacer o no llamativo a un tipo de contenido especifico.

Estos algoritmos son ideales para generar contenido que puede ser dispuesto de manera aleatoria, pero contenido más estructurado y con sentido dentro de su ambiente, como por ejemplo un nivel de un juego de rompecabezas, es muy difícil de crear debido a la cantidad de variables que deben ser evaluadas. Por ello, estos algoritmos son mejor utilizados como una herramienta para ayudar al desarrollador.

2.4.2 - Trabajos relevantes

A continuación, se presenta un resumen de 2 trabajos enfocados en PCG para videojuegos.

The rules behind a procedural death-match map generator, es un trabajo realizado por Erwin P. Wilhelmus[10], en el cual explica los procesos llevados a cabo para poder implementar un generador de mapas del tipo de juego *death-math* para el juego Unreal Tournament 3[11].

Death-match es un tipo de juego muy popular y básico. Se caracteriza por la inexistencia de equipos, por lo cual todos los jugadores compiten entre ellos. El objetivo principal es vencer a la mayor cantidad de oponentes posibles antes de que acabe el tiempo, o llegar al máximo puntaje.

El trabajo comienza con una breve investigación sobre los distintos métodos que existen para analizar el comportamiento de los jugadores en un mapa específico, lo cual es especialmente importante para los diseñadores, ya que con esta información les es posible cambiar elementos en un mapa para hacerlo más agradable a los jugadores. El método más resaltante utilizado para este análisis, es el llamado mapa de calor (Fig. 3), el cual se encarga de llevar la información sobre la posición de cada jugador en un mapa cada instante de tiempo, esto le permite al diseñador determinar qué áreas del mapa son más transitadas por los jugadores.

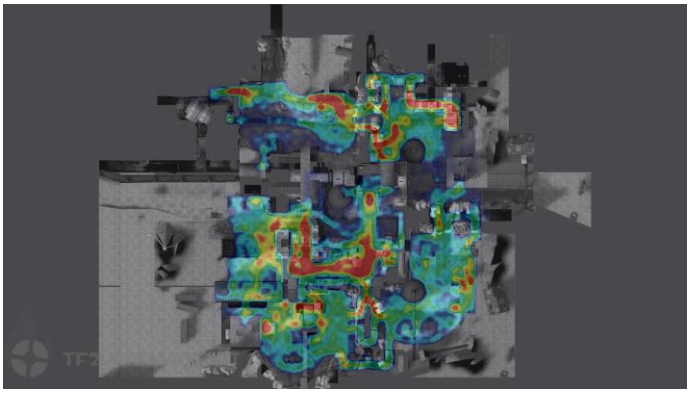

Figura 3: Mapa de calor en un videojuego. Las áreas menos y más transitadas varían entre los colores azul y rojo respectivamente.

Luego de esto se recopila el conjunto de mapas de *death-match* más aceptado y jugado por la comunidad de jugadores de Unreal Tournament 3 para analizar los mismos.

Una vez obtenidos estos mapas, se comienza a analizar la composición de cada uno, lo cual lleva a la identificación de un conjunto de reglas a seguir para la elaboración de estos mapas, como son:

- Lo mapas están compuestos por rutas circulares interconectadas.
- Existen rutas de *Novato*, las cuales no requieren que el jugador salte para recorrerlas.
- Existen las rutas *Avanzadas*, las cuales permiten recorrer el mapa rápidamente.
- Existen las rutas *Expertas*, las cuales requieren una combinación especial de técnicas por parte del jugador para poder ser alcanzadas.
- Los mapas están construidos con múltiples niveles de alturas.
- El máximo ángulo en el cual un jugador puede girar sin perder tracción es 22.5º.

Además de esto, existen tipos de áreas que deben estar presentes:

- Áreas de enfoque: grandes cuartos donde transitan frecuentemente los jugadores.
- Corredores: interconectan áreas de enfoque.
- Cuartos menores: sirven como cruces y usualmente se crean al intersectar 2 o más corredores.
- Escaleras: son construidas dentro de cuartos, actúan como puentes y permiten a los jugadores alcanzar ciertas áreas del mapa.
- Transporte no-arquitectónico: incluye elevadores, plataformas de salto y teletransportadores.
- Obstáculos: objetos como cajas o cualquier elemento que un jugador pueda utilizar para cubrirse.
- Aparición de jugadores: puntos en el mapa donde un jugador puede reaparecer luego de haber sido vencido. El jugador siempre aparece en un punto lejos de otros jugadores.

Al momento de implementar el algoritmo para generar este tipo de mapas, todas estar normativas definidas luego de analizar los mapas predilectos por los jugadores, son tomadas en cuenta por el algoritmo de PCG para la creación del nuevo mapa.

En este trabajo se define formalmente un conjunto de reglas y lineamientos que debe seguir cada elemento involucrado en un mapa de este tipo, para poder proveer la experiencia adecuada. Además de esto, culmina la creación de una aplicación que permite a un usuario intervenir en el proceso de creación del mapa del algoritmo PCG, para personalizar los resultados.

En el trabajo **Automatic content generation in the Galactic Arms Race game**, realizado por ErinJ. Hastings, Ratan K. Guba y Kenneth O. Stanley[12] se desarrolla un sistema de PCG que funciona en un juego multi-jugador en Web (en línea).

En este juego, llamado Galactic Arms Race (GAR), al ingresar por primera vez, los jugadores tienen un conjunto de armas a su disposición. A medida que un jugador avanza en el juego, se le da la posibilidad de tomar armas nuevas que son creadas por el sistema de PCG, el cual utiliza un algoritmo denominado cgNEAT (content-generating NeuroEvolution of Augmenting Topologies) cuya función es generar patrones de disparos distintos en armas nuevas. Estos patrones son los que determinan la viabilidad de un arma.

El sistema utiliza como método de entrada las elecciones de los jugadores, estas elecciones consisten en desechar y tomar nuevas armas que los mismos encuentran (Fig. 4). El desechar un arma, implica que el sistema reducirá la posibilidad de que un jugador encuentre un arma con un patrón de disparo similar, ya que estadísticamente hablando, si muchos jugadores desechan un arma con un patrón similar, es probable que dicho patrón no sea del agrado de los jugadores.

Figura 4: Arma generada de forma dinámica por el sistema en el videojuego Galactic Arms Race.

Este algoritmo se enfoca en producir las armas que son más populares entre los jugadores, pero debido a que utiliza un algoritmo evolutivo, es imposible generar un arma que haya existido previamente. Por lo tanto, el algoritmo tiene altas probabilidades de generar un arma con un patrón similar al de las armas más populares, pero siempre será un patrón único en el juego.

Luego de realizar un conjunto de pruebas, los resultados obtenidos por este trabajo resultaron satisfactorios para los autores. El sistema generó patrones de armas que los autores no habían previsto, lo cual resalta el hecho de que un algoritmo evolutivo es una buena opción para la generación procedimental de contenido. Además de esto, el sistema se balancea automáticamente al permitirles a todos los jugadores tener la posibilidad de obtener armas que actualmente son populares entre la mayoría de los jugadores.

Este trabajo representa un gran ejemplo de cómo un algoritmo de PCG de contenido intervenido por el usuario funciona en un juego en línea.

3.3 – Aplicaciones de la PCG

Actualmente existen diversas áreas en las cuales se hace uso de la PCG. Los videojuegos hoy en día están haciendo un uso cada vez mayor de este tipo de algoritmos implementándolos de distintas maneras y para lograr diferentes objetivos.

En el área del cine, se utilizan los algoritmos de PCG, como se usaban en sus inicios, para generación de terreno. Las aplicaciones de PCG ayudan a dar un tono mayor de realismo a cualquier película que necesite crear ambientes virtuales, pero su uso no está limitado exclusivamente a la generación de terreno. Así como ha sucedido con los videojuegos, la PCG puede adaptarse también en el cine para generar cualquier tipo de contenido que se necesite de la manera que se necesite, como por ejemplo, un bosque denso que presenta una gran variedad de vegetación.

La PCG también puede ser utilizada en el área de la arquitectura. Existen trabajos que hablan sobre la morfogénesis digital[13], la cual es una técnica utilizada para la creación de estructuras digitales cuyo proceso de formación se asemeja al de las estructuras naturales.

En el área científica, es posible utilizar algoritmos de PCG para crear datos de entrada para simulaciones de sistemas complejos, como por ejemplo el clima o la erosión.

Capítulo 3 – Mapas Procedimentales

En este capítulo se realizará un análisis detallado sobre el diseño del mapa del videojuego Dota 2[14] desarrollado por la empresa Valve Corporation. Este juego es la secuela del primer MOBA, Dota[15], donde se replica el funcionamiento y diseño de su predecesor y debido a que está constantemente actualizado, se utilizará como modelo de referencia.

Luego, analizando la composición y diseño de un mapa del género MOBA, se explicarán a fondo de que maneras influyeron estos aspectos en el diseño del algoritmo para generar procedimentalmente el mapa.

3.1 – Mapa del juego Dota 2

El mapa de Dota es un mapa estático que se encuentra dividido en 2 mitades. Cada mitad representa el territorio de un equipo, y su frontera se representa por un río, dividiendo equitativamente el territorio en 2 mitades iguales (ver la Fig. 6).

En el mapa, se encontraran unas bases que corresponderá uno a cada mitad que a su vez se asocia a un equipo. Las bases de cada equipo se encuentran dispuestas en esquinas opuestas del mapa, permitiéndoles así tener la mayor distancia posible entre ellas. Dentro de estas, se encuentra el objetivo que debe ser destruido para ganar el juego. En la terminología del juego Dota, estos objetivo se denominan Ancient, o Ancestro (nombre que se empleará en referencias futuras).

Al mismo tiempo, existen las llamadas líneas o carriles. Estas líneas son caminos que conectan directamente a las 2 bases del mapa y deben ser recorridas obligatoriamente para lograr el objetivo de destruir la base enemiga.

Adicionalmente a las líneas, existen zonas entre las mismas, denominadas junglas. En estas zonas, existen intrincados caminos para recorrer el mapa, que sirven como vías para posicionarse tanto defensiva como ofensivamente. Es importante destacar, que el mapa está dispuesto de tal forma de crear, como jugador, una estrategia de ataque y defensa hacia los Ancient.

