PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

FACULTAD DE CIENCIAS E INGENIERÍA

CONSTRUCCIÓN DE UN COMPONENTE DE SOFTWARE PARA LA BÚSQUEDA DEL CAMINO MAS CORTO Y EL CONTROL DE MOVIMIENTO EN UN VIDEOJUEGO DE ESTRATEGIA EN TIEMPO REAL

Tesis para optar por el Título de Ingeniero Informático, que presenta el bachiller:

Carbajal Montesinos, Henry

ASESORES: Mag. Claudia Zapata del Río

Mag. Johan Baldeón Medrano

Resumen

En la actualidad los videojuegos vienen logrando un gran auge en la industria de desarrollo de software, se estima que en el año 2013 alcanzaría ventas superiores a los 66 mil millones de dólares a nivel mundial (REUTERS, 2013); sin embargo, este crecimiento se ve principalmente en los países desarrollados; tal es así, que actualmente cuentan con diversos cursos, carreras universitarias y maestrías en temas relativos a videojuegos (ESA, 2013). En el Perú, sin embargo, la industria del desarrollo de videojuegos es aún incipiente; debido entre otros factores, a la falta conocimientos y experiencia en este rubro (COIDEV, 2012). Por lo que, cuando algunas empresas o grupos locales desean incursionar en esta industria, se ven en dificultades de distinta naturaleza.

Uno de los problemas más comunes, desde el punto de vista de la informática, es el desarrollo de la inteligencia artificial del videojuego; por lo tanto el presente proyecto de fin de carrera, pretende explicar de manera sistemática el desarrollo de los algoritmos y técnicas para la búsqueda del camino más corto, así como del control de movimiento de los personajes, ambos partes de la inteligencia artificial del mismo, de tal manera que los algoritmos desarrollados tengan un tiempo de ejecución y uso de espacio de memoria adecuados. Dicho componente además, ha sido utilizado en el videojuego “1814, La rebelión del Cuzco”, desarrollado por el grupo Avatar de la Pontificia Universidad Católica del Perú.

Dedicatoria

A mis queridos padres y hermanos por su invalorable comprensión y su continuo apoyo e impulso en el logro mis objetivos.

En memoria de mi mejor amigo y compañero Forrest, por siempre acompañarme en toda mi etapa de estudiante.

Agradecimientos

A mis asesores, por su apoyo y consejos durante el desarrollo de este proyecto.

Al grupo Avatar y mis compañeros con los que desarrolle el videojuego "1814, La rebelión del Cusco", por permitirme ser parte de este proyecto.

Tabla de Contenido

Índice de Ilustraciones

Índice de Tablas

1. Capítulo 1

1.1 Introducción

El presente capítulo, presenta la definición de la problemática del proyecto, así como los conceptos básicos que permitirán comprender mejor el contexto del problema, luego se realizará una revisión del estado del arte con distintas soluciones existentes al problema planteado.

El segundo capítulo describe el objetivo del proyecto, así como sus resultados esperados, indicando además, cuáles han sido las herramientas y los métodos utilizados para la realización de cada uno de estos; se describe también la metodología empleada definiendo el alcance y limitaciones del proyecto.
Los capítulos siguientes describen lo obtenido en cada resultado esperado. Empezando con la descripción de las características que se desean implementar en el componente, luego se plantea el diseño de la arquitectura de la solución, y se indican sus principales patrones arquitectónicos. Después se realiza la implementación y comparación de los algoritmos de búsqueda de camino. Finalmente se concluyen los resultados con la implementación del módulo de control de movimiento y las pruebas utilizando un prototipo de un videojuego RTS.

Por Último se muestran las conclusiones a las que se llegó durante el desarrollo del proyecto y se proponen trabajos futuros que podrían complementar a este proyecto.

1.2 Definición de la Problemática

Una de las principales dificultades en el desarrollo de videojuegos es la implementación de la inteligencia artificial, en adelante IA, la cual se emplea en distintos tipos de videojuegos, incluyendo a los del del tipo de estrategia en tiempo real o también conocido como RTS (RABIN, 2005). Dicho tipo de videojuego se caracteriza por simular conflictos y batallas entre distintos agentes inteligentes en un escenario o mapa. Estos agentes interactúan con su entorno así como con el resto de agentes del juego. Por lo tanto, se observa que no es trivial el desplazamiento de un agente de un punto a otro, de tal manera que siempre surgirá una complicación, que consiste en la necesidad de definir un camino a seguir y luego representar la transición sobre ese camino. Por ejemplo, si un agente se encuentra ubicado en la coordenada (3,5) y desea desplazarse a la coordenada (6,10), se debe calcular a través de qué coordenadas conviene realizar la transición, y mostrar dicho movimiento en tiempo real, verificando además, si durante su traslado este debe interactuar con algún otro agente u objeto.

De lo descrito anteriormente, se puede inferir que uno de los requerimientos más importantes en los videojuegos RTS, es que los agentes puedan navegar en su entorno. Dicho problema de navegación se divide en dos subproblemas: la búsqueda de caminos y el movimiento (MILLINGTON, 2009). El problema de búsqueda de caminos se encuentra en diversos tipos de videojuegos e inclusive en otras áreas de la informática, por lo tanto no es un problema exclusivo a los del videojuegos de tipo RTS; y si bien todos buscan resolver lo mismo, es decir cómo hallar una ruta entre dos puntos del espacio, cada problema demanda cubrir distintas consideraciones y requerimientos. En particular, los videojuegos RTS tienen limitaciones muy específicas en cuanto a los tiempos de respuesta de los algoritmos. Esto se debe principalmente a que la propia naturaleza de este tipo de videojuegos demanda mucha fluidez en su ejecución, ya que de lo contrario, el videojuego se volvería lento, lo cual quitaría interés. Debido a esto, es importante considerar ciertos criterios que permitan garantizar la utilización de una estrategia de búsqueda adecuada. Estos son el tiempo de respuesta, espacio de memoria y la optimalidad (RUSELL, 2006).

Por lo antes expresado, en el presente proyecto de fin de carrera se propone implementar parte de la solución de un componente de IA, que proporcione un mecanismo de navegación autónomo en un entorno en dos dimensiones, para lo cual se desarrollará un módulo de búsqueda en grafos, utilizando adaptaciones de los algoritmos de búsqueda A* y HPA*, respetando los criterios de tiempo, espacio de memoria y optimalidad; así como un módulo encargado de la gestión del traslado de los agentes a través del mapa y que permita ejecutar las diversas acciones relacionadas a las interacciones de los personajes tales como: atacar, curar, escapar, patrullar, entre otras; cabe indicar que para esto, se aplicará una toma de decisiones simple, basadas en las reglas y el guion del videojuego.

1.3 Marco Conceptual

A continuación se presenta las definiciones de los principales conceptos a tener en cuenta, los cuales permitirán comprender mejor la problemática, así como la solución que se plantea en los capítulos posteriores.

Un videojuego es un artefacto en un medio visual digital, concebido principalmente como un objeto para el entretenimiento (TAVINOR, 2006); mediante el cual, se crea una representación subjetiva y deliberadamente simplificada de la realidad emocional y es por lo tanto, la fantasía del jugador la clave en hacer el juego psicológicamente real (CRAWFORD, 2003).

Lograr dicha fantasía en el jugador, implica poder conectar al jugador con el videojuego, para lograr dicha conexión, Steve Rabin menciona que un videojuego debe representar un reto para el jugador, con oponentes “inteligentes”, de tal manera que se logre atraer la atención del jugador (RABIN, 2005). Es por esto que, desde la creación de los primeros videojuegos hasta la actualidad, una de las características, en la que más énfasis ponen los desarrolladores, es la inteligencia artificial, pues mediante esta se obtiene la complejidad y realismo necesario. **La inteligencia artificial** consiste en lograr que las computadoras sean capaces de ejecutar las tareas que realizan los humanos o animales (MILLINGTON, 2009), George Luger la describe como una rama de las ciencias de la computación que busca la automatización del comportamiento inteligente, mediante la creación de agentes inteligentes (Luger, 1998).

Es importante diferenciar que la inteligencia artificial académica estudia tópicos desde el punto de vista filosófico, psicológico y de la ingeniería, en el caso de la inteligencia artificial en videojuegos se estudian solo tópicos desde el punto de vista de la ingeniería (MILLINGTON, 2009). Por motivos del alcance del proyecto, para este documento cuando se mencione el concepto de IA, se está haciendo referencia a la inteligencia artificial en videojuegos.

Se define a un **agente inteligente**, como cualquier entidad que puede percibir su entorno a través de sensores y realiza una acción como consecuencia de dicha percepción, otra característica de un agente inteligente, es su autonomía (RUSSELL, 1995). La idea de autonomía se refiere a que la reacción de los agentes no sea en base al conocimiento previo, sino a lo que esta haya percibido en tiempo real, de tal manera que este interprete lo que sucede a su alrededor, para luego tomar una decisión.

Ian Millington propone un modelo de agente inteligente, el cual abarca los distintos componentes que este debe poseer. A continuación, en la Ilustración 1.1 se puede observar dicho modelo.

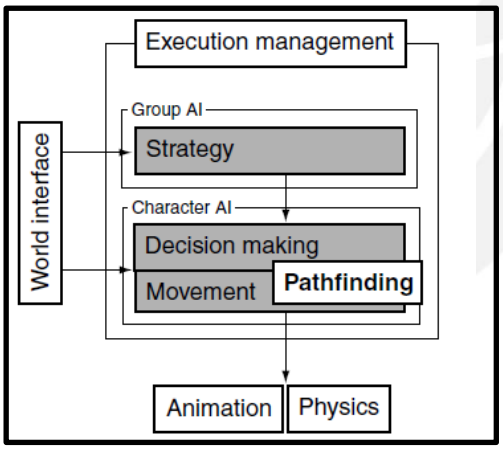

Ilustración 1.1 Modelo de agente inteligente, (Millington, 2009)

Como se observa, existen diversas áreas, sin embargo, no todos los tipos de videojuego requieren del mismo nivel de IA, por ejemplo el ajedrez se enfoca en la parte de la estrategia, mientras que los juegos de deporte requerirán mucho más del movimiento.

Los **videojuegos de estrategia en tiempo real (RTS)**, son aquellos en los que todos los jugadores, incluyendo la computadora compiten de manera simultánea e

ininterrumpida (PEDERSEN, 2003), además, en este tipo de videojuegos los acontecimientos suceden en tiempo real, por lo tanto a diferencia de otros tipos de videojuegos, como los juegos de rol, su nivel de predictibilidad es casi nulo, es por esto que existen infinidad de estados posibles del juego. A continuación se explican aquellos tópicos del modelo referenciado, los cuales serán importantes en la solución que plantea este proyecto (MILLINGTON, 2009):

- **Toma de decisiones:**

La toma de decisiones significa que un agente o personaje pueda determinar qué es lo que debe hacer, es decir un proceso de planificación. Las acciones que realicen dependerán totalmente del comportamiento que tenga el tipo de personaje.

El movimiento hace referencia a los algoritmos que convierten las decisiones tomadas, en un tipo de desplazamiento, siguiendo una dirección y basado en el tipo de acción que se pretende realizar.

- **Búsqueda de camino más corto:**

La búsqueda de caminos es el problema de planear una ruta de desplazamiento en un entorno de búsqueda dado un punto inicial y un punto final; dicha ruta debe considerar la evasión obstáculos estáticos que existan en su entorno. En general los videojuegos utilizan distintas técnicas que serán explicadas como parte del estado del arte, pero la manera común de resolver este problema es abstrayendo la complejidad del mapa a un grafo y posteriormente, aplicando sobre el un algoritmos de búsqueda en grafos. En la Ilustración 1.2 se observa un ejemplo de un camino hallado entre dos puntos.

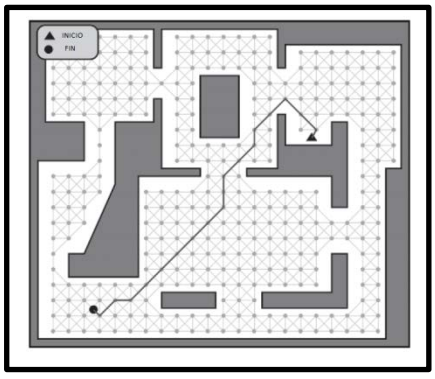

Ilustración 1.2. Búsqueda de caminos entre 2 puntos (BUCKLAND, 2005)

Son cuatro los criterios que se utilizan para evaluar a los algoritmos de búsqueda del camino más corto, estos son (RUSELL, 2006):

- **El tiempo de respuesta**

Es el tiempo que le toma al algoritmo encontrar una solución al problema, este factor es crítico en los videojuegos del tipo RTS.

- **El espacio de memoria**

Mide cuanto espacio de memoria ocupa el algoritmo durante su ejecución. El espacio de memoria se incrementa conforme se almacenan en memoria una mayor cantidad de nodos.

- **La optimalidad**

Es el factor que mide la calidad de la respuesta hallada, suele ser un factor que se sacrifica en cierta manera, para obtener un mejor tiempo de respuesta.

- **La completitud**

Es dice que un algoritmo es completo, si es siempre encuentra una solución a su problema, cuando esta exista.

Por otro lado, se define un **grafo como una representación simbólica de una red**, el cual está compuesto por un conjunto nodos que son enlazados mediante aristas. El enlace de estos representa una relación en el espacio. En la computación un grafo es una estructura de datos que se utiliza para establecer relaciones entre distintos componentes o nodos; A dichas relaciones se les puede asignar un valor numérico llamado costo o peso, de tal manera que se obtiene un grafo ponderado.

Un camino es un conjunto de nodos enlazados por sus aristas o arcos, en un grafo ponderado, el peso de un camino es la suma de los pesos de las aristas (o arcos) que forman dicho camino, por ejemplo en la

Ilustración 1.3 se muestra un grafo balanceado, del cual se observa que el camino (A, B, C, E) tiene un coste de 9.

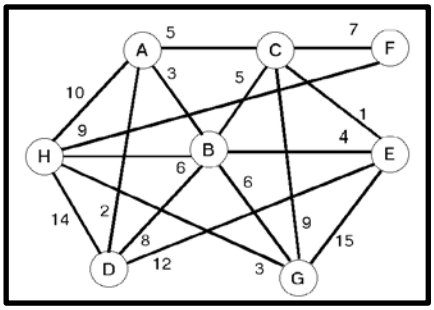

Ilustración 1.3. Grafo balanceado con 8 vértices (RABIN, 2005)

En el desarrollo de videojuegos RTS, los grafos, son las estructuras que se usan para definir la geometría del mundo o mapa del videojuego, por lo tanto, son los espacios de búsqueda sobre los cuales se aplican los distintos algoritmos de búsqueda de caminos. Por ejemplo en la Ilustración 1.4, se observa cómo se utiliza una estructura de grafo para representar un mapa en un videojuego.