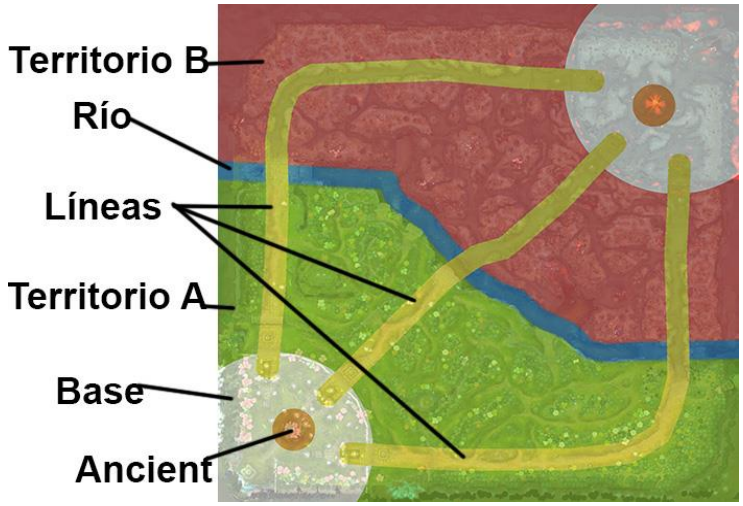

Figura 6. Ejemplo de una división de las zonas dentro de un mapa en Dota 2.

Antes de explicar detalladamente cada componente del mapa, es necesario introducir actores que juegan un papel fundamental durante un juego (ver Fig. 7). Estos son:

- Torres: Son estructuras cuya función es atacar a cualquier oponente que entre dentro de su rango de ataque.
- Minions (o Secuaces): Son criaturas que tienen como objetivo destruir la base enemiga. Son creadas en intervalos de tiempo fijos y constantemente recorren las líneas.
- Ancient: Esta es la estructura que debe destruir un equipo para poder declararse victorioso en el juego. Se encuentra dentro de la base de cada equipo.
- Niebla de guerra: La niebla de guerra se encarga de mostrarle a los integrantes de cada equipo únicamente lo que sus aliados, o el mismo, son capaces de ver. Existe para evitar que todos los jugadores sepan la ubicación de sus adversarios a todo momento. En la Figura 7 se puede apreciar una imagen de la niebla de guerra en acción. En este caso, el jugador, que se encuentra en el centro, ve todo lo que está a su alrededor excepto la zona que se encuentra detrás del árbol, la cual se oscurece ya que el mismo bloquea su visión.

Figura 7. Elementos del videojuego Dota 2: ancients, torres, minions y niebla de guerra.

A continuación se explicaran en qué consisten los distintos componentes de un mapa, cuál es su función, como interactúan entre ellos y como afectan el juego en su totalidad

3.1.1 – Río
Como se mencionó anteriormente, existe un río que recorre el mapa y que divide al mismo en 2 porciones iguales.

Este río es la ruta más corta para moverse de una línea a otra estando en la mitad del mapa. Es la ruta más transitada durante todo el juego y presenta un alto valor estratégico.

3.1.2 – Base
La base es donde cada equipo comienza el juego. Es una zona protegida por muros que ofrece refugio a los jugadores. Está compuesta por:

- Fuente: Es donde los jugadores comienzan el juego. Es la zona más segura en todo el mapa pues cura a los jugadores constantemente y se encuentra resguardada por una torre especial que es más poderosa que las demás.
- Tienda: Se encuentra localizada a un lado de la fuente y es donde los jugadores pueden comprar artefactos para su personaje.

El Ancient se encuentra protegido por 2 torres que no pertenecen a ninguna línea en específico, la cuales además se encuentran suficientemente cerca una de otra como para atacar al mismo oponente a la vez.

3.1.3 – Línea o carril
Son caminos que interconectan directamente una base con la otra. Tienen sus puntos de comienzo y final en las salidas de cada base. Están resguardadas por 3 torres por equipo, sumando un total de 6 torres que pertenecen a cada línea.

Para poder atacar directamente el Ancient de un equipo, es necesario primero avanzar completamente por al menos una de sus líneas. Esto implica destruir las 3 torres que resguardan dicha línea.

3.1.4 – Junglas
Son los espacios que se encuentran entre las líneas. Si se utiliza el río como línea divisoria, existen en total 4 junglas en un mapa, donde 2 corresponden a cada equipo ya que se encuentran en su territorio.

Estas junglas proveen caminos que permiten acceder a diversos sectores de cada línea, así como también al río sin la utilización de las líneas, lo cual le permite a los jugadores pasar desapercibidos mientras recorren el mapa.
Además de proveer caminos, en las junglas se encuentran los campamentos. Estos campamentos albergan minions que no pertenecen a ningún equipo y que funcionan de manera similar a los minions. Tienen como propósito proveer a los jugadores que pasen demasiado tiempo en la jungla dinero y experiencia para que los mismos no estén en desventaja con respecto a los demás jugadores.

3.1.5 – Diseño

El primer objetivo del juego es destruir el Ancient del equipo contrario, pero esto es imposible sin antes haber destruido las torres de al menos una línea. Tomando en cuenta esto, se puede afirmar que el juego tiene 2 objetivos primarios, destruir las torres de una línea y posteriormente el Ancient.

Para poder cumplir el primer objetivo, es necesario reducir por la mayor cantidad de tiempo posible las defensas del equipo contrario y aprovechar estas debilidades para poder dañar lo suficiente la torre y destruirla. Las defensas de un equipo se pueden reducir de las siguientes maneras:

- Venciendo a sus minions.
- Venciendo a jugadores oponentes.
- Distrayendo a los jugadores oponentes para que se alejen demasiado de una torre, lo cual da tiempo suficiente para hacer un daño considerable.

Para poder demostrar lo complejo que resulta vencer a un jugador oponente, es necesario definir 3 tipos de zonas que representan el riesgo que corre un jugador de ser atacado (ver Fig. 8). Estas zonas son:

- Zona verde: Una zona verde es aquella que se encuentra dentro del rango de ataque de una torre aliada. Es la zona más segura en la que un jugador puede estar fuera de la fuente de su base.
- Zona amarilla: Es la zona que se encuentra al terminar la zona verde. En esta, el jugador ya no se encuentra bajo la protección de la torre, sin embargo, está suficientemente cerca de la misma como para volver a ella si es atacado y probablemente salga ileso.
- Zona roja: Comienza después de la amarilla y es donde el jugador se encuentra más vulnerable. Cabe acotar que para poder atacar una torre enemiga, obligatoriamente se debe estar en una zona roja.

Figura 8. Representación de las zonas de peligro para un jugador.

Tomando en cuenta las zonas previamente explicadas, se debe notar que es considerablemente difícil atacar a un oponente que se encuentra en una zona amarilla o verde, ya que el mismo se encuentra en una posición defensivamente ventajosa. Sin embargo, para evitar que un jugador esté demasiado seguro en una línea, existe suficiente separación entre las torres de cada una como para tener una zona roja de por medio, lo cual le ofrece oportunidades de ataque al oponente.

Se debe recordar que los caminos de las junglas están diseñados para proveer oportunidades ofensivas y defensivas para ambos equipos. Es por esto que las entradas a la jungla de cada línea nunca deben estar ubicadas completamente dentro de una zona verde, pues esto parcializaría su uso para un solo equipo y haría la línea demasiado segura. También es importante resaltar que las junglas deben tener al menos un acceso al río, ya que es vital que los jugadores puedan moverse sin ser vistos (es decir, evitando las líneas) a lo largo del mapa.

Al analizar lo anteriormente expuesto, se puede inferir que un MOBA es un juego que consiste enteramente en defensa y ataque. Los jugadores deben destruir las torres enemigas y al mismo tiempo proteger las suyas, lo cual los obliga a colocarse constantemente en posiciones de riesgo donde son vulnerables a ataques.

El mapa no es más que un conjunto de restricciones espaciales que determinan que tan difícil es llegar a un lugar. Sin embargo, al utilizar este mapa en el contexto de un MOBA, se comienzan a interpretar estas restricciones espaciales como oportunidades estratégicas, las cuales dictan el ritmo que tendrá en juego. Por ello, la forma o configuración del mapa no son factores influyentes en los aspectos de un juego MOBA siempre y cuando se establezcan los objetivos necesarios, los cuales son destruir las torres y posteriormente el Ancient oponente.

3.2 – Mapa generado procedimentalmente

Previo a explicar cómo se implementó el algoritmo para crear procedimentalmente el mapa en Casmage, es necesario dar una breve introducción del concepto de algoritmo evolutivo.

3.2.1 – Algoritmos evolutivos

Son algoritmos utilizados en su mayoría en áreas de inteligencia artificial. Funcionan imitando el comportamiento de los seres vivos en la naturaleza, donde los individuos más aptos sobreviven[16]. La supervivencia del más apto implica que las nuevas generaciones tendrán variaciones de estas mismas características, las cuales les permitirán sobrevivir y reproducirse.

Estos algoritmos se utilizan creando inicialmente un conjunto aleatorio de individuos, los cuales son alterados según ciertos criterios dependientes del enfoque de algoritmos evolutivos que se esté utilizando. Los algoritmos genéticos, que son un subconjunto de los evolutivos, cruzan a los individuos y tienen una probabilidad de mutar alguno de ellos. Una vez cruzados, los individuos resultantes forman parte de una nueva generación y el proceso se repite.

Para elegir cuales individuos son los más aptos (lo que significa que tienen mayor probabilidad de ser cruzados con otros, permitiendo que sus características aparezcan en la siguiente generación), se utiliza la llamada función *fitness* o de aptitud. Esta función evalúa a cada individuo por separado y determina su probabilidad de ser cruzado.

3.2.2 - MOEA

Existe un tipo de algoritmo evolutivo denominado MOEA (Multi-Objective Evolutionary Algorithm, o Algoritmo Evolutivo Multi-Objetivo) que funciona de igual manera que un algoritmo evolutivo común. Sin embargo, su diferencia yace en que utiliza más de una función de aptitud para evaluar a cada individuo.

La función de aptitud usualmente se utiliza para definir como es un individuo tomando en cuenta un único aspecto. Por ejemplo: se puede utilizar una función de aptitud para evaluar si un terreno es transitable por un jugador, con lo cual se obtiene un algoritmo capaz de determinar si un camino es transitable.

Cuando se utilizan múltiples funciones de aptitud, se introduce el problema de que un individuo aceptado por una característica, puede ser rechazado por otra. Si utilizamos el mismo ejemplo anterior, pero agregamos una función de aptitud adicional, la cual se encarga de evaluar la distancia necesaria para recorrer el camino, entonces se introduce la posibilidad de que aun cuando el camino sea transitable, puede ser rechazado si es demasiado largo.

Este comportamiento permite al algoritmo considerar diversos factores a la hora de seleccionar un individuo. Sin embargo, se debe cuidar no utilizar funciones de aptitud demasiado complejas. Esto se debe a que teóricamente es posible utilizar infinitas funciones de aptitud, pero se debe considerar que por cada función de aptitud utilizada se incrementa el tiempo de procesamiento para elegir candidatos.

3.2.3 – Implementación

Los candidatos a utilizar por el algoritmo MOEA se crean asignando puntos clave sobre la posición y forma de los elementos en el mapa, como se muestra en el Alg. 1. El río es creado estableciendo un conjunto de puntos de control a lo largo de la diagonal del mapa, los cuales son perturbados para su uso al momento de desplegar visualmente el río.