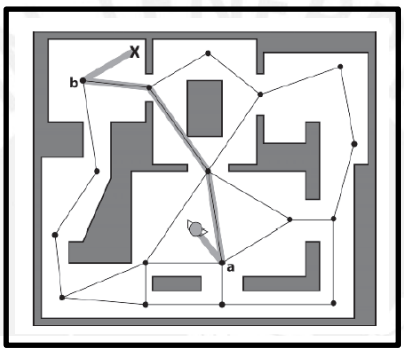

Ilustración 1.4. Mapa de un juego representado como un Grafo de navegación

1.4 Revisión del estado del arte

A continuación, se presentan cuáles son las formas y métodos que se ha venido utilizando en la industria de videojuegos para resolver el problema de búsqueda de caminos así como el de desplazamiento e interacción de los personajes.

1.4.1 Inteligencia artificial en videojuegos RTS

El campo de la inteligencia artificial en los videojuegos se ha desarrollado desde la creación de los primeros juegos en 1970. Uno de los primeros ejemplos de su aplicación son los movimientos de los fantasmas en el popular juego Pac-Man. Sin embargo, en los últimos años, debido a la mayor capacidad de procesamiento que proveen los nuevos dispositivos o consolas, se viene impulsando con más énfasis el desarrollo de este campo en los videojuegos (RABIN, 2005).

Actualmente se han alcanzado importantes avances en los RTS, debido a la complejidad que implica controlar muchos personajes en un escenario, y todo en tiempo real, los principales retos que implican estos videojuegos en el área de la IA, son los algoritmos de Pathfinding y navegación de los personajes en su entorno.

Los videojuegos de estrategia en sus inicios se caracterizaban por jugarse en un tablero y estar basados en turnos, esto significaba que los tiempos de respuesta de los algoritmos no eran una prioridad.

Con el videojuego Dune 2 (WESTWOOD STUDIOUS, 1992), Westwood creo un nuevo género que se convirtió en un éxito principalmente en los videojuegos de PC, este género, cambió la dinámica del juego basado en turnos por un juego en tiempo real, lo cual generaba presión en el tiempo de juego así como una mejor animación gráfica ampliando el mercado de este género a toda una nueva generación de jugadores (ADAMS, 2010).

Posteriormente se crearon dos de los videojuegos considerados precursores en la IA de los RTS, estos fueron: Warcraft: Orcs and Humans (BLIZZARD 2013) y Command and Conquer (WESTWOOD STUDIOUS, 1992); Ambos videojuegos son considerados sinónimos de algoritmos de Pathfinding o búsqueda de caminos, esto básicamente por ser el reto principal en esta clase de videojuegos (MILLINGTON, 2009). Ambos se caracterizan por utilizar escenarios basados en grillas, las cuales son pequeñas cuadrículas que representan sectores del mapa, tal y como se puede observar en la ilustración.

Ilustración 1.5. Mapa de tiles en Warcraft: Orcs and Humans

Además, utilizaron un entorno de navegación estático, es decir que no varía en el tiempo; sin embargo, ya en los últimos años se ha desarrollado videojuegos RTS de

entorno dinámico, como es por ejemplo Company of heros (Relic Entertainment, 2006); en este videojuego se producen cambios en la estructura del mapa en tiempo real, tal y como se muestra en la ilustración, esto implica que en dichos videojuegos una vez que se ha determinado el camino que debe seguir un agente, puede ver necesario replantear su camino en pleno desplazamiento, lo cual aumenta los costos de ejecución del algoritmos.

Además de las grillas utilizadas en juegos como Warcraft y Age of Empires, existen también otras técnicas que se utilizan para describir los espacios de búsqueda en los entornos de videojuegos en dos dimensiones.

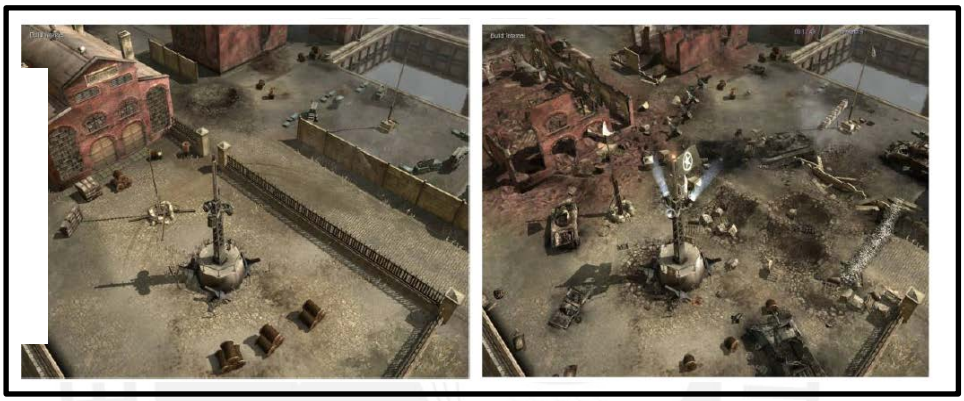

Ilustración 1.6. Estado inicial y final de un entorno (Relic Entertainment, 2006)

1.4.2 Modos de representación de espacios de búsqueda (mapas)

Existen varias técnicas para poder representar un mapa mediante una estructura sobre la cual se puedan ejecutar los algoritmos de búsqueda, las soluciones más utilizadas son: grillas, mallas de navegación y puntos de paso, si bien cada una de estas posee características diferentes, cualquiera de ellas será tratada como un grafo para la aplicación de los algoritmos de búsqueda. Esta simplificación se puede realizar ya que todas estas estructuras se pueden representar como un conjunto de nodos unidos por aristas o enlaces.

- **Mapa o grilla**

Una grilla o rejilla es una cuadrícula o conjunto de polígonos que representan al mapa donde se desarrollarán las acciones del videojuego. Se conoce como vecindario al conjunto de bloques adyacentes, los cuales forman las posibles conexiones entre puntos. Para efectos algorítmicos cada cuadricula será

equivalente a un nodo de un grafo. (Jouni Smed, 1999), en la Ilustración 1.7 se observa un ejemplo de representación mediante el uso de grillas cuadriculadas.

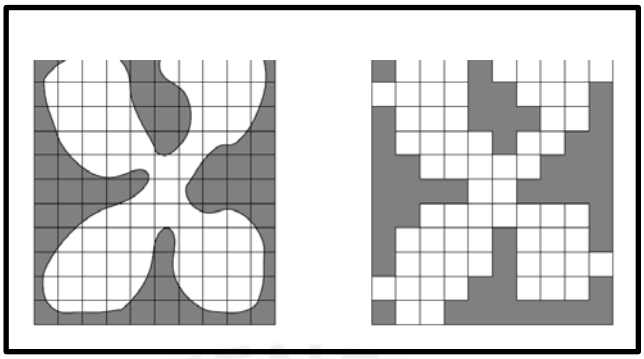

Ilustración 1.7. Mapas basados en Grillas (RABIN, 2005)

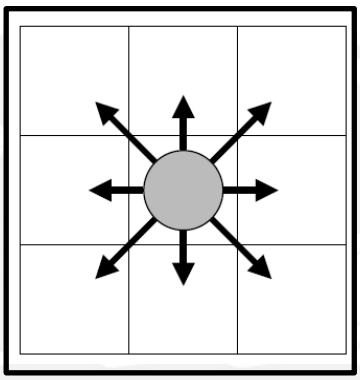

Ilustración 1.8. Direcciones válidas de movimiento de un personaje en una celda

Los mapas basados en grillas se caracterizan además, porque cada nodo está conectado con otros ocho nodos, como se aprecia en la Ilustración 1.8.

- ***Puntos de paso (Waypoints)***

Los puntos de pasos son estructuras que contienen un conjunto de nodos (puntos de visibilidad), con enlaces entre ellos, navegar de un punto de paso a otro, es un sub-problema con una solución simple.

Estos se caracterizan porque todos los puntos de paso son alcanzables desde cualquier otro punto de paso a través de un camino establecido por el Pathfinding, estos pueden implementarse junto con los mecanismos de grillas o mallas de navegación, de tal manera que se reduzcan los puntos de búsqueda.

En la Ilustración 1.9, se puede observar la representación de un mapa utilizando puntos de paso, como se observa dichos puntos no cubren el mapa en su totalidad, por lo que existen áreas no definidas. Los puntos de paso suelen ser creados manualmente cuando se diseña el mapa, por lo tanto es evidente que en entornos dinámicos donde puede cambiar en tiempo real la estructura del mapa, los caminos

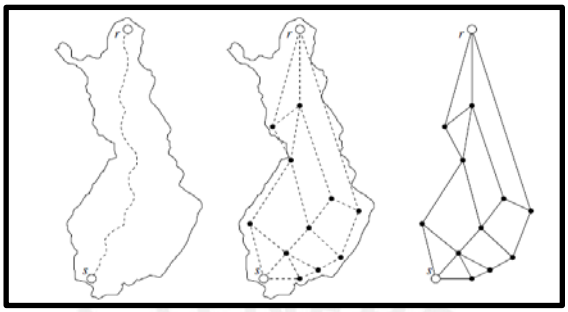

Ilustración 1.9. Ejemplo de Puntos de paso o waypoints (RABIN, 2005)

Trazados entre los puntos necesitarían replantearse ante dicho cambio, es por esto que se recomienda usar en entornos estáticos (De Loura, 2008).

- **Mallas de navegación (navigation mesh)**

Las mallas de navegación son un método para representar un mapa mediante un conjunto de polígonos convexos, que describen áreas transitables del entorno. Son estructuras simples e intuitivas que permiten a los personajes desplazarse y buscar caminos el mapa del juego (SNOOK, 2000).

Las mallas de navegación solamente tienen conocimiento de las partes estáticas del mundo, de tal manera que se necesita una capa distinta al Pathfinding, que se encargue de los obstáculos dinámicos (RABIN, 2005), en la Ilustración 1.10 se observa la representación de una malla de navegación.

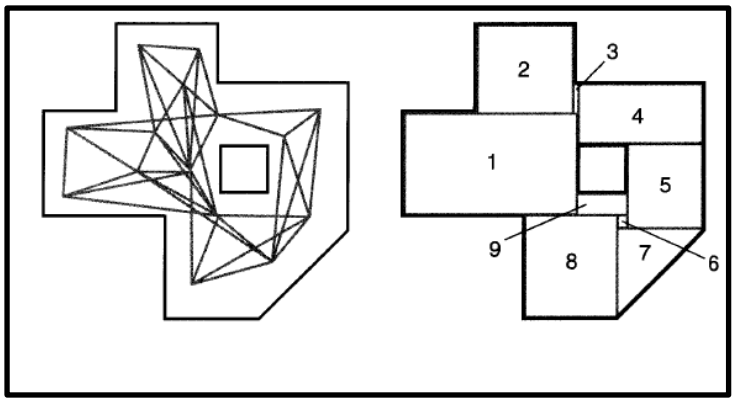

Ilustración 1.10. Ejemplos de mallas de navegación (RABIN, 2005)

1.4.3 Algoritmos que pueden resolver la búsqueda de caminos

En este apartado, se discuten algunos de los principales algoritmos de búsqueda los cuales se utilizan como solución al problema de pathfinding en videojuegos RTS. En general, se propone la utilización de algoritmos de búsquedas en grafos informados.

- **Algoritmo de Dijkstra**

El algoritmo de Dijkstra fue creado en 1959 por Edsgar Dijkstra (Dijkstra, 1959), es uno de los algoritmos más populares en las ciencias de la computación, el cual resuelve entre otros, un problema fundamental de la teoría de grafos: el problema del camino más corto en un grafo balanceado.

Este algoritmo se caracteriza porque su solución encuentra el camino más corto desde un punto de inicio a cualquier otro punto, es una versión básica del algoritmo A* (MILLINGTON, 2009).

Para hallar su camino, este algoritmo comienza a revisar cada vértice que es alcanzable desde el vértice origen y que no ha sido analizado, luego los agrega a

una lista de vértices que deben ser analizados; este proceso se repite y se va expandiendo al conjunto de vértices analizado, hasta encontrarse con el vértice destino, al buscar la solución óptima en cada iteración, la cantidad de nodos que explora en la búsqueda es elevada. En la Ilustración 1.11, se puede observar el camino hallado por el algoritmo de Dijkstra, los nodos celestes son aquellos que han sido explorados al buscar la solución y como se observa hay nodos explorados bastante alejados de la solución. Es por esto que el principal problema con este algoritmo es que si bien encontrará la mejor solución, este explora una cantidad de nodos de manera indiscriminada, muchos de los cuáles están alejados de la solución.

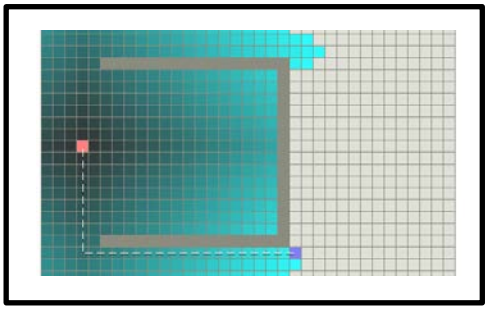

Ilustración 1.11. Camino hallado con algoritmo de Dikjstra (Standford, 2013)

- **Algoritmos de búsqueda primero el mejor**

Los algoritmos de búsqueda primero el mejor, son algoritmos informados, es decir utilizan el conocimiento que tienen del vértice destino, de tal manera que en cada paso, consideran el vértice que se aproxima más al vértice destino; es decir, se van expandiendo hacia el vértice más prometedor en términos de llegar al objetivo.

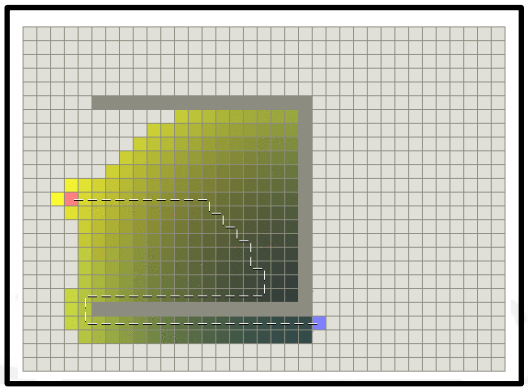

Ilustración 1.12. Camino hallado usando primero el mejor,(Standford, 2013)

Como se observa en la Ilustración 1.12, el algoritmo de búsqueda primero el mejor, analiza una menor cantidad de vértices en comparación con el algoritmo de Djikstra, lo que significa un menor consumo del procesador; sin embargo, el camino de solución no es el óptimo.

- **Búsqueda voraz**

Es un algoritmo de búsqueda del tipo primero el mejor, el cual busca minimizar el costo para llegar al objetivo, de tal manera que al momento de evaluar los nodos, siempre se expande hacia el nodo que se estime está más cerca del nodo objetivo. Dicha estimación evidentemente no puede ser determinada con total certeza, por lo tanto a la función que realiza dicha estimación se le conoce como función heurística (RUSSELL, 2006).

- **Algoritmo A***

Fue creado en 1972 por Peter Hart y Nils Nilsson (HART, 1972), es una especialización del algoritmo de Djikstra, pero a diferencia de este, utiliza una heurística que le permite en cada paso determinar la distancia del vértice o nodo a

analizar con el vértice final, similar a la búsqueda voraz, de tal manera que, obtiene una solución buscando reducir el costo para llegar al objetivo.

El algoritmo A* combina la aproximación heurística del algoritmo voraz con la precisión del algoritmo de Dijkstra, es decir su éxito radica en que combina la información que utiliza el algoritmo de Dijkstra (favoreciendo los vértices que están más cerca al origen) y la información que proveen los algoritmos de búsqueda voraz, favoreciendo el vértice que está más cerca al destino (STANDFORD, 2013).

- **Búsqueda iterativa en profundidad IDA***

Utiliza un método de valorización de nodos similar al algoritmo A*, es decir favoreciendo los nodos que están más cerca al origen y los que están más cerca al destino; sin embargo, se diferencia con este en que no guarda en memoria todos los caminos explorados, sino sólo del último que es explorado, esto significa que va a explorar los mismo nodos varias veces pues no guarda un recuerdo de lo que ya exploro (RUSELL, 2003). Como consecuencia de volver a explorar los mismos nodos, el tiempo de ejecución del IDA* es mayor al A*, pero reduce bastante el consumo de memoria.

- **Búsqueda Jerárquica**

La búsqueda jerárquica utiliza un planeamiento de camino similar a la manera de pensar de las personas, es decir, primero se planea el movimiento a nivel macro para las distancias largas y finalmente a nivel más detallado para buscar una aproximación de camino con mayor precisión (MILLINGTON, 2009), la búsqueda jerárquica sin embargo puede tener algunos comportamientos extraños, ya que esta logra un aproximado y no una solución óptima.

1.4.4 Algoritmos y mecanismos de movimiento e interacción

Otro requerimiento fundamental de la AI en videojuegos que se ha venido desarrollando, es el movimiento de los personajes. Algunos videojuegos como por ejemplo los basados en turnos, no demandan mucho en cuanto al movimiento de los agentes, sin embargo, los RTS siempre necesitan de ellos, dependiendo las características del videojuego, se han ido desarrollando nuevas técnicas en este

campo, a diferencia de los algoritmos de búsqueda, los cuales se desarrollan sobre un entorno estático, estos utilizan un entorno de navegación dinámico.

- **Comportamientos de dirección**

También conocidos como Steering behaviours, son unos algoritmos definidos por Craig Reynolds (MILLINGTON, 2009), para implementar el movimiento de agentes en un entorno, los más conocidos y utilizados en los RTS son: Seguir el camino, alineamiento, cohesión, seguimiento, persecución, entre otros. Estos algoritmos lo que hacen es modificar la dirección, velocidad y aceleración de los agentes de tal manera que se logre el comportamiento deseado.

- **Flocking**

Se conoce como flocking, al mecanismo que se obtiene en base a tres comportamientos de dirección, los cuales son: separación, alineamiento y cohesión. La separación busca que cada uno de los agentes verifique permanentemente no estar muy cerca del resto de agentes, de tal manera que si se aproximan mucho, los obliga a alejarse. Por otro lado, el alineamiento verifica que los agentes mantengan la misma dirección y velocidad de desplazamiento que el resto de agentes; finalmente la cohesión es el comportamiento que se encarga que los agentes no se distancien mucho de manera que se mantengan en grupo.

1.4.5 Conclusiones sobre el estado del arte

La revisión del estado del arte permitió obtener conclusiones importantes, respecto a las herramientas y algoritmos más utilizados para resolver problemas similares al planteado en este proyecto, además, permitió identificar las ventajas y desventajas en el uso de cada uno de estas.

En cuanto a la representación de los espacios de búsqueda, se determinó que son tres los mecanismos que más se utilizan para representar los mapas en videojuegos RTS de dos dimensiones: Las mallas de navegación, las grillas cuadriculadas y los puntos de paso o waypoints; como bien se ha mencionado en el estado del arte, cada uno ofrece sus ventajas y sus desventajas, las cuales se comparan de manera cualitativa en la

Tabla 1.1.

Característica	Mallas Navegación	Puntos de Paso	Grillas

Permite representar toda el área de búsqueda	SI	NO	SI
Tipo de elaboración	Complejo, depende del factor artístico en el diseño del mapa	Sencillo, Lo realiza manualmente el desarrollador	Intermedia, Complejidad aumenta con la isometría
Tipo de nodos	Polígonos	Puntos unidos por aristas	Cuadrículas o hexágonos
Tamaño del grafo de navegación asociado (cantidad de nodos)	Regular	Pequeño	Grande
Nivel de precisión de representación de los espacios	Aceptable, depende del tamaño y forma de los polígonos	Ineficiente	Aceptable, depende del tamaño de las cuadrículas.

Tabla 1.1. Tabla de comparación de técnicas para representar áreas de búsqueda

Como segundo punto tenemos a los algoritmos usados para resolver el problema de búsqueda de caminos sobre un área de búsqueda, al respecto, se observó que existen soluciones para distintos tipos de necesidades, así por ejemplo, el algoritmo de Dijkstra explora una mayor cantidad de nodos y por lo tanto siempre encuentra una solución óptima, aunque le tome un tiempo considerable conforme se incremente la cantidad de nodos a explorar.

También se explicó los algoritmos de búsqueda primero el mejor como por ejemplo el algoritmo voraz y el A*. El voraz reduce el costo del camino hallado, dándole prioridad a los nodos que se acercan más al nodo objetivo, para esto, utiliza una función heurística que le permita estimar cual es dicho nodo. Por otro lado el A* utiliza una heurística similar al algoritmo voraz, pero además busca reducir el costo del camino dándole prioridad también a los nodos que estén más cerca al nodo inicial similar al algoritmo de Dijkstra.

A continuación en la Tabla 2.1 se muestra una comparación cualitativa de las características de algunos de los algoritmos más utilizados.

Característica	Dijkstra	Voraz	A*	IDA*
Tipo de Solución	Óptima	Aproximada	Aproximada	Aproximada
Utiliza Heurística	NO	SI	SI	SI
Entornos Dinámicos	NO	NO	NO	NO
Espacio de memoria	Alto consumo	Regular consumo	Regular consumo	Consumo reducido
Cantidad de nodos explorados	Muy elevado	Aceptable con una buena heurística	Aceptable con una buena heurística	Elevado
Tiempo de Ejecución	Elevado, por la cantidad de nodos que explora	Aceptable con buena heurística	Aceptable con buena heurística	Elevado, porque explora el mismo camino más de una vez

Tabla 1.2. Tabla de comparación de algoritmos dé búsqueda de caminos informados

Es importante recalcar que todos estos algoritmos aumentan su tiempo de ejecución exponencialmente, conforme se alejan la distancia entre nodo inicial y final, de tal manera que en grafos con mayor cantidad de nodos, los algoritmos disminuyen su rendimiento.

Por lo tanto, el aporte que se busca dar en este proyecto, es el de combinar distintas técnicas y algoritmos , de acuerdo a distintos escenarios planteados, para aprovechar los beneficios que brinda cada una de estas y reducir sus desventajas, considerando además, un balance entre el tiempo de ejecución y la calidad del resultado, Para esto, se implementó un componente de búsqueda de caminos que utilice las características del algoritmo A*, es decir, que use una función heurística como lo hace lo hace el algoritmo voraz, dándole prioridad a los nodos que se acercan al destino pero a su vez manteniendo el camino correcto que propone Djikstra, priorizando el nodo más cerca al origen.

Además se planteó utilizar un algoritmo jerárquico HPA*, el cual use dos jerarquías de búsqueda, para lograr reducir el impacto de tiempo que tienen los algoritmos de búsqueda, cuando son usados en grafos compuestos por muchos nodos. Por lo

cual se utilizará un área de búsqueda basada en grillas en el nivel superior, buscando precisión y como apoyo para el segundo nivel una mezcla entre grillas y puntos de paso, para obtener un grafo pequeño en el segundo nivel, mejorando así los tiempos de ejecución.

Como se mencionó también, la utilización de búsqueda jerárquica significa que se pude tener efectos extraños en casos particulares, ya que esta halla un camino aproximado, por lo tanto, el componente propuesto utilizará la búsqueda A* y HPA* en distintas situaciones, las cuales se muestran con detalle en el capítulo 0. Este componente también contará con la implementación de algunos de los mecanismos mencionados sobre movimiento de los agentes, en particular el seguimiento del camino y algunas características del comportamiento de dirección.

2. Capítulo 2

En el presente capítulo se desarrolla el objetivo general, los objetivos específicos y los resultados esperados asociados a los objetivos del proyecto. Posteriormente, se indica cómo se logró cada uno dichos resultados esperados, detallando los métodos y las herramientas que se utilizaron en el proceso.

Finalmente, se explica la metodología que se utilizó en el desarrollo del producto, así como la gestión del proyecto; haciendo énfasis en las características específicas que dichas metodologías nos proporcionan y cómo se aplicaron en el desarrollo del proyecto.

2.1 Objetivo General

Realizar el análisis, diseño e implementación de un componente de búsqueda de caminos y control de movimiento, para implementar un mecanismo de navegación autónoma de los personajes en un videojuego de estrategia en tiempo real en dos dimensiones.

2.2 Objetivos específicos

Los objetivos específicos del presente proyecto son:

- **Objetivo 1:** Describir de manera sistemática los movimientos y formas de interacción de los personajes de acuerdo al guion y reglas de un videojuego RTS (en tiempo real) que utiliza un mapa isométrico de dos dimensiones.

- **Objetivo 2:** Diseñar la arquitectura de software necesaria para manejar el movimiento e interacción de los personajes de un videojuego RTS, que utiliza un mapa isométrico en dos dimensiones.

- **Objetivo 3:** Proponer una solución al problema de búsqueda del camino más corto en un videojuego de naturaleza RTS, sobre un mapa isométrico en dos dimensiones, considerando factores como velocidad de ejecución y consumo del memoria del algoritmo.

- **Objetivo 4:** Proponer las estructuras de datos y los mecanismos necesarios que permitan gestionar el desplazamiento e interacción de los personajes acorde a lo obtenido en la descripción sistematizada.

- **Objetivo 5:** Adaptar los algoritmos antes implementados, integrándolos al videojuego tomado como base para verificar su correcto funcionamiento.

2.3 Resultados esperados

- **Resultado 1 para el objetivo 1**: Descripción sistematizada de los movimientos y mecanismos de interacción que se requieren implementar.

- **Resultado 2 para el objetivo 2:** Arquitectura de software necesaria para gestionar el movimiento e interacción de los personajes, incluyendo las clases y tipos de datos que permitan adaptar los algoritmos.

- **Resultado 3 para el objetivo 2:** Prototipo que permita probar la arquitectura.

- **Resultado 4 para el objetivo 3:** Implementación de los algoritmos de búsqueda de camino A* y HPA* aplicados a un videojuego RTS.

- **Resultado 5 para el objetivo 3:** Resultados de la experimentación numérica al comparar los algoritmos A* y HPA*, respecto al tiempo de ejecución y consumo de memoria.

- **Resultado 6 para el objetivo 4:** Implementación de las estructuras de datos y los mecanismos necesarios para gestionar el desplazamiento y la interacción de los personajes entre sí y con los objetos del juego.

- **Resultado 7 para el objetivo 5:** Prototipo del videojuego con todas las funcionalidades que los algoritmos especificados han permitido desarrollar.

2.4 Herramientas, métodos y procedimientos

En el presente apartado, se detallan las actividades realizadas y herramientas utilizadas para alcanzar cada uno de los resultados esperados.

2.4.1 Mapeo de las actividades y herramientas a realizar:

A continuación, se presenta la Tabla 2.1, la cual muestra un mapeo entre los resultados esperados del proyecto con las actividades realizadas y herramientas utilizadas para el logro de cada resultado. Luego se describe cada una de las herramientas y se fundamente su uso.

Resultados esperados	Actividades a realizar	Herramientas a utilizarse
RE1: Descripción sistematizada de los movimientos y mecanismos de interacción que se requieren implementar.	I. Se revisó videojuegos existentes del tipo RTS, que cumplan con algunas de las características requeridas por el componente a implementar, además de trabajos realizados sobre el tema de IA en videojuegos. II. Se describió de manera	• Revisión de textos y otros proyectos de tesis sobre videojuegos. • Videojuego RTS: Age of Empires.(Blizzard, 2013)

Resultados esperados	Actividades a realizar	Herramientas a utilizarse
	sistemática las características y los mecanismos que se desean implementar	
RE2: Arquitectura de software necesaria para gestionar el movimiento e interacción de los personajes, incluyendo las clases y tipos de datos que permitan adaptar los algoritmos.	I. Se buscó de distintas fuentes, herramientas y métodos existentes que permitan diseñar un modelo de arquitectura de software para el componente a desarrollar. II. Se determinó utilizar el modelo de vistas 4+1 que propone la IEEE para el diseño de las distintas vistas de la arquitectura de software. III. Se determinó utilizar una arquitectura ya conocida, propuesta por Michael Doherty, para el desarrollo de videojuegos RTS. V. Utilizando el programa Visio 2010 y el lenguaje UML, se elaboraron los entregables que el modelo de vistas 4+1 sugiere para cada vista, respecto a la vista de desarrollo, se realizó basado en lo propuesto por Doherty.	• Revisión e investigación de literatura sobre arquitectura de software en videojuegos. • Planos arquitectónicos de la arquitectura de software del modelo de vistas 4 + 1. • Lenguaje unificado de modelado (UML). • Microsoft Visio 2010.
RE3: Prototipo que	I. Se implementó en lenguaje	• Entorno de

Resultados esperados	Actividades a realizar	Herramientas a utilizarse
permita probar la arquitectura.	Java las clases bases y las estructuras de datos del componente a desarrollar, utilizando el entorno de programación Netbeans 7.3.1, siguiendo la arquitectura descrita en el resultado esperado 2.	programación IDE Netbeans 7.3.1 • Java development kit 7
RE4: Implementación de los algoritmos de búsqueda de camino A* y HPA* aplicados a un videojuego RTS	I. Se investigó los distintos tipos de algoritmos de búsqueda que existen y pueden resolver el problema planteado. II. Una vez seleccionado el tipo de algoritmo requerido (búsqueda primero el mejor), se eligió los algoritmos el A* y el HPA*. III. Se seleccionó la heurística adecuada, con su respectiva métrica de distancia para implementar la función heurística del algoritmo. IV. Se definió los tipos y las estructuras de datos que se requería para su implementación. V. Se elaboraron los archivos de configuración de mapas que se utilizaron para cargar las estructuras que utilizan los	• Técnicas de búsqueda Heurística. • Métricas de distancia • Algoritmo A* • Entorno de programación IDE Netbeans 7.3.1 • Java development kit JDK 7 • Repositorio SVN Tortoise