Para poder crear las líneas y bases, se establecen 5 puntos clave, donde los primeros 3 representan los puntos donde convergen las líneas en la diagonal del mapa, mientras que los 2 últimos definen sus inicios y la ubicación de cada base. Adicionalmente el tamaño de las bases es dictado por una variable calculada aleatoriamente que define su radio.

Por último, se crean 4 junglas donde a cada una se le asigna una semilla y un conjunto de campamentos aleatorios. La semilla es utilizada al momento del despliegue gráfico del mapa en la función fractal que dibuja las junglas.

Habiendo creado un conjunto de candidatos inicial, el algoritmo procede a calcular la aptitud que posee cada uno. En este caso, 2 funciones de aptitud son empleadas para determinar la viabilidad de un candidato.

1. Distancia de jungla: Es una evaluación de 3 pasos donde se determina la distancia necesaria para recorrer 3 caminos por cada sector de jungla: línea lateral a río, río a línea media, línea media a línea lateral (ver Alg. 2). Esta función determina si la jungla ofrece los 3 caminos necesarios para hacer uso de la misma adecuadamente.

2. Diferencia de campamentos por jungla: Determina la diferencia que hay entre la cantidad de campamentos de las junglas de un equipo, con las del otro (ver Alg. 3). Cada campamento presenta una oportunidad para los jugadores para incrementar sus habilidades, es por esto que la diferencia en la cantidad de campamentos de cada equipo no debe exceder cierto límite para mantener el balance entre ellos.

Al finalizar la evaluación, los resultados de ambas funciones de aptitud se suman, obteniendo un resultado que se encuentra dentro del rango [0,2]. Es en esta suma donde se puede apreciar el algoritmo de MOEA en funcionamiento (ver Alg. 4).

El algoritmo generará una cantidad fija de generaciones, donde por cada generación nueva existe la posibilidad de agregar nuevos candidatos creados aleatoriamente, para que las nuevas generaciones no estén conformadas únicamente por características presentes en los individuos de la anterior.

Una vez terminado, el algoritmo escoge al individuo con el mayor valor de aptitud y se comienza el proceso de *Dibujo*, donde se interpretan las características del mapa para poder desplegarlo gráficamente.

Inicialmente se comienza con un mapa vacío, en el cual se interpretara la información del río para poder colocarlo a lo largo del mapa como se puede apreciar en la Fig. 9. El conjunto de puntos que conforman el río es utilizado en una curva de Bézier para poder dibujarlo.

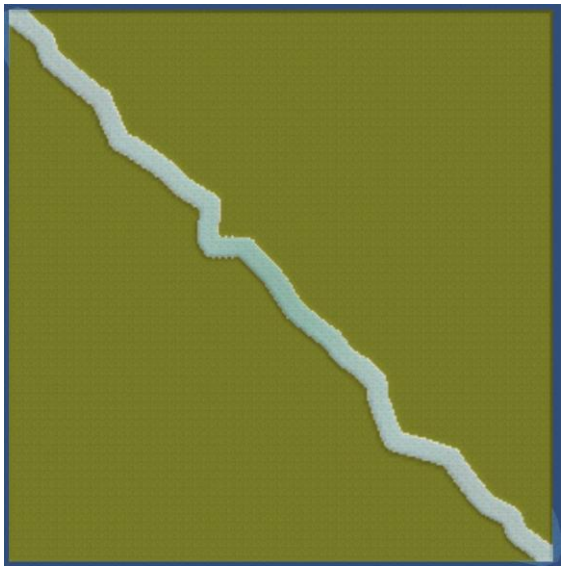

Figura 9. Río generado como primer paso para crear el mapa.

Luego se interpretan las intersecciones de las líneas y las posiciones de cada base y se colocan todos los componentes necesarios en el mapa para representar estos elementos (Fig. 10).

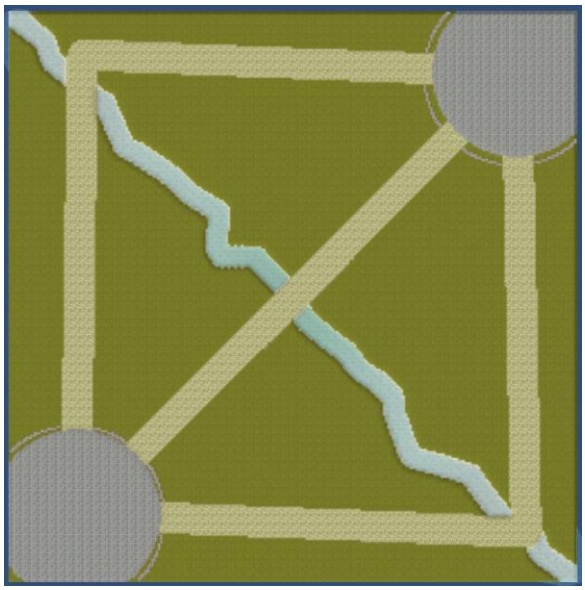

Figura 10. Líneas y bases generadas en el segundo paso durante la creación del mapa.

Por último, se utiliza la información de cada jungla para definir cómo deben ser dibujadas en el mapa (Fig. 11). La semilla es utilizada en una función fractal para poder dibujar sus caminos y se colocan los campamentos en posiciones adyacentes a los mismos.

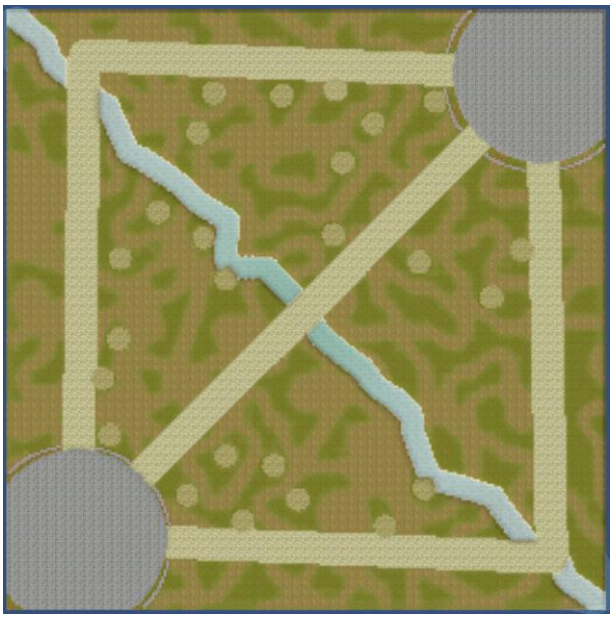

Figura 11. Junglas generadas como tercer paso en la creación del mapa.

Creando el mapa de esta manera, se asegura que el mismo sea diferente cada vez que se crea. Si bien las líneas y las bases no alteran lo suficiente su forma como para considerarlas un cambio en la jugabilidad del mapa, el porcentaje cubierto por las junglas y el río es lo suficientemente grande como para afirmar que la mayoría del mapa se ha generado procedimentalmente y que estos caminos dictaran el paso del juego.

Habiendo creado un mapa con todas las características necesarias, es momento de incluir a los personajes que se desenvolverán en dicho espacio. En el siguiente capítulo se explicará detalladamente cómo funciona el sistema de adaptación de personajes y como el mismo es capaz de adaptar las características de un personaje al estilo de juego del jugador.

Capítulo 4 – Sistema de Adaptación de Personajes (SAP)

Previo a explicar en qué consiste el SAP implementado en Casmage, es necesario describir como están compuestos los personajes de un juego MOBA tradicional.

4.1 – Personajes en un MOBA

Los personajes (ver la Fig. 12) siguen el modelo tradicional de un personaje de un juego de rol, donde los mismos poseen un conjunto de atributos y un conjunto de habilidades que es propio de cada uno. Los atributos varían entre distintos videojuegos, pero tienen la característica en común de que alteran de alguna manera las capacidades del personaje. Las habilidades por otro lado, son acciones únicas para un personaje, que le permiten interactuar de manera especial con su entorno. Estos atributos y habilidades son lo que los jugadores utilizan como herramientas durante el juego, para poder cumplir sus objetivos.

Figura 12. Ejemplo de personajes utilizados en el videojuego Casmage.

Durante el juego, los jugadores obtienen puntos de experiencia venciendo oponentes. Estos puntos de experiencia se acumulan y al momento de alcanzar cierto umbral, se le da al jugador un incremento en su nivel. El nivel es el indicador que determina que tan fuerte es un personaje durante el juego.

Los atributos de un personaje establecen características de este. Por ejemplo, la cantidad de puntos de vida del personaje o la cantidad de daño que el mismo puede resistir. Estos atributos pueden ser incrementados tanto al ganar un nivel como comprando artefactos para el personaje.

Las habilidades de un personaje conforman el conjunto de acciones que el mismo puede realizar, y que determinan la manera en la que el mismo debe ser jugado para poder ser efectivo. La eficacia de las habilidades incrementa únicamente cuando el personaje gana un nivel.

4.2 – Adaptación de personajes

La adaptación de personajes (SAP) es un sistema de PCG intervenido por el usuario, que se encarga de modificar activamente los atributos y habilidades de un personaje, según el estilo de juego del jugador. Utiliza como métrica, la eficiencia del jugador durante el combate. Por lo cual, un jugador que utiliza eficientemente sus habilidades durante un combate, es recompensado por esto.

Debido a que el sistema de incremento de atributos y mejora de habilidades es reemplazado por el SAP, se elimina la necesidad de tener un contador de niveles, lo cual supone un cambio de paradigma en el modelo tradicional del personaje de rol, esto se debe a que la progresión de un personaje deja de medirse de manera discreta y se asemeja más a un modelo continuo. Por esto, el jugador ya no decide cual habilidad mejorar al subir de nivel, sino que las habilidades que haya utilizado, mejoraran para corresponderse a su estilo de juego.

Ejemplo:

> La habilidad "Bola de Fuego", es una habilidad que le permite al jugador disparar una bola de fuego hacia un oponente. El impacto de esta bola hace daño al oponente y genera una explosión que daña a todos los oponentes cercanos a la zona del impacto. La cantidad de daño que hace el proyectil, la explosión y su radio están directamente relacionados con el valor del atributo *Ataque Mágico* que el personaje posee (Fig. 13).