Resultados esperados	Actividades a realizar	Herramientas a utilizarse
	algoritmos como entorno de búsqueda. VI. Se Implementó ambos algoritmos elegidos para la búsqueda de camino utilizando el lenguaje de programación java, en el entorno Netbeans.	
RE5: Resultados de la experimentación numérica al comparar los algoritmos A* y HPA*, respecto al tiempo de ejecución y consumo de memoria.	I. Se determinó las variables que se utilizarían en la experimentación. II. Se plantearon las hipótesis de un experimento para comparar las medias de los tiempos ejecución y el uso de memoria de los algoritmos A* y HPA*. III. Se determinaron los estadísticos a utilizar. IV. Se eligió cuáles serían los casos de pruebas para las corridas de los algoritmos. V. Se ejecutó los algoritmos en el entorno de programación Netbeans, y se anotó los tiempos de ejecución para cada algoritmo. VI. Se realizó un experimento de comparación de medias para	• Resultado esperado 4. • Herramientas estadísticas para un experimento de comparación de medias en dos muestras. • IBM SPSS Statics 20 • Microsoft Excel 2010

Resultados esperados	Actividades a realizar	Herramientas a utilizarse
	las muestras obtenidas de la ejecución de cada algoritmo utilizando el software SPSS Statics 20. VII. Se analizaron los resultados del experimento para descartar o aceptar las hipótesis planteadas.	
RE6: Implementación de las estructuras de datos y los mecanismos necesarios para gestionar el desplazamiento y la interacción de los personajes entre sí y con los objetos del juego	I. Se utilizó el resultado esperado 3, el cual contiene la descripción de las características que se desean implementar. II. Se definió los tipos y las estructuras de datos que se requería para implementar los mecanismos requeridos. III. Se Implementaron los algoritmos de control de movimiento e interacción de los personajes acorde a lo especificado en el resultado esperado 3.	• Revisión e investigación de literatura sobre navegación de agentes. • Entorno de programación IDE Netbeans 7.3.1 • Java development kit 7
RE7: Prototipo del videojuego con todas las funcionalidades que los algoritmos especificados ha	I. Se utilizó el resultado esperado 3, el cual contiene la descripción de las características que se desean implementar, para verificar	• Entorno de programación IDE Netbeans 7.3.1 • Java development kit

Resultados esperados	Actividades a realizar	Herramientas a utilizarse
permitido desarrollar.	cada caso de prueba. II. Se adaptó el componente implementando con los otros componentes (lógicas y gráficas), utilizando el entorno de programación Netbeans y el repositorio Subversion Tortoise. III. Con los mismos casos de pruebas utilizados para la elaboración de las muestras que se utilizaron en la experimentación numérica.	7 • Módulo de la lógica del videojuego • Módulo del manejo gráfico del videojuego • Librería XStream

Tabla 2.1 Tabla de actividades y herramientas utilizadas

2.4.2 Descripción de las herramientas utilizadas

A continuación, se describen cada una de las herramientas utilizadas para alcanzar los resultados esperados y se justifica el uso de cada una de ellas.

- **Herramienta 1: Modelo de vistas 4+1 de Kruchten**

Los planos arquitectónicos del modelo de vistas 4+1, son un modelo utilizado para describir la arquitectura de sistemas de software elaborado por Philippe Kruchten, este recomienda la utilización de cinco vistas concurrentes: La vista lógica, de procesos, física, de desarrollo y la de escenarios; cada una de estas vistas permite además, modelar el diseño desde distintos puntos de vista. (KRUCHTEN, 2013).

Justificación del uso:

Como resultado esperado 3, se pretende obtener el diseño de la arquitectura de software necesario para la implementación de los distintos componentes del

videojuego, por lo tanto como primer paso se determinó utilizar el modelo de vistas 4 + 1 de Kruchten, recomendado por el estándar “IEEE 1471-2000” (IEEE, 2013) para el diseño de las distintas vistas de la arquitectura.

- **Herramienta 2: Lenguaje unificado modelado UML**

UML es un lenguaje visual usado para especificar, construir y documentar artefactos utilizados en el diseño de sistemas de software. Estos artefactos permitirán visualizar, validar y comunicar de manera adecuada la arquitectura del software (UML, 2013).

Justificación del uso:
Este lenguaje de modelado se utilizó para el diseño de los distintos diagramas que propone el modelo de vistas 4 + 1 para cada uno de los entregables o artefactos. Dichos entregables se encuentran en los documentos anexos del proyecto.

- **Herramienta 3: Microsoft Visio 2010**

Visio 2010 es un software propiedad de Microsoft utilizado para crear, actualizar y publicar diagramas fáciles de comprender, además, permite modelar procesos presentando la información de una manera visual mediante el uso de organigramas, flujos de trabajo y procesos de negocio (VISIO, 2013).

Justificación del uso:
Se utilizó este programa para la elaboración de los distintos entregables que propone el modelo de vistas 4+1 de kruchten, siguiendo el estándar de UML.

- **Herramienta 4: Entorno de programación IDE Netbeans**

El IDE Netbeans es un entorno de programación que permite desarrollar aplicaciones de escritorio, móviles y web en distintos lenguajes de programación, tales como Java, HTML, Javascript. (NETBEANS, 2013)

Justificación del uso:
Se decidió utilizar Netbeans 7.0 como entorno de programación, debido a las ventajas que ofrece tales como: Edición, corrección de código dinámicamente, procesos de refactorización y limpieza de código; así como las herramientas de debug y análisis que permiten optimizar el código. También permite la instalación

de plugins y librerías de manera sencilla; además, esta herramienta es de descarga gratuita.

- **Herramienta 5: Java development kit 7**

Es la plataforma que provee un conjunto de herramientas para el desarrollo y despliegue de aplicaciones en Java, debido al uso de una máquina virtual JVM, es independiente de la plataforma donde se ejecuten sus aplicaciones

Justificación del uso:
Se decidió utilizar JDK 7 por ser una herramienta de descarga libre, además que el lenguaje de programación java orientado a objetos es ya conocido y por lo tanto, permitió reducir la curva de aprendizaje.

- **Herramienta 6: Técnicas de búsqueda Heurística**

Para evaluar un algoritmo hay dos principios básicos: La calidad del resultado y su tiempo de ejecución, mediante el uso de una heurística se sacrifica algunos de los dos principios, a fin de beneficiar al otro. Se decidió utilizar como heurística la métrica de distancia de Chebyshev (STANDFORD, 2013).

Justificación del uso:
Como se eligió el algoritmo de búsqueda de caminos del tipo primero el mejor, se tuvo que elegir una heurística para que en cada iteración se dé prioridad al ”mejor nodo”; por lo tanto, la heurística empleada para el algoritmo A* fue la de favorecer al nodo que tuviera el menor costo para llegar a su destino, sin importar los obstáculos de por medio.

- **Herramienta 7: Métricas de distancia**

Son fórmulas que permiten calcular la distancia entre dos puntos en un plano XY, las más comunes son: la euclidiana, de Chebyshev y la distancia Manhattan.

Justificación del uso:
Se eligió una de las métricas, como herramienta para implementar la función heurística de los algoritmos de búsqueda de caminos, ya que como se mencionó,

dicha heurística requería calcular la distancia entre el nodo a evaluar y el nodo destino. En particular, se eligió la distancia de Chebyshev.

- **Herramienta 9: Algoritmo A***

Es un algoritmo de búsqueda de grafos primero el mejor, creado por Peter Hart y Nils Nilsson, es utilizado para resolver el problema del camino más corto (Hart, 1998).

Justificación del uso:
Se utilizó este algoritmo en vista que se suelen utilizar muchas adaptaciones de este para resolver el problema de búsqueda de caminos en videojuegos (RABIN, 2005), si bien presenta ciertas falencias, es parte del proyecto, superar dichas fallas.

- **Herramienta 10: Repositorio SVN Tortoise**

Es una herramienta utilizada para el control de versiones, como no es parte de ningún IDE, se puede utilizar como controlador de versiones con cualquier IDE (TIGRIS, 2013); para este caso, se utilizó Netbeans.

Justificación del uso:
Se eligió esta herramienta para el control de versiones, ya que como se ha mencionado, el componente que se desarrolla en este proyecto, es parte de un proyecto mayor, desarrollado por otros; por lo tanto, resultó de mucha utilidad controlar las distintas versiones; al igual que otras herramientas utilizadas, es de descarga gratuita.

- **Herramienta 12: Herramientas estadísticas para un experimento de comparación de medias en dos muestras**

Se Realizó un experimento numérico para comparar las medias de dos muestras poblacionales independientes, para lo cual se utilizó el estadístico T-Student, además se realizaron la pruebas de Kolgomorov-Smirnov y la prueba de Levene.
Justificación del uso:
Se utilizó T-Student como estadístico para la comparación de medias, lo cual sólo se puede aplicar bajo el supuesto que la muestra tenga una distribución normal y

tenga varianzas iguales; para la verificación de lo primero, se utilizó la prueba K-S, y para verificar las varianzas se tuvo que utilizar la prueba de Levene.

- **Herramienta 13: IBM SPSS Statics 20**

Es un software estadístico de análisis predictivo fácil de utilizar para usuarios empresariales, analistas y programadores estadísticos, el cual provee funciones analíticas básicas para la estadística de datos (IBM, 2013).

Justificación del uso:
Se utilizó como herramienta de apoyo para la ejecución de los experimentos estadísticos mencionados en la herramienta 12.

- **Herramienta 14: Microsoft Excel 2010**

- **Herramienta 16: Módulo del manejo gráfico del videojuego**

Es un módulo creado en otro proyecto de fin de carrera, el cual gestiona el manejo de los recursos gráficos en un videojuego del tipo RTS.

Justificación del uso:
El último resultado esperado es la adaptación del componente implementado en un prototipo del videojuego, para verificar el correcto funcionamiento de los algoritmos, como parte de dicha adaptación, se utilizó el módulo mencionado.

- **Herramienta 17: Liberia Xstream**

Es una herramienta utilizada para serializar objetos XML desde Java, la cual se utiliza para transporte, configuración y persistencia (XStream, 2013).

Justificación del uso:
Se utilizó para la creación y carga de los archivos de configuración de los personajes del videojuego, tal y como se muestra en el capítulo 4.

2.4.3 Metodología

Para la gestión del proyecto se utilizó PMBOK, y para la implementación del producto la metodología ágil Extremme programming. Ambas metodologías proporcionan un conjunto de principios, de los cuales se eligieron los que se consideraron adecuados para este proyecto.

PMBOK plantea un conjunto de principios los cuales incluyen prácticas tradicionales que son muy utilizadas en la de dirección de proyectos, además reconoce cinco grupos de procesos y nueve áreas de conocimiento (PMI, 2008). A continuación se presentan como se utilizaron algunos de estos en la gestión del proyecto.

- **Proceso de iniciación**

Identificación de los Stakeholder:

El principal interesado, es el autor del proyecto, los asesores y los profesores del curso de Tesis 1 y 2. Otro interesado es el grupo avatar (AVATAR PUCP) quienes solicitaron la implementación del videojuego que a su vez requiere del componente de búsqueda de caminos y control de movimiento.

- **Proceso de Planificación**

Desarrollo del Plan para la Dirección del Proyecto:

Se elaboró un cronograma para controlar los avances del trabajo, ver anexo 8, en el cual se puede observar cada actividad que se debe realizar para cumplir los objetivos del proyecto y las fechas establecidas.

- **Gestión del Alcance**

Se realizó la definición del alcance, el cual permitió establecer que hará y que no el componente que se implementará, de tal manera que se pueda establecer límites al proyecto y por lo tanto los tiempos que se estimen para su desarrollo sean correctos.

- **Definición y secuencia de actividades**

Se realizó la definición de actividades, que permitieron obtener resultados esperados del proyecto, se pueden observar en el apartado 2.4.1.

- **Gestión de los riesgos del proyecto**

Para la gestión de los riesgos que pueden surgir durante el desarrollo del proyecto, se elaboró un plan para la gestión de riesgos, ver anexo 1.

- **Gestión de los costos del proyecto**

La definición dela viabilidad económica del proyecto, esto se encuentra en el apartado 2.6.2.

Para el producto, se utilizó la metodología Extremme Programming, también conocida como XP; la cual, es una metodología ágil creada por Kent Beck, se caracteriza por haber sido utilizada con éxito por muchas compañías de distintos tamaños e industrias y porque puede ser usado por equipos de tamaño pequeño a mediano para el desarrollo de software de alta calidad en un tiempo previsible y con una sobrecarga de trabajo mínima. (Extreme Programming, 2013)

El ciclo de desarrollo de software en XP se divide en cinco etapas principales, las cuales de se desarrollan de manera iterativa, así por ejemplo, la etapa del planeamiento se realizará durante todo el ciclo al igual que la codificación o las pruebas, a continuación se explicará con mayor detalle cuales son cada una de las buenas prácticas que nos ofrece XP como metodología en cada etapa y cómo se utilizaron en este proyecto.

- **Etapa de planeamiento**

En esta etapa, se crean las historias de usuarios, las cuales son el equivalente al catálogo de requerimientos definidos por RUP, con la diferencia que utilizan un lenguaje menos técnico.

A diferencia de otras metodologías no ágiles en las cuales el planeamiento de los requerimientos se realiza casi en su totalidad en el inicio del proyecto, en XP estas se realizan en los planning meetings o reuniones de planeamiento, las cuales se realizan al comienzo de cada iteración obteniendo como producto un plan de lanzamientos, ver anexo 10, en donde se indican las fechas de entrega de cada historia de usuario.

- **Etapa de Gestión**

Para la gestión del desarrollo del proyecto, la metodología XP sugiere tener un ambiente de programación abierto, en el cual, el equipo de desarrollo trabaje de manera colectiva, también es importante tener el control de la velocidad de desarrollo del proyecto, esto se consigue verificando cuantas historias de usuarios se han cumplido en cada iteración de tal manera que se podrá planificar el siguiente reléase de manera más exacta.

- **Etapa de Diseño**

El diseño en XP tiene como objetivo principal la simplicacidad, aunque siempre el código debe cumplir los siguientes criterios: testeable, entendible e identificable.
Otro concepto importante en el diseño es la creación de spike solutions, las cuales permiten simplificar la solución de algún problema, aislándolo y encontrando potenciales soluciones, de tal manera que se enfoquen en lo esencial del problema y se deje de lado otros factores.

Es importante también la Refactorización del código, la eliminación de redundancias y funcionalidades sin uso, así como mantener código limpio y comentado.

- **Etapa de Codificación**

Como se ha mencionado, al comienzo de cada iteración se debe determinar el release plan de cada release meeting, en él se debe priorizar las principales tareas a realizar por parte de los desarrolladores, es importante mantener un estándar de codificación para contribuir con el collective ownership o propiedad de código colectiva.

Otro concepto importante en esta etapa es la programación en pares, mediante esta se consigue obtener un código conocido por varios de los desarrolladores, además se contribuye a la retroalimentación de código y a obtener un código más eficiente.