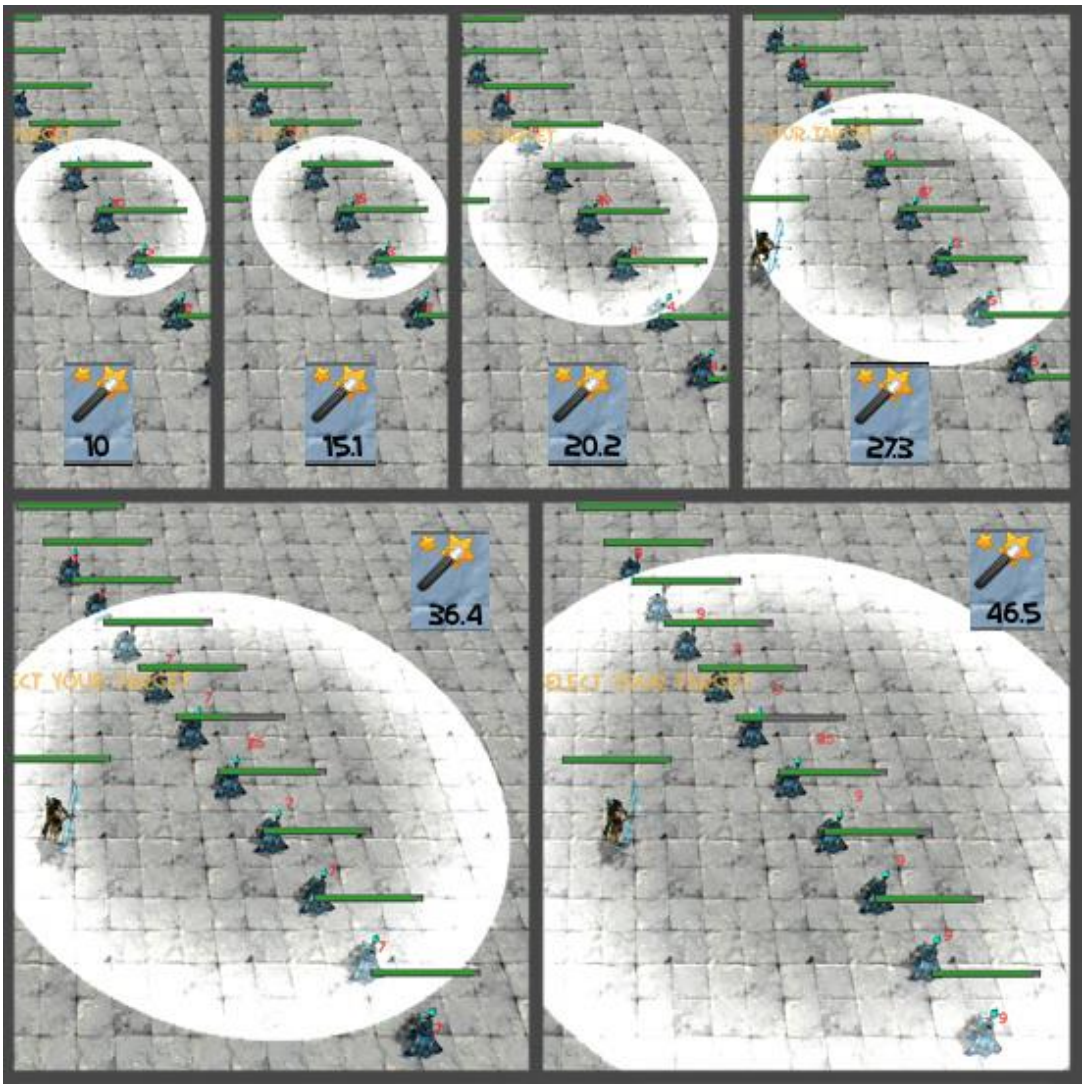

Figura 13. Relación entre el radio de explosión de la Bola de fuego y el atributo Ataque Mágico.

En la Fig. 14 se puede apreciar un diagrama de flujo que muestra el proceso que lleva a cabo el SAP cada vez que se utiliza esta habilidad.

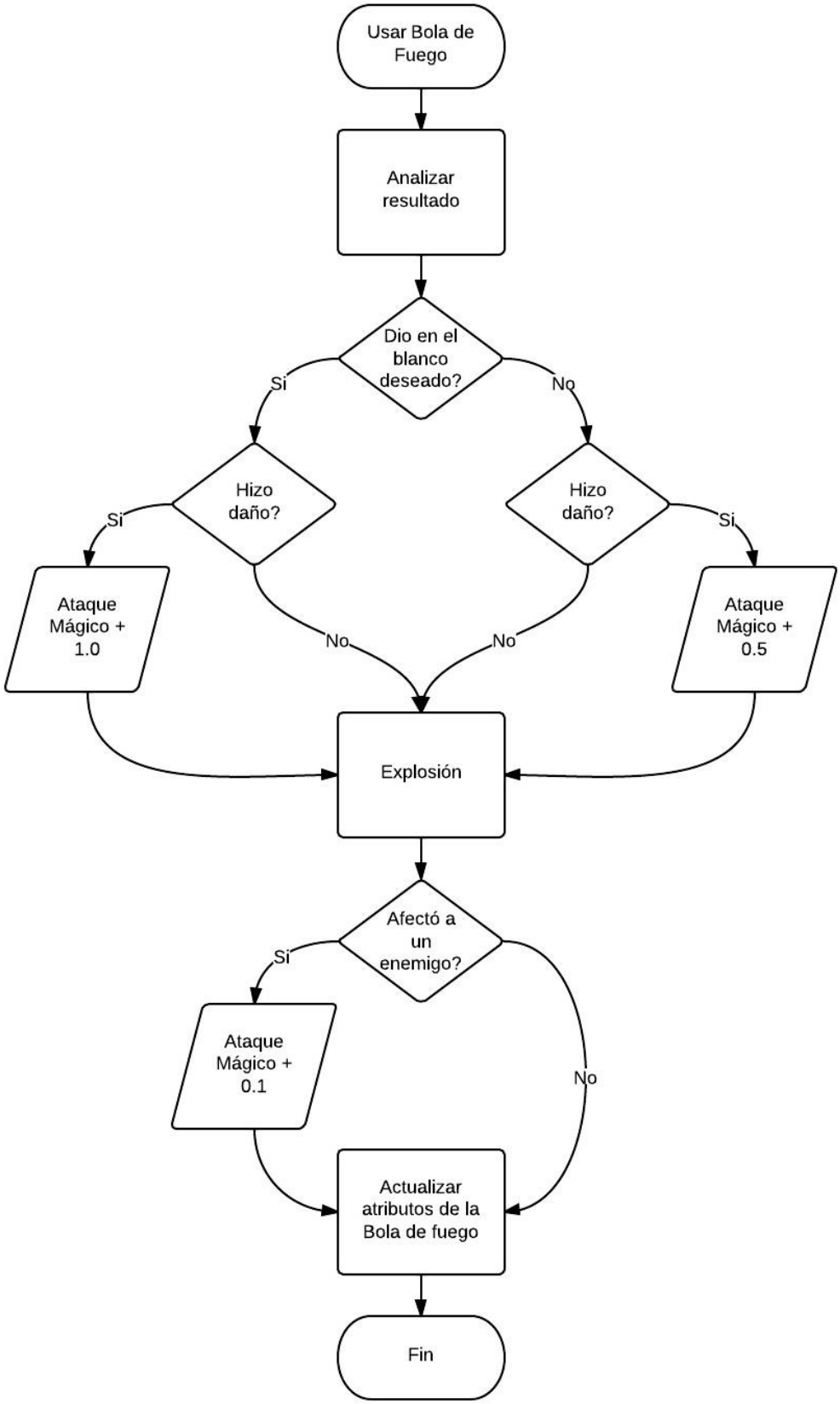

Figura 14. Diagrama de flujo del SAP para la habilidad Bola de Fuego.

Cuando un jugador hace uso de su habilidad de Bola de fuego, se dispara un proyectil desde la ubicación de su personaje que va dirigido al oponente seleccionado por el jugador. Una vez que este proyectil colisiona con algún oponente, el sistema de SAP analiza el resultado de esta acción y determina que tan eficiente fue el jugador. Si el oponente contra el cual colisionó el proyectil no es el elegido por el jugador, se le da una bonificación de 0.5 al ataque mágico del jugador, de lo contrario, la bonificación es de 1.

Luego de haber colisionado contra algún oponente, se crea una explosión en el lugar de colisión que daña a todos los oponentes que estén dentro de su radio. Si esta explosión daña a algún oponente entonces se le da una bonificación de 0.1 al jugador por cada oponente dañado.

Se puede apreciar como el sistema ofrece una mayor recompensa para los jugadores que hacen un uso eficiente de las habilidades de su personaje.

4.3 – Sistema de Red

El sistema de red implementado consiste en un modelo cliente-servidor, donde un servidor inicia el juego, y distintos clientes se conectan al mismo a través de la red.

Para poder implementar un modelo cliente-servidor, básicamente existen 2 opciones:

1. Servidor autoritativo (Fig. 16): El servidor realiza todos los cómputos necesarios en el juego y envía la información a los clientes. Los clientes solo replican la información del servidor.

2. Servidor no autoritativo: Los clientes procesan información del juego localmente y la envían al servidor. El servidor actúa como un canal de comunicación entre los jugadores.

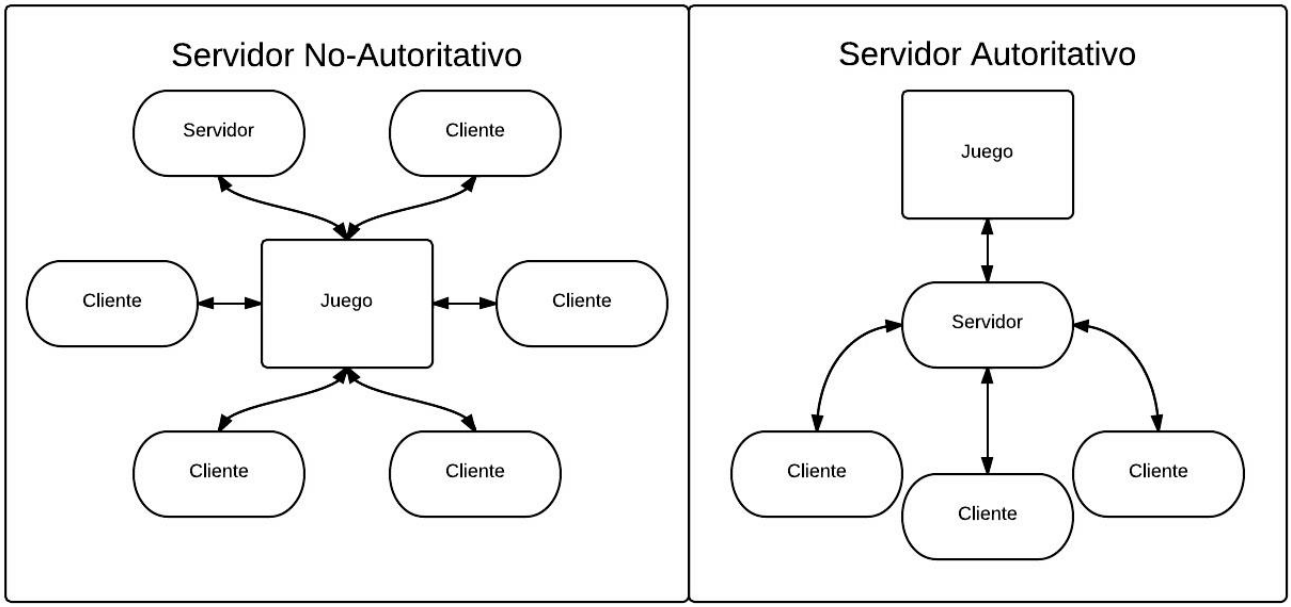

Figura 15. Comparación entre un servidor no-autoritativo y autoritativo.

La implementación de un servidor no-autoritativo es la más sencilla de las 2 soluciones planteadas. Sin embargo, es la que posee más fallas, debido a que al permitir a los clientes tomar decisiones sobre lo que ocurre en el juego (Fig. 15), un jugador que haga uso de una aplicación capaz de modificar los paquetes de información enviados hacia el servidor puede fácilmente cambiar algún valor para obtener ventaja sobre el resto de los jugadores.

Es debido a esto que un servidor autoritativo encaja mejor en un ambiente donde todo debe ser controlado por una misma entidad para garantizar completa transparencia de los resultados. Además de esto, la utilización de un servidor autoritativo alivia la carga de procesamiento de los clientes debido a que estos solo deben realizar peticiones al servidor y esperar respuesta del mismo.

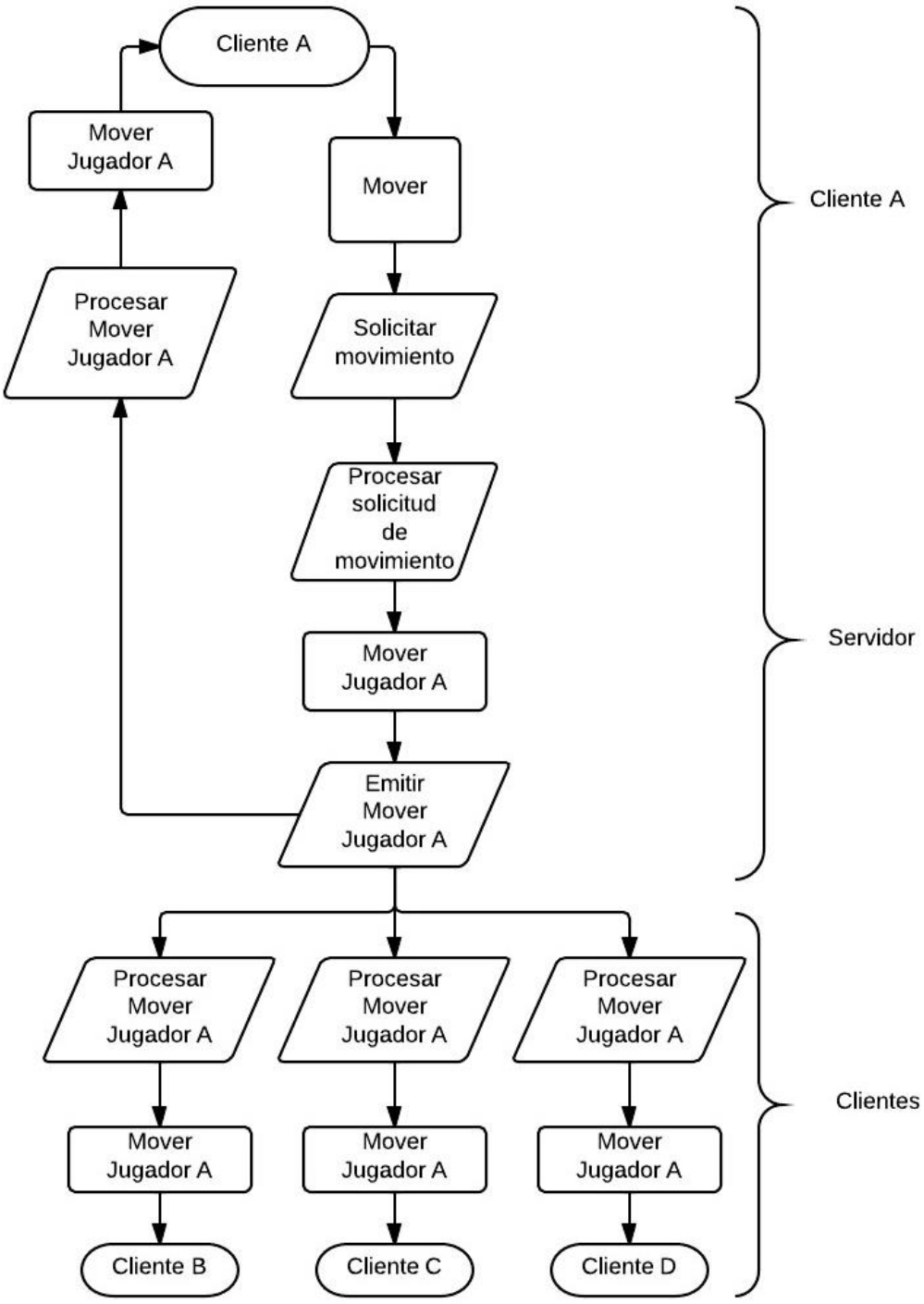

Figura 16. Diagrama de flujo para mover un personaje en un servidor autoritativo.

Generalmente se opta por implementar un servidor dedicado, el cual es una aplicación diferente al juego. Esta aplicación de servidor tiene como función manejar los mensajes y eventos que ocurren en el juego, pero tiene la ventaja de que no necesita procesar ninguna información adicional que no es pertinente al servidor (por ejemplo el despliegue de información visual de modelos 3D), lo cual ofrece un gran incremento a las capacidades de procesamiento del servidor.

El servidor utilizado en Casmage funciona tanto como cliente como servidor, es decir que cuando se crea el servidor y comienza un juego, ya existe un cliente conectado por defecto.

Para que un videojuego basado en MOBA funcione de manera adecuada existen diversos mensajes de red que deben enviarse constantemente para asegurar la fidelidad de la información que recibe cada cliente conectado. Además de esto, deben tomarse

medidas para mantener el estado del juego en el cliente tan actualizado como sea posible sin sobrecargar el tráfico de red. Los siguientes son algunos métodos utilizados para resolver problemas de red en Casmage:

- Si un cliente no recibe información sobre que otro personaje está moviéndose, entonces el mismo tendría información errada en su juego sobre lo que realmente está sucediendo. Para reducir el impacto de este problema, se envía en un intervalo fijo de tiempo la posición de todos los personajes del juego a todos los clientes. De esta manera, aun si un cliente se encuentra en el caso descrito, puede tener información más fiable sobre donde están los personajes en el mapa.

- Todos los clientes reciben una actualización sobre el estado de una entidad únicamente cuando el mismo ha cambiado. Es preferible no enviar información que los clientes ya poseen a manera de reducir el tráfico en la red.

- Cuando un jugador utiliza una Bola de fuego, no todos los eventos que esta habilidad dispara son manejados por mensajes de red, esto es debido a que el comportamiento de esta habilidad permite que el cliente replique exactamente lo que está ocurriendo sin necesidad de recibir esta información del servidor. Es importante recordar que el replicar acciones en un cliente tiene como finalidad reducir el tráfico en la red.

- Para poder permitirle a un cliente visualizar el mapa que ha generado el servidor, no es viable enviar la información de cada celda en la cuadricula que se utiliza para desplegar visualmente el mapa, ya que esto representa una gran cantidad de información, que colapsaría rápidamente el canal por el cual se transmiten los mensajes de red. Se optó por enviar la información que se utiliza para poder dibujar el mapa, información que es descrita en el Capítulo 3.2.3. De esta manera, con un volumen de información mucho menor, se logra replicar completamente en el cliente el mapa generado por el servidor.

Habiendo descrito el funcionamiento del SAP, se puede apreciar como los personajes se ajustan a las acciones realizadas por el jugador para corresponderse con su estilo de juego. A su vez, se analiza la implementación del sistema de red, el cual se encarga de permitir la interacción de varios jugadores en un mismo juego.

En el siguiente capítulo se colocan figuras técnicas sobre la implementación de aspectos anteriormente explicados, como el funcionamiento del algoritmo MOEA para seleccionar candidatos, o la composición de las habilidades para que las mismas puedan ser utilizadas por todos los personajes involucrados en el juego.

Capítulo 5 – Diseño e implementación

En este capítulo se muestran diagramas de clases que muestran la composición del sistema encargado de generar procedimentalmente los mapas, así como también la jerarquía utilizada para la utilización de las distintas habilidades.

Además, se describen algoritmos detallados sobre la implementación de los candidatos y como son utilizados por el algoritmo de MOEA para determinar cuál es el más apto.

5.1 – Diagramas de clases

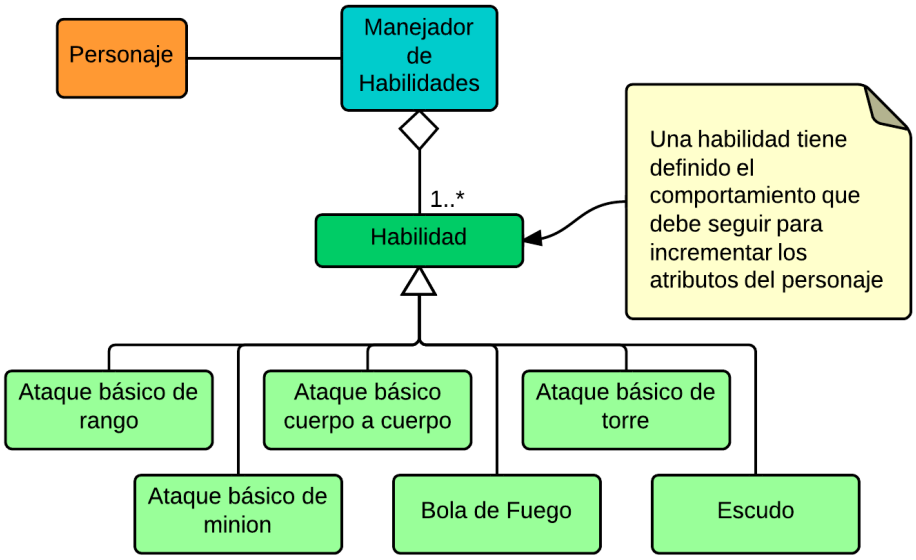

Figura 17. Diagrama de clases sobre las habilidades de un personaje.