Finalmente, se debe considerar el factor de la integración, al respecto, XP recomienda que el desarrollo sea en paralelo, pero la integración de código sea secuencial.

- **Etapa de Pruebas**

Las pruebas se realizan de manera constante mediante pruebas unitarias, por lo tanto, un release plan no puede ser dado como terminado, hasta que se verifique cada una de las pruebas asociadas.

Hay una resistencia a consumir mucho tiempo realizando pruebas sin embargo, las pruebas continuas permitirán simplificar futuras revisiones, además permite reducir la cantidad de bugs y errores que puedan surgir en el desarrollo.

2.4.4 Adaptación de la metodología

A continuación se explica la manera en que se utilizó la metodología, y como beneficio en el desarrollo del componente.

- Desarrollo iterativo e incremental con pruebas unitarias continuas:

El producto final de este proyecto es un componente del videojuego, y por lo tanto debe ser posteriormente integrado con otros componentes los cuales en conjunto permitirán implementar el videojuego en su totalidad.

Cada iteración planificada con su respectivo release plan, permitió que se puedan al realizar pequeñas funcionalidades por separado, además que se pudo ir replanteando las prioridades de cada historia de usuario en función de los logros de cada iteración.

- **Propiedad de código colectivo**

La propiedad de código colectivo permitió que cada uno de los desarrolladores de los distintos componentes del videojuego, conozcan el código en su totalidad, de tal manera que se facilitó la integración de los distintos componentes, en particular para el caso de la implementación del módulo de búsqueda de camino y movimiento e interacción de los personajes, era importante conocer los mecanismos del módulo de la interfaz gráfica del videojuego, pues requiere de sus métodos para mostrar visualmente los algoritmos del propio módulo, así como del módulo con las reglas y la lógica del videojuego.

- **Simplicidad en el código y documentación necesaria:**

Esto permitió que cada componente implementado pueda ser manejado como librerías de clases, las cuales, si bien estarán integradas al videojuego, también podrán ser utilizadas en el desarrollo de otros proyectos similares.

- **Velocidad del desarrollo del proyecto**

El tiempo que se estimó para cada iteración en promedio fue de tres semanas, además en cada una de estas se priorizó con la implementación de las tareas que son parte principal del componente, además, se priorizó para las primeras iteraciones la implementación del algoritmo de búsqueda de caminos y posteriormente la ejecución del movimiento de los personajes.

- **Historias de usuario**

La elaboración de las historias de usuarios fue realizada en conjunto por los asesores del presente proyecto y los desarrolladores del mismo. En el anexo 2 se detallan las historias de usuario del componente de búsqueda de camino y movimiento e interacción de los personajes que se elaboraron.

- **Integración de código**

Tal y como sugiere XP, la codificación en cada iteración de los distintos módulos se hizo en paralelo; sin embargo, la integración se realizó por medio de tokens (virtuales), así cada desarrollador tenía un turno determinado en el cual podía integrar cualquier cambio de su proyecto propio en el repositorio principal del videojuego.

2.5 Alcance y limitaciones

A continuación se describe el alcance del proyecto y posteriormente se definen las principales limitaciones y dificultades para su desarrollo.

2.5.1 Alcance

Como se ha mencionado en el capítulo 1, el objetivo de este proyecto es la implementación del componente de búsqueda de caminos y control de movimiento de los agentes de un videojuego RTS; cabe indicar que este componente será parte de un proyecto mayor, ya que la gestión de los mecanismos de la lógica del videojuego así como el manejo de los recursos gráficos será parte de otros proyectos de fin de carrera. Sobre el componente mencionado, se busca que sus clases y métodos desarrollados puedan ser adaptables a cualquier videojuego que cumpla con las reglas descritas en la sistematización de los mecanismos de interacción de personajes para un videojuego de naturaleza RTS, las cuales se describen en el capítulo 0.

La implementación del componente contará con un gestor de mapas, el cual pueda interpretar las estructuras que funcionarán como áreas de búsqueda para los algoritmos, dichas estructuras serán cargadas de archivos de configuración e interpretadas por el componente como un grafo representado por una grilla cuadriculada; cabe indicar, que este gestor no creará los archivos de manera automática.

Se contará también con un módulo que se encargue de obtener el camino más corto entre dos nodos de un grafo; dicho módulo utilizará la adaptación de los algoritmos de búsqueda de caminos A* y HPA*, además se demostrará mediante

métodos de experimentación numérica la notable mejora que proporciona el algoritmo HPA* en términos de tiempo y uso de espacio de memoria de ejecución sobre el A* para ciertas circunstancias de búsqueda, indicando en qué casos el módulo utilizará cada uno de los algoritmo.

Acerca del módulo de desplazamiento e interacción de los personajes, este permitirá que los agentes se desplacen en ocho direcciones por un mapa isométrico en dos dimensiones con un ángulo de inclinación de 30 grados, se debe considerar que el caso de movimiento tres dimensiones esta fuera del alcance del proyecto, además seguirán el camino hallado por los algoritmos de búsqueda, evadiendo los obstáculos estáticos que se hallan en su camino. Respecto a los obstáculos dinámicos, el módulo de movimiento verificará que no haya colisiones con el resto de personajes; sin embargo, vale indicar que el manejo de las evasiones es parte de un componente de la física del videojuego, el cual no está dentro del alcance de este proyecto.

Las principales limitaciones de estos algoritmos están relacionadas al tamaño del mapa, es decir la cantidad de nodos del área de búsqueda así como de la cantidad de unidades que se desplacen en el mapa, pues soportará una distancia de búsqueda máxima entre el punto inicial y final del mismo modo; sólo se podrá ejecutar el algoritmo de búsqueda en un mismo instante para un número limitado de personajes.

2.5.2 Limitaciones y riesgos

Dado que el último de los objetivos es adaptar el componente a un prototipo del videojuego, en caso que uno de los os desarrolladores que vienen implementando otros componentes del mismo, como por ejemplo, la elaboración de los algoritmos que permitan controlar el flujo del juego y sus reglas, o el manejo de recursos gráficos se retrasaran, se provocaría un retraso para poder cumplir con dicho objetivo, además es necesario asegurarse de la compatibilidad de los distintos componentes.

Otra limitación es que si bien hay bastante información para la implementación de algoritmos de búsqueda de camino, no sucede lo mismo con los control de movimiento e interacción de los personajes, por lo tanto no existe un estándar para su implementación, esto se debe principalmente a que la empresas desarrolladoras

de videojuegos, suelen ser propietarias de sus propios motores de videojuegos, y por temas comerciales no muestran su código fuente.

La gestión de riesgos, esta se encuentra en el Anexo 1.

2.6 Justificación y viabilidad

A continuación, se presenta la viabilidad y justificación del proyecto desarrollado.

2.6.1 Justificación del proyecto de tesis

Este proyecto será de ayuda para quienes empiezan a insertarse en la industria de desarrollo de videojuegos o aplicaciones similares y que requieran emplear este tipo de algoritmos. Esto incluye a desarrolladores de videojuegos, así como alumnos de la especialidad.

Un valor importante del proyecto, es que permitirá demostrar cuando un componente de búsqueda de caminos es eficiente y por lo tanto, utilice los algoritmos adecuados en cada caso particular. Siempre considerando los cuatro criterios establecidos: Optimalidad, tiempo de respuesta, uso de memoria y completitud.

También permitirá documentar y describir de manera clara, las características que debe tener un mecanismo de navegación autónomo en videojuegos RTS y como fue implementado, por lo que futuro puede ser utilizado como punto de partida para nuevas implementaciones, que permitan mejorarlo y darle algún valor agregado volviéndolo más eficiente y agregándole nuevas funcionalidades.
Su valor práctico es que el producto que se obtendrá permitirá implementar los algoritmos de búsqueda de caminos y control de movimiento que posteriormente serán adaptados al videojuego de estrategia en tiempo real “1814 La Rebelión de Cuzco”, que viene desarrollando el grupo Avatar en la PUCP (AVATAR, 2012).

2.6.2 Análisis de viabilidad del proyecto

A continuación se analiza la viabilidad del proyecto, desde tres perspectivas: la viabilidad técnica, temporal y económica.

- **Viabilidad Técnica**

El componente ha sido desarrolla en Java, un lenguaje de programación de alto nivel utilizado para el desarrollo de aplicaciones web, videojuegos, aplicaciones móviles y de escritorio, entre otras; además, la aplicación será soportada tanto por Windows como por Linux gracias a su máquina virtual JVM el cual permite que las aplicaciones Java puedan ser ejecutadas en distintos entornos y sistemas operativos (ORACLE, 2013).

- **Viabilidad Temporal**

Para este proyecto, se cuenta 5 meses para el desarrollo del producto; sin embargo, este empezó a desarrollarse con anticipación, y por lo tanto se estimó que trabajando un promedio de 3 horas al día, se podría acabar en menos de los 5 meses.

- **Viabilidad Económica**

Todas las herramientas que se usen, tanto el lenguaje de programación, el entorno de programación son de uso gratuito, además se cuenta con los libros en formato virtual y físicos necesarios para el desarrollo del mismo, por lo que se espera que esto no sea un factor en contra.

3. Capítulo 3. Descripción sistematizada del Videojuego RTS

En este proyecto se plantean tres aspectos básicos que poseen los videojuegos de este género, y en particular el videojuego “1814, La rebelión del Cuzco” (AVATAR, 2013): La gestión del entorno de desplazamiento o mapa, la búsqueda del camino ideal en cada situación del juego y el comportamiento de los personajes del videojuego.

3.1 Elementos y características del videojuego

A continuación se presenta la descripción de cada uno de los distintos elementos y características que componen el videojuego, los cuales son importantes en la implementación de los módulos mencionados.

3.1.1 Discretización del mapa

La discretización es un concepto matemático, el cual consiste en transformar ecuaciones o modelos continuos a modelos discretos, este concepto es fundamental en los videojuegos RTS, ya que la pantalla en la cual se muestran los

mapas está dividida en miles de pequeñas unidades llamadas pixeles (posiciones físicas en la pantalla).

La implementación de un algoritmo de búsqueda en un grafo con tantos vértices, es ineficiente; por tanto, mediante la discretización del mapa, podemos generar un grafo lo suficientemente grande para mantener la exactitud en los desplazamientos, pero a la vez no tan grande como para afectar el tiempo de ejecución de los algoritmos.

En el proyecto se ha seleccionado el modelo de grillas cuadriculadas, de tal forma que, cada espacio del mapa en el videojuego estará representado de manera discreta en un archivo de configuración.

En el anexo 5, se observa un ejemplo de cómo se realiza la discretización del mapa mediante dos archivos un archivo de texto para la jerarquía superior y otro para la jerarquía inferior, ambos estarán compuestos de caracteres que representan distintos tipos de terrenos transitables y no transitables, que permitirán identificar las posiciones lógicas de los personajes mediante una conversión isométrica.

3.1.2 Atributos de los personajes

A continuación se describen las principales características que tendrán los personajes del videojuego.

- ***Velocidad de movimiento:***

La velocidad de movimiento es el factor que indica a cuanta velocidad deberá desplazarse cada personaje; en base a este factor y a su dirección, se calcula cuál será su próxima posición en el mapa.

- ***Rango de ataque:***

El rango de ataque indica la distancia máxima y mínima en la cual un personaje puede atacar a otro del bando contrario; es decir, para que el ataque sea efectivo, el personaje objetivo debe estar en un radio menor que el rango máximo de ataque y mayor que el rango mínimo.

- ***Rango de visión:***

El rango de visión de cada personaje, les permite detectar a cualquier unidad del bando contrario, en un rango determinado.

- ***Velocidad de Ataque:***

Este atributo establece cada cuánto tiempo un personaje realizará un ataque a otro del bando contrario y que esté dentro de su rango de ataque.

- ***Vida:***

Contabiliza los puntos de vida que posee cada personaje. Cuando este llega cero, el personaje muere y es eliminado del bando.

- ***Daño:***

El daño establece la cantidad de puntos de vida que resta cada ataque de un personaje a otro del bando contrario.

- ***Nivel:***

El nivel determina la cantidad de experiencia acumulada de cada personaje. Conforme se incrementa el nivel del personaje, se mejora los otros atributos.

3.1.3 Tipos de personajes

En los RTS se enfrentan dos o más bandos, formados por un conjunto de personajes de distintos tipos y características; sin embargo, generalmente se agrupan en tres géneros principales: Héroes, personajes de rango y de melee.

- ***Personajes de Rango:***

Los personajes de rango se caracterizan porque pueden atacar a distancia; sin embargo, de acuerdo al tipo de personaje de rango que sean, contarán con un radio máximo y mínimo de ataque, lo cual los obliga a retroceder cuando un personaje

del bando contrario se le aproxima a una distancia menor al radio mínimo establecido.

- ***Personajes de Melee:***

Los personajes del tipo melee no atacan a distancia, deben aproximarse al objetivo a una distancia corta. Estos pueden ser infantería, artillería, caballería, entre otras, de acuerdo al tipo de personaje de melee que sea, se les asigna una velocidad de movimiento y ataque.

- ***Héroes:***

Los héroes, pueden del tipo rango como melee, estos se caracterizan porque van ganando experiencia en las batallas lo cual les permite incrementar su nivel. En incremento de nivel en cualquier unidad significa una mejora en sus habilidades, tales como velocidad de movimiento y de ataque, o su rango máximo de ataque. Debido a esto es necesario considerar que el tipo de movimiento y ataque de cada personaje debe ser configurable.

3.2 Personajes como agentes de movimiento

Cada personaje será tratado en adelante como agentes de movimiento en un entorno, los cuales podrán moverse en ocho direcciones. Los agentes siempre parten de una posición lógica de origen, y buscan llegar a una posición lógica destino, estos pueden desplazarse tanto de manera independiente como en conjunto.

Se observa también que cuando el desplazamiento es colectivo, estos pueden desplazarse manteniendo una formación, o colocándose cada agente uno detrás del otro.

El proceso de navegación de los agentes consta de cuatro etapas:

- La selección de los agentes que se van a desplazar
- La búsqueda del camino optimo
- La ejecución del desplazamiento a partir del camino hallado
- La interacción constante de los agentes con su entorno

3.2.1 Selección de agentes de desplazamiento

La selección de los agentes controlados por el jugador se realiza dejando presionado el botón izquierdo del mouse, y arrastrándolo sobre el conjunto de agentes que se desea seleccionar, también puede realizarse la selección de un solo agente, en la ilustración 18, se observa la operación de selección.

En el caso del desplazamiento de los agentes controlados por la máquina, la selección de los personajes a desplazarse, se realiza en función de la situación actual del entorno de los agentes, buscando que la maquina realice acciones que la beneficien.

3.2.2 Búsqueda del camino optimo

Una vez seleccionado los agentes que se desplazaran, continúa la etapa de buscar el camino óptimo que deben seguir el o los agentes para llegar al destino seleccionado. Encontrar dicho camino no es tarea fácil, pues es necesario encontrar un balance entre la velocidad de respuesta del CPU y la precisión del camino seleccionado.