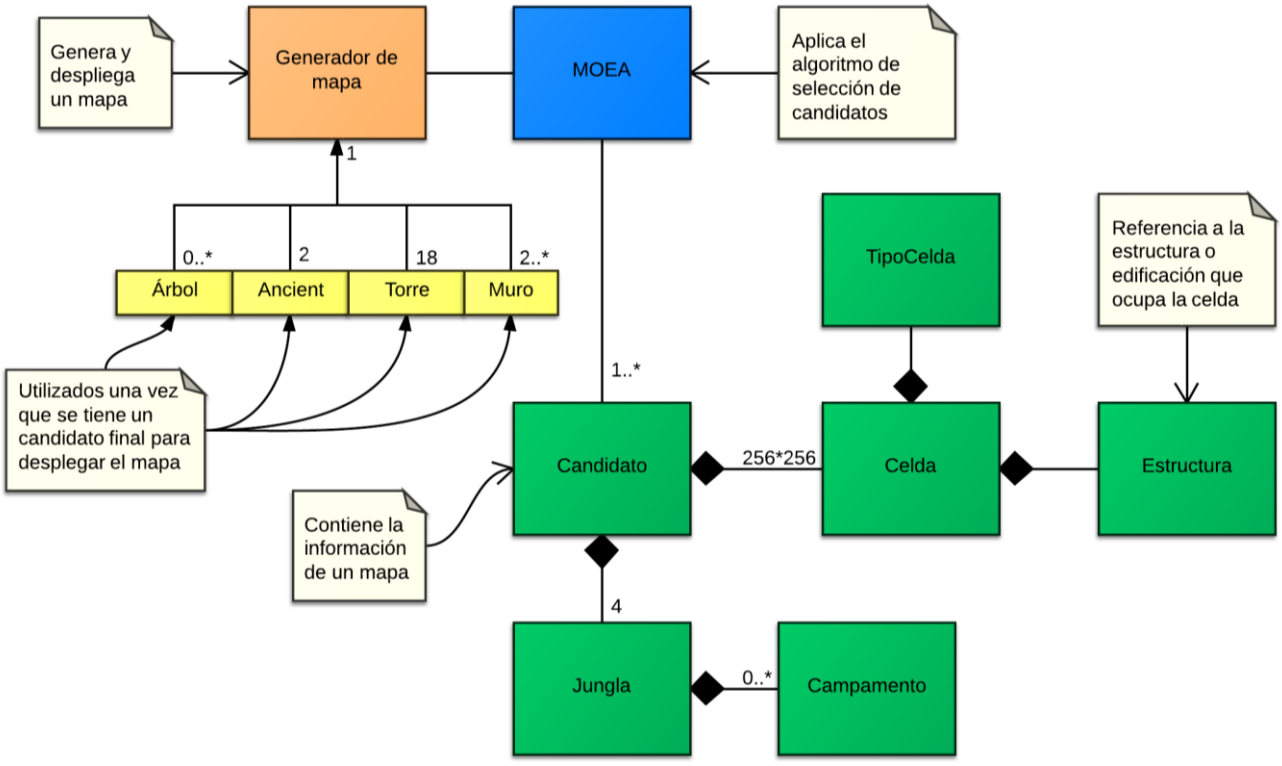

Figura 18. Diagrama de clases sobre la generación procedimental de un mapa.

En la Fig. 17 se puede observar la estructura bajo la cual se diseñaron las habilidades utilizadas por los jugadores. El manejador de habilidades es el que se encarga de controlar los momentos en los que un personaje puede utilizar una habilidad.

Cada habilidad implementada posee información sobre cómo debe modificar los atributos del personaje una vez se dan los eventos adecuados.

El algoritmo de MOEA utilizado para crear el mapa utiliza la estructura descrita en la Fig. 18. Se crea inicialmente un conjunto de candidatos, donde cada candidato contiene la información necesaria para crear un mapa. El algoritmo de MOEA entonces procesa la información de los candidatos e itera tantas veces como haya solicitado el Generador de mapa, el cual se encarga de desplegar la información del mapa del mejor candidato elegido.

5.2 – Algoritmos

```
class Candidato{
    Celda[][] mapa = new Celda[256][256];
    Vector2[] ptosDeControl = new Vector2[32];
    Vector2[] intersec = new Vector2[3];
    Vector2[] bases = new Vector2[2];
    float basesRadio = 0.0f;
    Vector2[] torres = new Vector2[18];
    Jungla[] junglas = new Jungla[4];

    public Candidato(){
        //Río
        ptosDeControl = CrearPtosDeRio();
        //Líneas
        //Usualmente se refiere a las lineas por su position en ingles
        //Top, Mid y Bot (Top, Middle y Bottom)
        intersec[0] = GenerarInterseccionTop();
        intersec[1] = GenerarInterseccionMid();
        intersec[2] = GenerarInterseccionBot();
        //Bases
        //Se refiere a la base en la esquina inferior izquierda como base azul
        //Se refiere a la base en la esquina superior derecha como base roja
        basesRadio = GenerarRadio();
        bases[0] = UbicarBaseAzul(basesRadio);
        bases[1] = UbicarBaseRoja(basesRadio);
        //Torres
        //Las primeras 9 torres corresponden al equipo azul
        for( int equipo=0; equipo<2;equipo++){
            for(int linea=0; linea<3; linea++){
                torres[(equipo*9)+(linea*3)] = UbicarTorre(equipo, 0);
                torres[(equipo*9)+(linea*3)+1] = UbicarTorre(equipo, 1);
                torres[(equipo*9)+(linea*3)+2] = UbicarTorre(equipo, 2);
            }
        }
        //Junglas
        for(int cuadrante=0; cuadrante<4; cuadrante++){
            junglas[i] = new Jungla(cuadrante);
            junglas[i].semilla = Random();
            junglas[i].campamentos = CrearCampamentos();
        }
    }
}
```

Algoritmo 1. Algoritmo que define la creación de un candidato para ser usado por el MOEA.

```
float EvaluarDistanciaDeJungla(Candidato cand){
    float[][] distancias = new float[4][3];
    int topRio, botRio = 0;
    int topRioMid, botRioMid = 1;
    int midTop, midBot = 2;
    for(int sector=0; sector<4; sector++){
        if(sector%2 == 0){
            distancias[sector][topRio] = CalcDist(sector, topRio);
            distancias[sector][topRioMid] = CalcDist(sector, topRioMid);
            distancias[sector][midTop] = CalcDist(sector, midTop);
        }
        else{
            distancias[sector][botRio] = CalcDist(sector, botRio);
            distancias[sector][botRioMid] = CalcDist(sector, botRioMid);
            distancias[sector][midBot] = CalcDist(sector, midBot);
        }
    }
    float[] promedioDist = PromedioDistPorSector(distancias);
    float[] diferencias = new float[2];
    diferencias[0] = Normalizar(Abs(promedioDist[0] - promedioDist[2]));
    diferencias[1] = Normalizar(Abs(promedioDist[1] - promedioDist[3]));

    if(AmbiasDifExcedenLimite(diferencias))
        return 0.0f;
    else if(NoExcedenLimite(diferencias))
        return (diferencias[0]+diferencias[1])/2.0f;
    float difNoExcede;
    if(ExcedeLimite(diferencias[0])) difNoExcede = diferencias[1];
    else difNoExcede = diferencias[0];

    return difNoExcede/4.0f;
}
```

Algoritmo 2. Función de aptitud de distancia de caminos en jungla.

```
float EvaluarDiferenciaDeCampamentos(Candidato cand){
    int[] pesoPorSector = new int[4];
    for(int sector=0; sector<4; sector++)
        pesoPorSector[i] = SumaCampamentos(cand.junglas.campamentos);
    float[] diferencias = new float[2];
    //Diferencias
    diferencias = Normalizar(Abs(pesoPorSector[0]-pesoPorSector(2)));
    diferencias = Normalizar(Abs(pesoPorSector[1]-pesoPorSector[3]));
    return (diferencias[0]+diferencias[1])/2.0f;
}
```

Algoritmo 3. Función de aptitud de diferencia entre campamentos.

```
Candidatos[] candidatos = new Candidatos[numeroDeCandidatos];
float[] aptitudCandidatos = new float[numeroDeCandidatos];
int mejorCandidato;

void MejorCandidato(){
    for(int i=0; i<numeroDeCandidatos; i++){
        aptitudCandidatos[i] = 0.0f;
        //Ambas funciones retornan un valor en el rango [0,1]
        aptitudCandidatos[i] += EvaluarDistanciaDeJungla(candidatos[i]);
        aptitudCandidatos[i] += EvaluarDiferenciaDeCampamentos(candidatos[i]);
    }
    mejorCandidato = Max(aptitudCandidatos);
}
```

Algoritmo 4. Elección del mejor candidato basado en su aptitud.

El Alg. 1 describe como es inicializado un candidato para ser utilizado en el algoritmo de MOEA. Inicialmente se crean los puntos que conforman el río, luego de esto se definen las intersecciones de las líneas en el mapa, las cuales se posicionan sobre el río. Se establece entonces el radio de las bases y la ubicación de las mismas, para luego definir la posición de las torres y posteriormente la creación de los 4 sectores de junglas. Una vez establecida toda esta información es posible desplegar un mapa completo utilizando este candidato.

La función descrita por el Alg. 2 define el proceso de evaluación de los caminos que constituyen una jungla. Se evalúan 3 caminos por cada jungla: camino desde la línea lateral (línea superior o inferior) hasta el río, camino desde el río hasta la línea media y camino desde la línea media hasta la línea lateral. Una vez medidos estos 3 caminos, se promedian estos valores y se procede a realizar una comparación entre los 4 valores obtenidos, los cuales se corresponden a los 4 sectores de junglas que hay en un mapa.

El Alg. 3 evalúa la proporción de junglas que hay en cada equipo con respecto al otro. Esto lo hace analizando la diferencia de sectores equivalentes en cada lado del mapa (diferencia entre sectores superiores y diferencia entre sectores inferiores), luego de obtener esta diferencia, se normaliza cada diferencia (se utiliza un valor máximo de diferencia para la normalización) para poder obtener un valor máximo de 2, el cual es reducido a la mitad y retornado a la función de MOEA.

El resultado de los Alg. 2 y Alg. 3 es un valor en el rango [0,1] donde 1 representa el maximo valor de aptitud posible a ser obtenido.

El Alg. 4 muestra el procedimiento a seguir para realizar la evaluación del mejor candidato. Se cicla entre todos los candidatos de una generación y para cada uno se suman los resultados de las 2 funciones de aptitud a evaluar, descritas en Alg. 2 y Alg. 3. Luego de esto, se busca el índice del candidato con el valor de aptitud más alto y se guarda el mismo para posteriormente desplegar la información de mapa que este guarda.

Los detalles de implementación ofrecidos son vitales al momento de elaborar un sistema capaz de crear un mapa procedimentalmente siguiendo las reglas de diseño establecidas. Además de esto, es necesario que cada habilidad herede sus características de una clase superior que le permita interactuar como es debido con las acciones del jugador y modificar los atributos del personaje de manera adecuada.

En el siguiente capítulo se describen las pruebas realizadas y los resultados obtenidos tras utilizar los sistemas anteriormente descritos.

Capítulo 6 – Pruebas y resultados

Previo a definir las pruebas realizadas junto con sus resultados, es importante establecer los parámetros bajo los cuales se llevaron a cabo estas pruebas, así como los recursos que se han utilizado para poder completar Casmage.

6.1- Ambiente de trabajo

Las pruebas se han realizado bajo un ambiente con las siguientes especificaciones:

- Sistema operativo: Windows 8 – 64 bits.
- Procesador: AMD Phenom II X4 – 3.40 GHz.
- Memoria RAM: 8 GB DDR3.
- Tarjeta gráfica: ATI Radeon 4870 HD.
- Ancho de banda: Download: 4 Mbps / Upload: 768 kbps .

Como herramienta principal para el desarrollo de este trabajo se utilizó Unity (ver Fig. 19) por sobre otras herramientas, debido a la flexibilidad que posee en cuanto al desarrollo de videojuegos y a la familiaridad que se tiene con la utilización de esta herramienta.

Figura 19: Logo de Unity.

Tomando en cuenta que el objetivo principal de una herramienta como Unity es acelerar el proceso de desarrollo de un videojuego, la misma implementa por defecto todas las funcionalidades para desplegar gráficos 3D, así como también manejo de interacciones físicas, entre otras.

6.1.1 – Contenido externo utilizado

Para crear Casmage, se utilizaron diversos recursos externos para complementar el proceso de desarrollo, entre estos se encuentran librerías, modelos 3D, texturas y otros recursos, los cuales son nombrados a continuación:

- Accidental Noise Library[17]: Esta librería permite la utilización de algoritmos fractales y fue utilizada en la creación de las junglas. Es importante resaltar que debido a que esta librería fue creada para C++, debió utilizarse una versión traducida a C# para poder hacer uso de la misma en Unity.
- Personajes y minions: Los modelos y animaciones utilizados para los personajes y minions son recursos pertenecientes al videojuego League of Legends[18].
- Estructuras y terreno: Los modelos de las estructuras (Ancient y Torres) y las texturas del terreno son recursos pertenecientes a Dota 2.
- Efectos de partículas: Se utilizó el paquete de efectos Simple Particle Pack[19].
- Árboles, muros y rocas: Recursos creados por Anguel Roumenov[20].

El conjunto de pruebas realizadas se divide en 3 categorías:

- Mapas generados.
- Estadísticas de jugadores.
- Estadísticas variadas.

En la primera se hará una comparación entre los mapas generados en Casmage y se determinara la proporción en la cual se obtuvieron mapas clasificados como no viables. En la siguiente categoría se estudia cómo se desenvuelven los jugadores en un juego y como cambian los atributos de sus personajes sobre el tiempo. Por último, se presentan las estadísticas relacionadas con el desempeño del juego a nivel de red y procesamiento.

6.2 – Mapas generados

Debido a que el algoritmo de MOEA utilizado para crear los mapas no impide la aparición de mapas que incumplan ciertas de las reglas establecidas para generar los mismos, se generaron 20 mapas de los cuales 2 no son viables para su uso en un juego. La Fig. 13 muestra los 20 mapas utilizados para realizar esta evaluación. Cada uno es el resultado de aplicar el algoritmo de MOEA a un conjunto de candidatos.

Los puntos amarillos representan los campamentos en las junglas, los puntos rojos representan también campamentos en las junglas pero se utilizan para denotar que existe un exceso de campamentos de ese lado del mapa.

Las zonas de color morado señalan que existe poco terreno donde un jugador puede cubrirse estando cerca del río. Idealmente se espera que un jugador tenga un lugar donde cubrirse de ataques provenientes del río.

Aquellos mapas que no tienen puntos rojos ni zonas moradas son mapas viables para ser jugados. Sin embargo, aquellos que contengan tanto puntos rojos como morados (mapas número 10 y 18) son los que pueden traer mayores problemas a la hora de permitir un juego balanceado entre equipos.

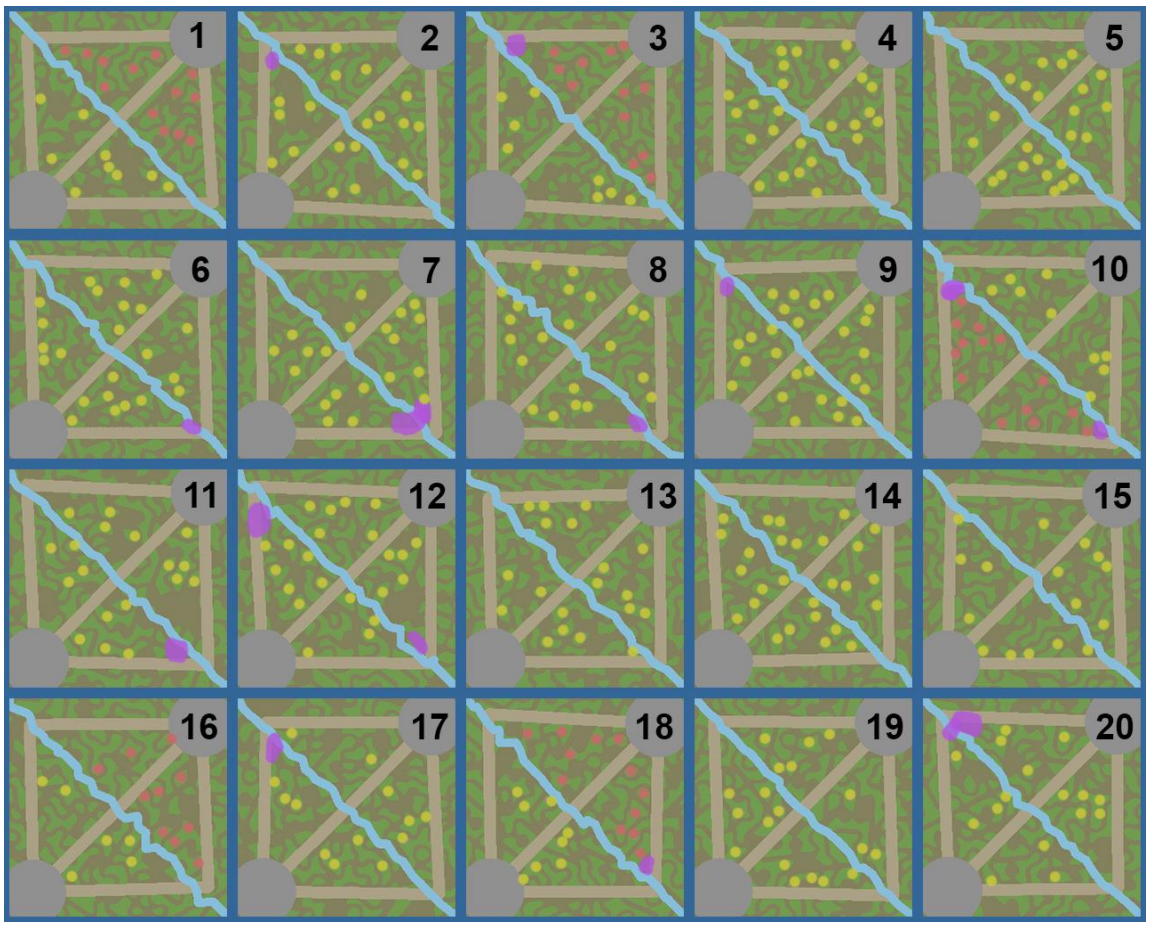

Figura 20.Comparación entre 20 mapas generados procedimentalmente.

6.3 – Estadísticas de jugadores

Es importante evaluar como distribuye su tiempo un jugador durante el juego. Para ello se realizó un conjunto de pruebas para obtener información sobre las acciones de un jugador dentro de una ventana de tiempo.

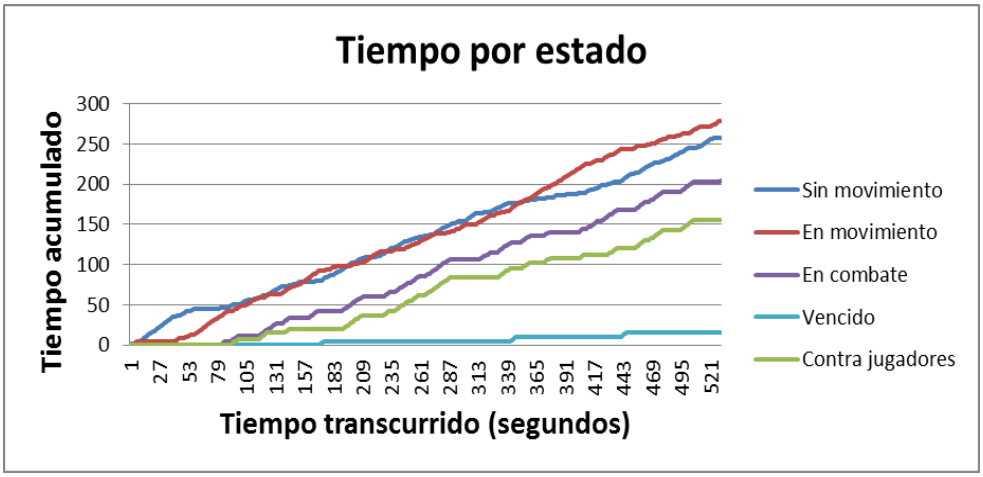

Figura 21. Estados del jugador sobre tiempo.

La Fig. 21 muestra como un jugador distribuye su tiempo en cuanto a movimiento, tiempo en combate, tiempo en combate contra otro jugador y tiempo vencido. El eje horizontal representa el tiempo transcurrido desde que el jugador comenzó a jugar.

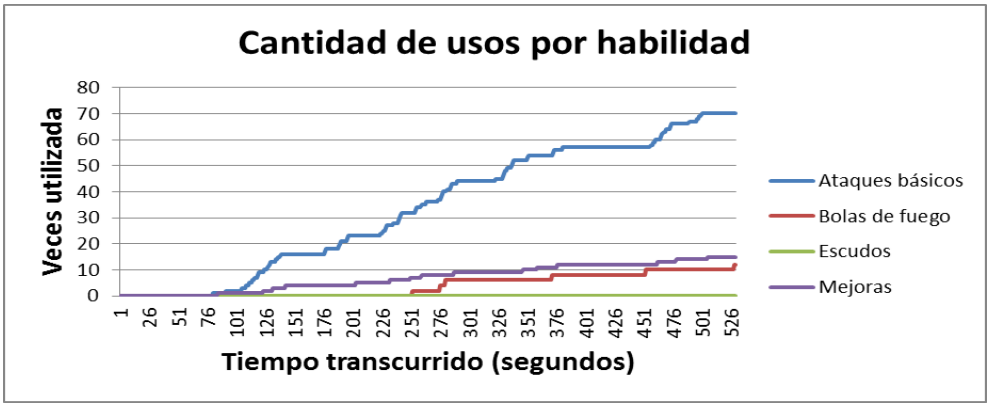

Figura 22. Relación entre la utilización de las habilidades y las mejoras de los atributos.

La Fig. 22 muestra la utilización de las habilidades del personaje sobre el tiempo. Cada vez que ocurre un incremento en alguno de los valores indica que el jugador ha entrado en combate. El eje horizontal representa el tiempo transcurrido desde que el jugador comenzó a jugar.