El algoritmo que se encargará de la búsqueda de dicho camino debe considerar las zonas transitables y no transitables establecidas en el archivo mapa.txt, el cual contiene las posiciones lógicas del mapa.

Cada posición lógica del archivo, será procesada como un nodo en un grafo, por motivos de eficiencia, al desplazar varios agentes que se encuentren cercanos, estos se juntan formando un batallón de desplazamiento, el cual tendrá en memoria un mismo camino, conformado por un conjunto de posiciones lógicas las cuales se guardaran en una pila de la memoria.

3.2.3 Ejecución del desplazamiento

La tercera etapa consiste en ejecutar el desplazamiento de cada personaje seleccionado, para esto, se van extrayendo los valores de la pila de posiciones lógicas que conforman el camino, y se empieza a ejecutar el recorrido, cambiando las posiciones absolutas de cada uno de los personajes y actualizando sus

direcciones; es decir, esta etapa se ejecuta tanto a nivel de posiciones físicas como lógicas.

3.2.4 Interacción de los agentes de movimiento

Los agentes son autónomos y deben procesar información de su entorno para calcular la acción a realizar.

En cada instante del juego, evalúan la información que poseen de su entorno para calcular la acción que deben realizar, de tal manera, deciden si deben atacar a algún personaje en particular, huir, perseguir o mantener la posición actual.

3.2.5 Principios de un agente autónomo

Para que un agente sea completamente autónomo, son tres los principios que debemos considerar, a continuación se detallan cada uno de ellos.

- **Limitación para percibir su entorno:**

Esto significa que cada agente debe estar programado de tal manera que tenga referencia de lo que sucede en su entorno; sin embargo, esta percepción debe estar limitada a una zona y no tener la información de todo el entorno de manera absoluta.

- **Procesamiento de información del entorno para calcular acción:**

Cada agente, tendrá conocimiento de su entorno, es decir los obstáculos, personajes aliados y oponentes que estén próximos a él, de tal manera que en cada situación del juego, podrán tomar una decisión en base a este conocimiento.

- **No tienen un líder:**

Cada agente seguirá su camino determinado, y las acciones que realice serán en base a la información que procese de su entorno; es decir no ejecuta acciones siguiendo a un líder, sino de acuerdo al conocimiento del entorno que posea.

4. Capítulo 4. Diseño de la Arquitectura

El capítulo describe la arquitectura de software que se eligió en la implementación de la solución, para lo cual como se ha indicado en el apartado 0, se empleó la metodología ágil XP. Dicha metodología no exige la presentación entregables específicos de la arquitectura, salvo las historias de usuarios; solamente sugiere que se presenten los entregables que se consideren pertinentes para el proyecto (Extreme Programming, 2013).

4.1 Arquitectura de la solución

Los videojuegos modernos son proyectos en ingeniería de software bastante complejos, los cuales pueden llegar a tener millones de líneas de código fuente. Por lo tanto, es necesario el diseño de patrones arquitectónicos que permitan la reutilización de código, tal cual se indica en la ingeniería de software (GREGORY, 2009).

Es importante resaltar que el diseño de la arquitectura planteada para el producto, se enfocará en mostrar los patrones arquitectónicos del videojuego a alto nivel.

Luego, se especificará con mayor detalle el diseño arquitectónico del componente de IA.

Para el diseño de la arquitectura se utilizó como referencia el modelo de vistas 4+1 de Kruchten, el cual cumple con el estándar IEEE 1471-2000 (IEEE, 2013). Dicho modelo permite diseñar la arquitectura desde distintos puntos de vista, las cuales son: Vista de procesos, lógica, física, de desarrollo; y por último, la vista de escenarios conocida como la vista +1; se le conoce así, por ser la que integra al resto de las vistas (KRUCHTEN, 2013). Además, se utilizó UML como herramienta para el modelado y la elaboración de los entregables. En este proyecto no se detallará la vista física, la cual describe la interacción entre los distintos dispositivos de hardware, siendo esta útil para el diseño de sistemas cliente servidor o sistemas distribuidos y no para el tipo de producto desarrollado en este proyecto.

4.1.1 Vista de escenarios.

La vista de escenarios describe las principales funcionalidades que tiene el software. Basados en la metodología ágil aplicada (XP), el entregable que se obtendrá como resultado para la documentación será las historias de usuario (ver anexo 2). En ella se describen las distintas funcionalidades con las que contará el componente implementado.

4.1.2 Vista Lógica

La vista lógica describe el diseño del modelo de clases, cuando se sigue un diseño orientado a objetos, como es el caso. Además muestra la interacción de los distintos elementos, los cuales dan soporte a los requerimientos funcionales. A continuación se muestran las clases principales de este componente.

Para la representación de esta vista además, se utilizó un diagrama de clases con sus plantillas de clase, las cuales se pueden apreciar en sus respectivos entregables (ver anexo 3 y 4 respectivamente).

4.1.3 Vista de procesos

La arquitectura desde la vista de procesos muestra la interacción de los distintos elementos descritos en la vista lógica, de tal manera que, muestra cómo se ejecutan las distintas tareas mediante la interacción de clases. Esta vista toma en cuenta algunos requerimientos no funcionales, tales como: Rendimiento, disponibilidad, integridad y tolerancia a los fallos.

Rendimiento e integridad

Los procesos principales de la solución son la búsqueda de camino regular y la búsqueda de caminos jerárquica, ambos son parte del componente de IA, dichos procesos son invocados por el hilo principal del juego y posteriormente actualizan los valores del sistema de objetos, el concepto de sistema de objetos se explicará más adelante. Es fundamental para un videojuego del tipo RTS, los tiempos de respuesta de los algoritmos, puesto que parte de su naturaleza implica la fluidez del juego en tiempo real, por lo tanto, se espera que estos procesos tengan un rendimiento óptimo. El rendimiento óptimo hace referencia tanto al tiempo de ejecución como a la precisión del resultado. En el capítulo 0 se valida la optimalidad de estos procesos.

Tolerancia a fallos y disponibilidad

Otra característica deseada en el diseño es contar con un nivel de tolerancia a fallos aceptable. Por lo tanto, si se logra un diseño en el cual cada componente tenga un nivel de independencia considerable, en caso suceda un fallo en algún proceso de este, no deberían verse afectados el resto de componentes, de tal manera que no se interrumpa el flujo del videojuego.

Hay que considerar sin embargo, que a pesar del grado de independencia de los componentes, siempre pueden suceder errores o bugs debido a alguna situación no considerada; por esto será necesario que todo segmento de código crítico cuente con sus respectivos métodos para la captura y el manejo de errores; estos pueden ser del tipo: Errores de jugador o errores de programación. Los primeros deben manejarse alertando al jugador del error que ha cometido, por ejemplo, cuando se intenta que un agente se desplace a un lugar no transitable; por otro lado, los errores de programación deben ser manejados mediante métodos de capturas de

errores como códigos de retorno o excepciones, de tal manera que se garantice siempre la continuidad del videojuego (GRIGORY, 2009).

4.1.4 Vista de desarrollo

En esta vista se definen los patrones arquitectónicos del software, desde un punto de vista de los componentes implementados, es decir desde el punto de vista del programador. Debido a que el software implementado no es un sistema de información, sino un videojuego, se buscó como referencia un patrón arquitectónico de software aplicado a videojuegos, en particular a los del tipo RTS. Al respecto, Michael Doherty propone un patrón arquitectónico que cumple con principios estándar de los videojuegos RTS (DOHERY, 2003), el que será utilizado como base para el diseño de esta vista.

4.2 Patrones arquitectónicos de la solución

La arquitectura propuesta está diseñada pensando en maximizar la reutilización de recursos y buscando la adaptabilidad que sugiere XP, para esto, se separaron los componentes generales de los específicos del videojuego. Finalmente se utiliza una estructura centralizada denominada sistema de objetos, la cual permite reducir el acoplamiento entre los componentes de la arquitectura (DOHERTY, 2003).

En la Ilustración 4.1 se observa la interacción de los componentes al alto nivel. Ahí se distinguen cuatro componentes principales: El gestor de datos, el motor del video juego, el sistema de objetos, y un conjunto de recursos que sirven de apoyo para el videojuego.

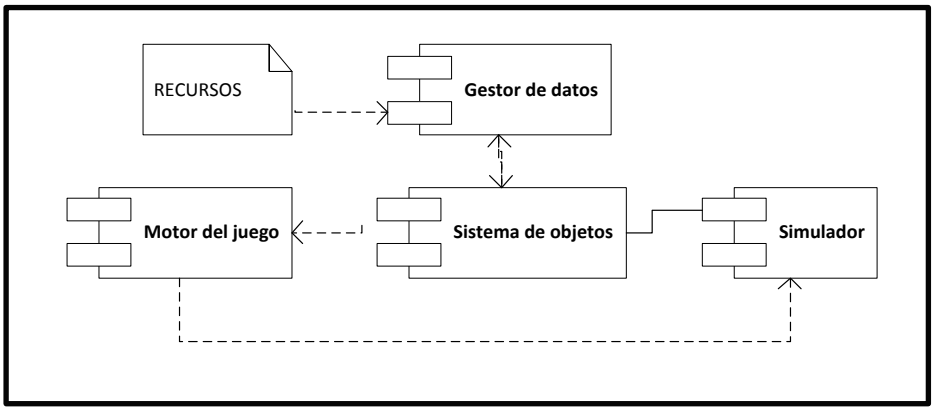

Ilustración 4.1. Componentes a alto nivel (adaptado de Doherty, 2003)

A continuación en los siguientes apartados, se explicará con mayor detalle, el diseño y funcionamiento del sistema de objetos el simulador y el motor del videojuego; sin embargo, es importante indicar que la implementación del motor del videojuego esta fuera del alcance de este proyecto, aunque evidentemente se hace uso continuo de él.

4.2.1 Gestor de datos

Es el encargado de proveerle al sistema de objetos los datos del sistema de archivos u otros elementos de almacenamiento que son parte de los recursos; tales como, los archivos de configuración de los personajes y de los mapas, cuya estructuras se explican en el apartado 4.4.

4.2.2 Sistema de objetos

Como se ha indicado, Doherty propone un diseño de arquitectura centralizado, teniendo como núcleo al sistema de objetos, el cual a su vez funciona como una interfaz para la comunicación del resto de componentes de alto nivel. Básicamente su función es la de dar mantenimiento a los datos y actualizar la información del videojuego; por otro lado, de la manera en la que se implemente impactará en la flexibilidad que tenga el videojuego para soportar futuros cambios (DOHERY, 2003). Al estar centralizado hacia objetos, cuando se desee acceder a los datos, primero debe buscar al objeto que contenga dichos datos y posteriormente la información relevante de los atributos contenidos por el objeto hallado.

4.2.3 Motor del juego

El motor del juego, puede ser dividido en un conjunto de subsistemas; los cuales, no siempre son nombrados de la misma manera; sin embargo, como resultado de una observación empírica podemos encontrar algunos de los siguientes componentes en cualquier videojuego: Gestor gráfico, de audio, redes, y entrada de datos. (LOKI, 2001). La siguiente ilustración, muestra la interacción de los componentes que forman el motor del videojuego.

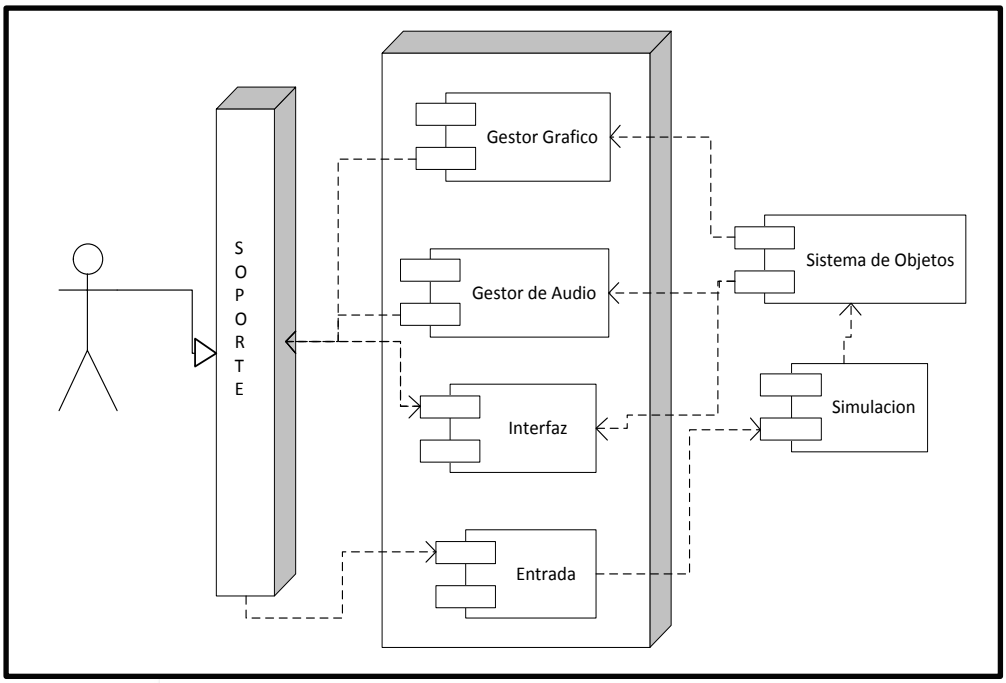

Ilustración 4.2. Diagrama de componentes del motor del juego (Adaptado de Doherty, 2003)

Es necesario que cada componente tenga una interfaz bien definida, de tal manera que pueda ser fácilmente intercambiado con una librería similar de ser necesario.

El motor del juego es la interfaz encargada de interactuar con los dispositivos de entrada y salida del computador. Este recibirá instrucciones de los dispositivos de entrada, se comunicará con el sistema de objetos y del componente de simulación, luego recibe un resultado del sistema de objetos, y mediante el gestor gráfico mostrará de manera visual los cambios sucedidos.

4.2.4 Componente de Simulación.

El componente de simulación es el corazón del videojuego, ya que este define como se verá y funcionará el mismo, además este componente de alto nivel contiene a los componentes que serán parte de la solución que se implementará en este proyecto, en particular las del componente de IA, y control de la simulación.