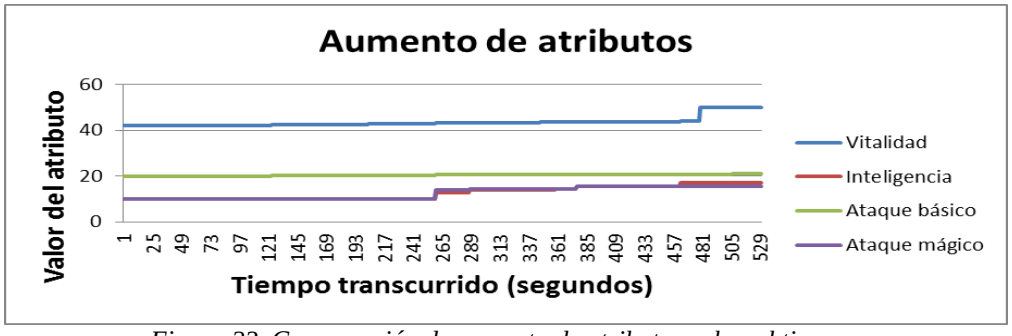

Figura 23. Comparación de aumento de atributos sobre el tiempo.

En Fig. 23, se puede apreciar cómo cambiar los atributos de un personaje con respecto al tiempo, y como se relacionan estos mismos al combate. El eje horizontal representa el tiempo transcurrido desde que el jugador comenzó a jugar.

Al observar la figura 22 y la figura 23 se puede apreciar la relación que existe entre la utilización de habilidades y la modificación de los atributos de un personaje conforme pasa el tiempo. Es posible apreciar como un jugador está obligado a entrar y salir constantemente de combate para poder mejorar las características de su personaje y tener mayores posibilidades de ganar en un enfrentamiento

6.4– Estadísticas variadas

En la solución planteada, se consideran muchas variables existentes durante la ejecución de un videojuego en línea. Factores como el rendimiento del hardware gráfico o valores relacionados con la transmisión de paquetes por la red desde el servidor al cliente y viceversa, son fundamentales.

En la Fig. 28 se aprecia la información relacionada con el consumo de recursos de la aplicación actuando como servidor. Debido a que esta ventana de estadísticas es una funcionalidad del editor de Unity, el consumo de recursos mostrado está por debajo de lo que comúnmente se utilizará, pues existe cierta carga extra al procesamiento para soportar las funciones del editor.

Figura 24. Información estadística del procesamiento gráfico y de la red en tiempo real.

Debido a la gran cantidad de información que debe manejar un servidor autoritativo no dedicado, al utilizar el juego como servidor se presenta una gran carga al procesamiento, lo cual reduce considerablemente los cuadros por segundo (fps) que puede procesar el juego. Sin embargo, este comportamiento es común en este tipo de servidores y no influye en gran manera el desempeño del juego para los clientes.

Se puede observar que la cantidad de memoria de video utilizada varía entre los 6 MB y 18 MB, lo cual supone un bajo consumo de memoria de video. Igualmente, es posible mostrar la información del tráfico de la red. Por cada jugador conectado al servidor, este envía entre 10 kbps y 20 kbps según el estado actual del cliente, y recibe del mismo menos de 1 kbps de tráfico. Este valor se multiplica por la cantidad de clientes conectados al servidor. Por ejemplo, para 4 clientes conectados a un servidor, se espera que el servidor tenga un tráfico de salida de 80 kbps como máximo.

Se realizaron pruebas de juego con 3 clientes conectados al servidor, lo cual supuso un tráfico de red para el servidor que se ubicaba alrededor de los 60 kbps en subida y 3 kbps de bajada (información enviada por los clientes al servidor).

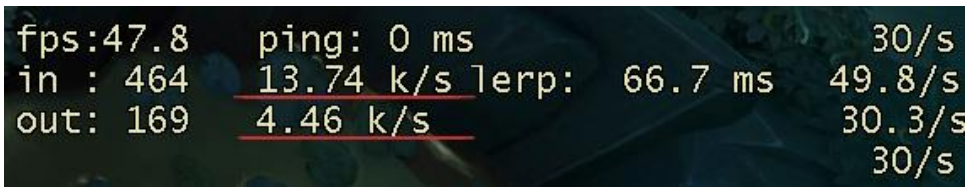

Figura 25. Información sobre el uso de recursos en el videojuego Dota 2.

Habiendo realizado pruebas en el videojuego Dota 2 sobre el tráfico de red (ver Fig. 25), se observó que la cantidad de información enviada al cliente se encuentra entre 8 kbps y 16 kbps. Tomando en cuenta este rango de valores, se puede determinar que el manejo de los mensajes de red en Casmage está por encima de un MOBA común y por ende es posible aplicar optimizaciones para reducir el tráfico de salida del servidor.

Capítulo 7 – Conclusiones y Trabajos Futuros

Los videojuegos del género MOBA tienen la característica de que todo su contenido permanece estático. El mapa en el que se juega es siempre el mismo y las capacidades y estilo de juego de un personaje están realmente limitadas debido a la rigidez de sus propiedades y habilidades.

Se desarrolló entonces un videojuego del género MOBA, el cual utiliza activamente algoritmos de PCG para crear el mapa donde se desenvuelven los jugadores de manera procedimental, así como también un sistema capaz de modificar los atributos de los personajes utilizando como método de entrada las acciones del jugador.

Los aspectos en la creación del mapa se enfocan en preservar las normativas de diseño establecidas en Dota, lo cual garantiza que este mapa cumplirá adecuadamente su función dentro del juego, con el valor agregado de la aleatoriedad producida por el algoritmo de PCG que lo generó. Sin embargo, debido a que se utiliza un algoritmo genético para seleccionar el mapa, siempre existe la posibilidad de obtener un mapa que resulte no ser viable de alguna manera, lo cual puede llegar a producir un juego frustrante para un equipo que quede en desventaja por esto.

Una vez creado el mapa, los jugadores son capaces de elegir personajes que estarán modificando sus características activamente según las decisiones tomadas por el jugador. El cambio de paradigma en el funcionamiento de los personajes creado por el SAP, permite la utilización de un modelo donde cualquier tipo de evento es capaz de modificar los aspectos de un personaje, lo cual ofrece una flexibilidad sin precedentes en un MOBA. Es importante resaltar el hecho de que eliminar un indicador tan importante como el nivel del personaje, implica la necesidad de diseñar un nuevo método para darle a saber a un jugador que tan fuerte es un personaje solo con verlo, pues es necesario tener la capacidad de determinar esto antes de entrar en combate.

Para el manejo de los mensajes de red, se implementó un sistema que comúnmente se encuentra en este tipo de juegos. Se utilizó un esquema de cliente-servidor, donde el servidor actúa como cliente al momento de iniciar la conexión debido a que se trata de un servidor autoritativo no dedicado. Sin embargo, habiendo comparado el tráfico de red generado por el servidor de Casmage contra el tráfico generado por un servidor de Dota 2, se puede afirmar que existe un exceso de información que está siendo transmitida en Casmage hacia el cliente.

En cuanto a los resultados se refiere, es posible apreciar la factibilidad que presentan los algoritmos de PCG en un videojuego que, desde sus inicios, ha estado atado a la utilización de elementos estáticos. Aun cuando la utilización de estos algoritmos trae consigo un incremento en el contenido que puede ofrecerse al usuario, es necesario resaltar el hecho de que diseñar estos algoritmos presenta un problema de diseño complejo debido a la cantidad de variables a tomar en cuenta.

Como investigaciones futuras, se propone hacer mayor énfasis en el diseño del SAP para poder implementar mejores indicadores sobre las capacidades de un personaje en un momento dado. Se planea crear un sistema de equipamientos donde cada pieza se corresponda con una habilidad en posesión del personaje, de esta manera, es posible indicar visualmente que tan desarrollada esta una habilidad y de esta forma se compensa la falta de un indicador de nivel.

En lo que a la creación del mapa se refiere, en vista de que los únicos elementos que no son alterados lo suficiente son las líneas y las bases, se propone aumentar la cantidad de información procesada por el algoritmo utilizado para crear mapas, para que el mismo incluya la posibilidad de cambiar la cantidad de líneas y la posición de las bases.

Referencias

[1] Robert L. Devaney, *The Fractal Geometry of the Mandelbrot Set*.
Extraído de: **math.bu.edu/eap/DYSYS/FRACGEOM/node2.html**. Noviembre, 2000.

[2] IDV Incorporated. *SpeedTree*. Extraído de: **www.speedtree.com.** Diciembre, 2002.

[3] Andrew Doull, *The Death of the Level Designer: Procedural Content Generation in Games*. Extraído de: **bit.ly/H6l579**. Enero, 2008.

[4] Blizzard Entertainment. *Diablo*. Extraído de: **us.blizzard.com/es-mx/games/legacy**. Diciembre, 1996.

[5] Derek Yu, *Spelunky*. Extraído de: **spelunkyworld.com/original.html**. Diciembre, 2008.

[6] EA Games. *Spore*. Extraído de: **www.spore.com.** Septiembre, 2008.

[7] Stanford Virtual Worlds Group. *Dryad*. Extraído de: **dryad.stanford.edu/**. Diciembre, 2009.

[8] Stanford Virtual worlds Group. Extraído de: **vladlen.info.**

[9] Togelius J., Yannakakis G. N., Stanley K. O., Browne C. *Search-based Procedural Content Generation*. Reporte Técnico, IT University of CopenHagen, University of Central Florida, Imperial College London.
Extraído de: **blog.itu.dk/mpgg-e2010/files/2010/09/togelius2010searchbased.pdf**. Septiembre, 2010.

[10] Erwin P. Wilhelmus. *The rules behind a procedural death-match map generator*. Julio, 2012

[11] Epic games .*Unreal Development Kit*. **www.unrealengine.com/udk.** Noviembre, 2009.

[12] Hastings. E. J., Guha R. K., Stanley K. O. *Automatic Content Generation in the Galactic Arms Race Video Game*. Diciembre, 2009.

[13] Kolarevic B. *Digital Morphogenesis and Computational Architectures*. Extraído de: **cumincad.scix.net/data/works/att/4077.content.pdf.** Noviembre, 2000.

[14] Valve Corporation. *Dota 2*. Extraído de: **blog.dota2.com**. Julio, 2013.

[15] *Dota*. Extraído de: **www.playdota.com**.

[16] Araujo L., Cervigon C. *Algoritmos Evolutivos, un enfoque práctico*. 2009

[17] Joshua Tipetts. *Accidental Noise Library*. Extraído de: **accidentalnoise.sourceforge.net.** Julio, 2011.

[18] Riot Games. *League of Legends.* Extraído de: **na.leagueoflegends.com.** Octubre, 2009.

[19] Unity Technologies. *Simple Particle Pack.* Extraído de: ***bit.ly/19RSgkO*.** Septiembre, 2013.

[20] Anguel Roumenov. Portafolio disponible en: **sohardtoremember.com.**