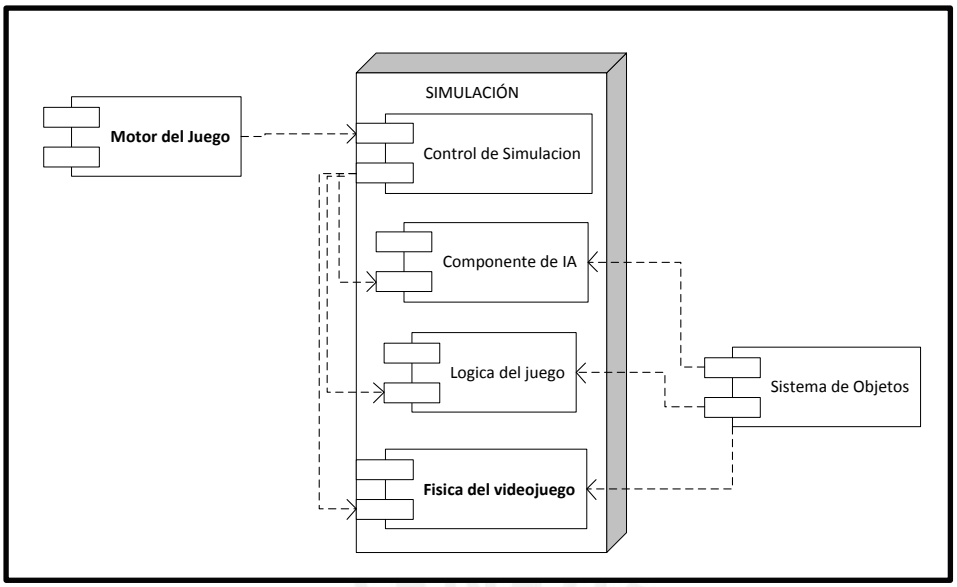

Ilustración 4.3. Componente de simulación (Adaptado de Doherty, 20013)

4.2.5 Componente de IA del jugador

Uno de los componentes más importantes en el desarrollo de videojuegos es el de IA, como se observa en la ilustración 22, está compuesto por cuatro componentes, los cuales interactúan entre sí a manera de cascada. En un primer nivel se encuentran las reglas propias del videojuego, posteriormente las máquinas de estado finito son las que permiten caracterizar el comportamiento de cada personaje, determinando así, la acción que deba realizar; dichas acciones suelen implicar un desplazamiento, por lo que se recurre al componente de Pathfinding.

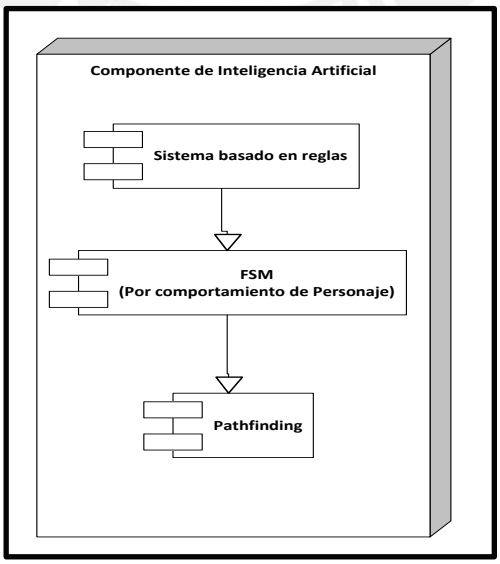

Ilustración 4.4. Componente de IA (adaptado de Millington, 2009)

Los métodos de las clases de este componente, tanto para la búsqueda de caminos como para la interacción de personajes, pueden ser invocados de dos maneras; porque el jugador ejecuta un comando o por una reacción automática ante una situación determinada por el estado actual del videojuego.

4.3 Funcionamiento de la arquitectura diseñada

Como se detalló en el apartado 3.2, el mecanismo completo de búsqueda de camino y movimiento e interacción de los personajes que se desea implementar, consta de cuatro etapas claramente diferenciadas. En los siguientes apartados se detallará de qué manera la arquitectura planteada permite implementar dichas etapas, las cuales son:

- La selección de los agentes que se van a desplazar
- La búsqueda del camino óptimo
- La ejecución del desplazamiento a partir del camino hallado
- La interacción constante de los agentes con su entorno

A continuación, se explicará mediante un ejemplo la interacción de los distintos componentes descritos en los apartados anteriores, a fin de lograr implementar el mecanismo descrito.

Supongamos que el jugador selecciona un conjunto de personajes utilizando el mouse, luego presiona el botón derecho del mouse sobre alguna zona del terreno del mapa para que se desplacen los personajes. El funcionamiento de los componentes de la arquitectura seria el siguiente:

Primero el componente de entrada del motor de juego recibe la información del mouse; luego el componente de entrada se comunica con el componente lógico del simulador y realiza las conversiones isométricas pertinentes para calcular la coordenadas lógicas correspondientes al área de selección; tras esto, el componente lógico informa de dicha área al sistema de objetos, el cual actualizará los estados de los personajes que se encuentren en dicha área, al estado: ´Seleccionado´. Luego, el sistema de objetos comunica al gestor gráfico que ya se seleccionaron algunos personajes. Finalmente el gestor gráfico se encarga de dibujar en pantalla un aro debajo de los personajes que tengan el estado ´Seleccionado´, y lo muestra visualmente en la pantalla.

Posteriormente el jugador presiona el botón derecho del mouse en alguna zona transitable del mapa; siguiendo los mismos pasos mencionados anteriormente, se le informa al sistema de objetos cual es la posición destino, y este le colocará dicho valor al atributo posición destino de los personajes que se encuentren con el estado ´Seleccionado´. Luego es el turno del componente de IA, el cual por medio del sistema basado en reglas verifica la acción que debe realizar; como detecta que en el sistema de objetos hay personajes que se encuentran en una posición lógica, la cual es distinta a su posición lógica destino, por lo tanto concluye que estos deben desplazarse. Luego el FMS verifica la acción desplazar de acuerdo a las características del personaje leídas del sistema de objetos. Después es necesario determinar el camino que deben a seguir y por lo tanto el componente de Pathfinding ejecutara los algoritmos de búsqueda de caminos e informará al sistema de objetos del camino (conjunto de celdas) que deberá seguir cada personaje para poder llegar al destino.

Como es evidente, hasta el momento los personajes tienen ya trazado un destino al que deben llegar y el camino que deben seguir; sin embargo, el desplazamiento aún no ha finalizado, puesto que el movimiento se ejecuta por partes y el personaje va avanzando algunos pixeles en la posición física en cada iteración.

4.4 Archivos de configuración

En la Ilustración 4.1, se observa que el componente gestor de datos se encarga de recibir datos de configuración, por medio del componente de recursos. Dentro de dicho componente se encuentran los archivos de configuración de los mapas y bandos del videojuego, a continuación se explican la configuración de dichos archivos, que serán clave para el funcionamiento de los algoritmos de búsqueda de caminos y el de movimiento e interacción de los personajes.

4.4.1 Mapas: Representación de los estados del espacio de búsqueda

Los mapas se estructuran mediante los archivos de configuración: mapaNivel.txt. Ilustración 4.6. Los estados que tengan el área de búsqueda, serán cargados en la estructura Mapa a partir del conjunto de caracteres de los archivos mapaNivel.txt. A continuación el significado de cada uno de los distintos estados.

Ilustración 4.5 Mapa de Configuración de archivos (Creado por el autor)

- O: No pertenece al mapa
- T: Terreno transitable
- E, Y, Z, R, W: Edificaciones del mapa
- A, B, C: Zona de Eventos
- U: Terreno transitable (zona de sombra, detrás de las edificaciones).

4.4.2 Bandos de Personajes

La clase Juego contendrá los siguientes bandos: bando del jugador, de la máquina y el bando neutral, cada uno de estos bandos es llenado al comenzar cada nivel del juego o al recuperar partidas guardadas, a través de la clase GuardableNivel.java. Los archivos se encontraran en la siguiente ruta: \Configuración\NivelN\ConfCambioNivel\Personajes. La estructura que siguen dichos archivos se muestran en el Anexo 8.

5. Construcción del módulo de búsqueda de caminos

En el presente capítulo, se explicará el proceso que se siguió en la construcción del módulo de búsqueda de caminos.

Para lo cual, primero se describen las principales consideraciones que se deben tener al implementar los algoritmos de búsqueda, tales como: la determinación de los entornos de búsqueda, el rendimiento en tiempo de ejecución del algoritmo y la ocupación de memoria, dichas características permitirán justificar la selección del algoritmo.

Luego se detallará la manera en la que se construyó y adapto cada uno de los algoritmos, indicando cada una de sus respectivas estructuras, así como la interacción de las mismas.

5.1 Construcción del entorno de búsqueda

El primer paso que se efectuó en la construcción del módulo, fue la creación del entorno de búsqueda mediante los archivos de configuración de los mapas

(descritos en el capítulo 4), los cuales como se mencionó en el apartado 4.3, son utilizados para representar al grafo donde se ejecutarán los algoritmos de búsqueda.

Para esto se empleó el método de grillas cuadriculadas, de tal manera que mediante fórmulas de conversión isométrica se realizó el mapeo de la imagen de un mapa con un ángulo de inclinación de 30° a un archivo de texto de configuración discreto, de tal manera que cada carácter del archivo represente un nodo del grafo de búsqueda.

5.1.1 Entorno de búsqueda estático

Los entornos de búsquedas pueden ser clasificados en distintos tipos en función de la naturaleza del videojuego.

Existen videojuegos con entornos dinámicos los cuales se caracterizan porque su entorno varia una vez que ya se determinó el camino y por lo tanto es necesario recalcularlo; sin embargo, para el presente proyecto se considerará un entorno estático, el cual se caracteriza por tener una navegación más sencilla ya que el entorno es predecible en todo momento (REESE, 1999).

Cabe destacar que los únicos factores dinámicos en este tipo de entornos serán los propios agentes que se desplazan en él; sin embargo, estos no son triviales los algoritmos de búsqueda es por esto que para los algoritmos de control de movimiento e interacción de los personajes, los cuales se detallan en el capítulo 0, su entorno sí será dinámico.

5.1.2 Construcción de los mapas de configuración

Para la construcción de los archivos de configuración de los mapas, primero se estableció dos sistemas de coordenadas: el sistema de coordenadas físicas y el sistema de coordenadas lógicas, a continuación se explicará el uso de cada uno de dichos sistemas.

- **Sistema de coordenadas físicas y lógicas**

El sistema de coordenadas lógicas es determinado en proporción a la altura y el ancho que tendrá cada casilla de la grilla, de tal manera que cada posición del sistema de coordenadas lógico estará compuesta de varias posiciones físicas.

Ambos sistemas son importantes para la aplicación de los distintos algoritmos; sin embargo, para el algoritmo de búsqueda de caminos, el cual es tratado en este capítulo, solo importa el sistema de coordenadas lógicas, esto se debe a que la complejidad del detalle que pueda manejarse a nivel de pixeles debe ser transparente para el algoritmo, ya que a este solo le interesa el grafo de nodos transitables y no transitables que se obtiene producto de la abstracción del mapa.

El uso del sistema de coordenadas físicas se explicará en el siguiente capítulo con la implementación de los algoritmos de movimiento e interacción de los agentes.

- **Estructuras para la gestión de los entornos o mapas**

Utilizando las funciones isométricas mencionadas en el apartado anterior, se realizó el mapeo de las coordenadas físicas a lógicas, de tal manera que el sistema de coordenadas lógicas se representa en un archivo de configuración, para esto, se le asigna a cada posición lógica un carácter en el archivo, dicho carácter indicará si la posición lógica es o no transitable, tal y como se indica en apartado 4.3.1.

Posteriormente cada uno de los archivos de configuración son leídos y asignados a una lista de objetos Mapa, la cual es contenido por el objeto Nivel. Finalmente la clase Gestora de niveles contendrá al conjunto de niveles, además, será el encargado de proporcionar al bucle principal del juego la estructura del nodo o mapa que será utilizada por el algoritmo de búsqueda.

Es importante destacar que cada Nivel podrá contener varios mapas, pues un mismo nivel puede distintos escenarios, tanto interiores como exteriores y por lo tanto es necesario que el gestor de niveles informe al algoritmo cual es el grafo o estructura mapa indicado sobre el que se debe realizar la búsqueda.

5.1.3 Consideraciones de tiempo de ejecución y memoria

Como se ha explicado en la problemática, para la selección del algoritmo de búsqueda de caminos que se implementará, es necesario tener en consideración que en un videojuego RTS son muchos los agentes que requieren desplazarse en el mismo instante, y por lo tanto, si el algoritmo ocupa mucho espacio en la memoria o su tiempo de ejecución es excesivo, se producirá retardos en la fluidez del videojuego.

La acción de planificar el camino es una operación que se ejecutará periódicamente en la fase de actualización de cada uno de los agentes del videojuego, cuando estos requieran desplazarse. Por lo tanto, es necesario utilizar de manera adecuada la memoria, y tener tiempos de ejecución aceptables, en los siguientes apartados se explicará en qué casos es preferible utilizar el algoritmo A* y cuando el HPA* para obtener un tiempo de ejecución adecuado.

5.1.4 Criterios para selección del algoritmo

Los principales criterios que permiten evaluar cuál será el algoritmo apropiado serán: optimalidad, tiempo de respuesta y ocupación de memoria, los cuales han sido definidos en el marco conceptual **(REESE, 1999).**

La memoria no suele presentar mayores problemas en el caso de entornos estáticos, como si lo hace en los entornos dinámicos; sin embargo, cada vez que la búsqueda involucre explorar una mayor cantidad de nodos, requerirá de mayor espacio de memoria. Finalmente el tiempo de ejecución es el criterio clave para la elección del algoritmo pues un videojuego RTS a diferencia de uno basado en turnos no debería presentar demoras en su ejecución.

Es por esto que la selección de una heurística apropiada, acorde a las necesidades del videojuego, así como la utilización de los algoritmos A* y A* jerárquico o HPA*, garantizará que se cumplan los criterios establecidos.

5.2 Algoritmo A*

Tal y como se describe en el estado del arte, el algoritmo A* se diferencia del bastante utilizado algoritmo de Dijkstra por emplear una heurística la cual permitirá

obtener una solución, que si bien será menos exacta, se hallará en un menor tiempo.

A continuación, se explicará las principales características del algoritmo así como el proceso de su adaptación al componente de búsqueda de caminos.

5.2.1 Consideraciones del algoritmo

- Interfaz del algoritmo

El algoritmo A* recibe como entrada de datos (input) el nodo inicial, el cual es la posición lógica actual del personaje que se desea desplazar, también recibe el nodo final, que es la posición a donde se quiere dirigir, además, toma el entorno de búsqueda de la estructura Mapa.

Finalmente el algoritmo devolverá un conjunto de nodos o posiciones lógicas en una estructura de Pila. Es importante destacar que la estructura debe ser una pila pues como se explicará en los siguientes apartados la manera en la que se extraen los nodos de la estructura devuelta por el algoritmo cumple la característica de una estructura de datos LIFO, es decir el último nodo en entrar a la estructura será el primero en salir.

- **Estructuras de datos**

    - **Lista abierta LA**: Contiene los nodos que podrían o no ser parte de la solución, deben ser evaluados.

    - **Lista cerrada LC**: Contiene los nodos que por el momento no necesitan ser evaluados, aunque pueden o no se parte de la solución.

    - **Lista de vecinos LV**: Es la lista temporal que contiene los nodos contiguos al nodo a explorar.

- **Fórmula para el cálculo del costo de los nodos**

El algoritmo se basas principalmente en la fórmula que se describe a continuación, esta permite valorizar los costos de desplazamiento en función de la distancia

Entre el nodo de evaluación con los nodos de inicio y de fin.

$\mathbf{F = G + H}$, donde

G es la distancia del nodo inicial (NI) al nodo a evaluar (NE)

H es la distancia heurística

5.2.2 Función Heurística

De la fórmula de costo observamos conforme aumente o disminuyan los valores G y H, esto impactará en el valor F del nodo, por lo tanto de acuerdo a la heurística que elijamos, podremos modificar el comportamiento del algoritmo A*, de tal manera que cumpla o no con los criterios optimalidad y tiempo de respuesta mencionados en la problemática.

Esta habilidad de variar el comportamiento del algoritmo A* basado en la heurística y la función de costo, suele ser muy útil en la implementación de videojuegos RTS, además generalmente no se requiere el camino más corto posible, sino uno que se le aproxime (STANDFORD, 2013).

Las heurísticas para mapas en forma de grillas están claramente definidas, en caso de que la grilla permita el movimiento en 4 direcciones, se recomienda usar la distancia Manhattan, por otro lado, para las grillas que cuenten con 8 direcciones se recomienda usar la distancia diagonal, también llamada distancia Chebyshev (STANDFORD, 2013). Por lo tanto en el presente proyecto, se utilizará la distancia Chebyshev como heurística en el algoritmo.

5.2.3 Adaptación del algoritmo al componente

Como se mencionó en la descripción sistematizada, específicamente en el apartado 3.2, los pasos necesarios para mover un agente de un punto inicial a un punto final en su entorno son cuatro: selección de agentes a desplazar, búsqueda del camino, la ejecución del desplazamiento y la interacción con los otros agentes. De tal manera que la secuencia del funcionamiento del componente es la siguiente:
Los eventos del mouse del componente de entrada y salida detectan que se seleccionó un conjunto de agentes y presionó el botón derecho del mouse en

alguna posición física del mapa, este a su vez realiza la conversión mediante las formulas isométricas y determina la posición lógica del mapa en la que se presionó el botón del mouse, luego el bucle principal, definido en la arquitectura de la solución como hilomovimiento, se encarga de verificar que agentes deben desplazarse a la posición lógica indicada, y empieza a invocar al algoritmo de búsqueda de caminos para cada personaje.

Tras esto, el bucle principal se encarga de invocar a la clase PathFInder cuando así se requiera, Finalmente la clase Pathfinder devuelve la pila con el conjunto de posiciones lógicas determinadas como parte de la solución del mejor camino.

Los algoritmos A* y HPA* son clases del componente de búsqueda de caminos, sus datos de entrada y de salida son los que están definidos en el punto 5.2.1. Dicho método trabaja en conjunto con los métodos reconstruyeCamino, devuelveVecinos, menorFScore y disxtanciaHeuristica, el funcionamiento de cada uno de estos métodos se puede observar en el siguiente apartado mediante un pseudocódigo.
En el siguiente capítulo donde se detalla el módulo de movimiento e interacción de los personajes se explica con mayor detalle el funcionamiento del flujo completo.

5.2.4 Funcionamiento del algoritmo

El algoritmo recibe su input de entrada, y las listas vacías, luego agrega el nodo inicial NI a la lista abierta de nodos, después de esto, entra en un bucle el cual recorre cada uno los nodos NX pertenecientes a la lista de nodos abiertos y verifica cuál de ellos debe ser el siguiente nodo a evaluar, para determinar cuál es el elegido, compara el costo F del nodo NX con cada uno de sus nodos vecinos NV que sean transitables, y que no estén en la lista cerrada.

Si comprueba que el costo F del nodo NX es menor que el del nodo NV, procede a verificar si el nodo NV está en la lista abierta, en caso de no pertenecer a dicha lista, le calcula su costo F, le asigna como nodo padre al nodo NX y lo agrega a la lista. Por otro lado si dicho nodo NV, ya está incluido, el algoritmo compara el costo G (distancia del inicio al nodo) del nodo NV, con el costo que tendría si se fuese a ese nodo a partir del nodo NX. En caso que el costo G del nodo NV sea inferior, el algoritmo prosigue. En caso contrario cambia coloca como padre del nodo NV al nodo NX y posteriormente recalcula su costo F.

El bucle termina cuando el nodo a evaluar NX es el nodo final o si la lista abierta se queda vacía, lo que significará que no hay solución. Por último el algoritmo reconstruye el camino a partir del nodo final y recorriendo en búsqueda de los padres de cada nodo, para luego agregarlo a la pila que contenga el mejor camino. En la siguiente ilustración, se puede apreciar el funcionamiento. También ayudará a comprender mejor el algoritmo, el pseudocódigo del siguiente apartado.

5.2.5 Pseudocódigo

```
ALGORITMO A*
        NI→ LA
        REPETIR
                NE = NODO_A_EVALUAR (LA)
                SI (NE = NF) O (LA = { })
                        FIN
                CASO CONTRARIO
                        NE→ LC
                        LV ← OBTIENE_NODOS_VECINOS (NE)
                        LA ← EVALUAR_NODOS_ADYACENTES (LV, NE)
                FIN SI
        FIN REPETIR
                        CAMINO = RECONSTUYE_CAMINO (NF)
```

```
NODO_A_EVALUAR:
        LOOP  LA →   NE
                SI F (NE) < FMENOR
                        FMENOR = NE
                FIN SI
        ENDLOOP
        RETORNA FMENOR
```

```
RECONSTUYE_CAMINO
    REPETIR
        NC = PADRE (NF)
        SI  NC <> NI
            NC  →  CAMINO
        CASO CONTRARIO
            FIN
        FIN SI
    FIN REPETIR
```

```
EVALUAR_NODOS_ADYACENTES:
    LOOP  LV →  NV
    SI NV ∉ LC Y TRANSITABLE (NV)
        SI NV ∉ LA
            LA ← NV
            PADRE (NV) = NE
            G (NV) = DISTANCIA (NV, NE)
            H (NV) = DISTANCIA_HEURISTICA (NV, NF)
            F (NV) = G (NV) = H (NV)
        CASO CONTRARIO
            SI G (NV) > G (NE) + DISTANCIA (NV, NE)
                PADRE (NV) = NE
                G (NV) = DISTANCIA (NV, NE)
                H (NV) = DISTANCIA_HEURISTICA (NV, NF)
                F (NV) = G (NV) = H (NV)
            FIN SI
        FIN SI
    FIN SI
```

5.3 Algoritmo HPA*

Como se ha explicado, el algoritmo A* funciona explorando los nodos vecinos en función del costo de los nodos que se encuentran en una lista abierta; sin embargo, conforme se incrementa el número de nodos del área de búsqueda el tiempo de

ejecución del algoritmo se incrementa, lo que hace que el algoritmo A* no sea eficiente para un videojuego RTS en algunas situaciones.

Es por esto que se plantea usar el algoritmo A* jerárquico, el cual permite crear distintas áreas de búsqueda, de esta manera se optimizará el tiempo de ejecución en la búsqueda de caminos.

5.3.1 Creación de áreas búsqueda

El uso de distintas grillas, dependerá de la distancia entre el nodo inicial y el nodo final sobre el que se realizará la búsqueda, de tal manera que, para que las búsquedas en las cuales la distancia supere un mínimo determinado, se utilizará una grilla con nodos de mayor dimensión, es decir en un nivel jerárquico superior, esto permitirá que la cantidad de nodos a explorar sea menor, lo cual finalmente contribuirá a reducir el tiempo de ejecución del algoritmo.

En la siguiente ilustración se puede observar el uso de dos grillas, en niveles de jerarquía distintos. La primera grilla está formada por lados de celdas de 18 por 18, mientras que la grilla de nivel 2 está formada por lados de 3 por 3 celdas.

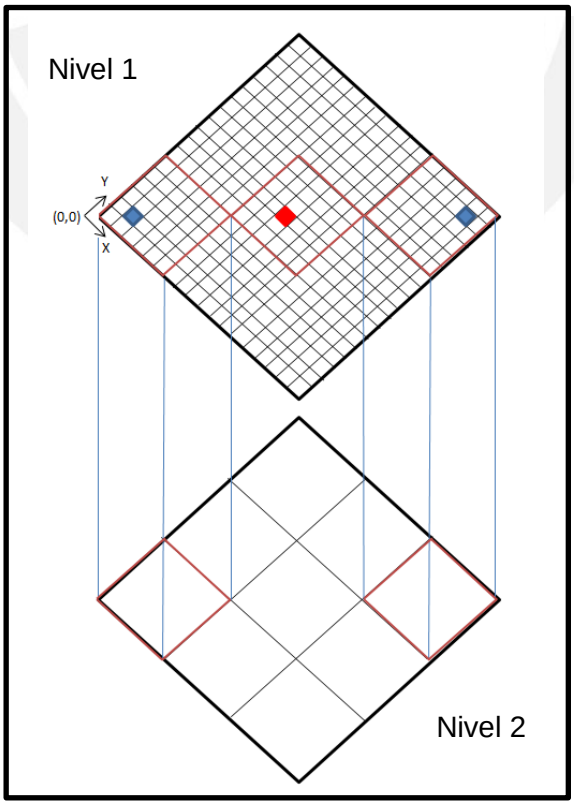

Ilustración 5.1. Jerarquías de áreas de búsqueda (Elaborado por el autor)

Continuando con la revisión de la ilustración, a modo de ejemplo se puede observar la diferencia entre ejecutar el algoritmo A* y el HPA*, si se requiere buscar un camino entre la celda I (1,1) y la celda F (17,17).

Al aplicar el algoritmo A* se procedería a buscar directamente el camino utilizando el área de búsqueda de la grilla superior, considerando I como celda inicial y F como celda final, la cantidad de nodos que distan entre I y F es 16.

Por otro lado si se aplicase el algoritmo HPA*, primero se hallaría la búsqueda de camino A* con la grilla inferior para celdas I (0,0) y la celda F(2,2), una vez hallado dicho camino, se ejecuta nuevamente el algoritmo A* utilizando la grilla superior, es decir, para las celdas I (1,1) y la F (9,9), luego a la pila de celdas que se obtiene, se le agregará las celdas que resulten de aplicar el algoritmo A* nuevamente, para los puntos I (9,9) y F (17,17).

Como se observa el para encontrar la solución, el algoritmo HPA* invocará 3 veces al algoritmo A*; sin embargo, cada una de las búsqueda que realice este algoritmo será para distancias más cortas que si se aplicase directamente entre las celdas (1,1) y las celdas (17,17).

Como se demostrará en el apartado 0 el algoritmo HPA* puede llegar a ser bastante más eficiente en tiempo de ejecución que el algoritmo A* para distancias largas.

5.3.2 Consideraciones del algoritmo

El algoritmo HPA* recibe como entrada de datos (input) el nodo inicial, y el entorno de búsqueda en la estructura Mapa, dicho entorno debe ser de un nivel jerárquico superior, además, utiliza las mismas estructuras que el algoritmo A*, así como la misma función heurística tal y como se encuentra indicado en el apartado 5.2.1.

5.3.3 Adaptación del algoritmo

Cuando el bucle principal del videojuego detecte la orden para ejecutar el desplazamiento un agente de un punto a otro, primero deberá verificar la distancia en el sistema de coordenadas absoluto entre el punto inicial y final, luego se debe

verificar si dicha distancia excede a la distancia máxima determinada para la aplicación del algoritmo A*, en ese caso, se procede a ejecutar el algoritmo HPA*.

5.3.4 Pseudocódigo

Algoritmo HPA*

```
CAMINO = BUSQUEDA_A* (mapaN2, NI, NF)
REPETIR
        N = PULL (CAMINO)
        SI NC = NF
                FIN
        FIN SI
        CN = CENTRO_NODO (N)
        BUSQUEDA_A* (mapaN1, N, CN)
FIN REPETIR
```

La función CENTRO_NODO devolverá el nodo central en la grilla de nivel 1, para la celda de la grilla de nivel 2 dada.

6. Experimentación numérica

A continuación, en el presente capítulo, se describe la experimentación numérica realizada, la cual permite demostrar que el algoritmo HPA* tiene un mejor rendimiento que el A*; y que dicha mejora en el rendimiento es aún más evidente bajo ciertas condiciones. Para evaluar el rendimiento, se consideran los factores de tiempo de ejecución y espacio de memoria requerido por el algoritmo. Finalmente, se utilizó el software IBM SPSS Statics para la realización del experimento.

6.1 Prueba de Hipótesis 1: Rendimiento por tiempo de ejecución

6.1.1 Definición del problema de la experimentación

En este primero experimento, lo que se busca demostrar a través de la experimentación numérica, es la mejora en el tiempo de ejecución para la búsqueda de un camino de parte del algoritmo HPA* sobre el A*.

6.1.2 Variable de la experimentación

La variable utilizada para la experimentación es la siguiente:

- Tiempo de ejecución: El tiempo que demora el algoritmo en encontrar el camino solución entre dos nodos de un grafo.

6.1.3 Hipótesis Principal

La prueba pretende demostrar que la media de los tiempos de ejecución para resolver el mismo problema varía de acuerdo al algoritmo utilizado

Hipótesis 1

Sean:

X1: *La media del tiempo de ejecución del algoritmo HPA**

X2: *La media del tiempo de ejecución del algoritmo A**

- ***Ho: X1 - X2 = 0****: La media de los tiempos de ejecución en la búsqueda de caminos utilizando el algoritmo HPA* es igual a la media del tiempo de ejecución usando el algoritmo A*.*
- ***H1: X1 ≠ X2:*** *La media de los tiempos de ejecución en la búsqueda de caminos utilizando el algoritmo HPA* es diferente a la media del tiempo de ejecución usando el algoritmo A*.*

6.1.4 Estadístico a utilizar

Como el tamaño de la muestra es menor a 30 y se conoce la desviación estándar, se recomienda utilizar el estadístico T-Student , de tal manera que se verifique la diferencia de medias. Para este experimento, se eligió un con un nivel de significación de 5%, es decir hay una posibilidad de 0.05 de rechazar la hipótesis, siendo esta cierta.

T-Student:

$$t = \frac{X1 - X2}{\sqrt{\frac{S1^2}{n1} - \frac{S2^2}{n2}}}$$

Donde:

S1: varianza muestral del tiempo de ejecución usando algoritmo HPA*
S2: varianza muestral del tiempo de ejecución usando algoritmo A*

n1: tamaño de la muestra 1
n2: tamaño de la muestra 2

6.1.5 Consideraciones para la experimentación

Se tomaran dos muestras aleatorias con los siguientes tamaños

- N1 tamaño de la muestra utilizando el algoritmo HPA* : 12
- N2 tamaño de la muestra utilizando el algoritmo A* : 12

- Grados de libertad : n1 + n2 – 2 = 12 + 12 – 2 = 22

- Para el experimento se utilizó distintos juegos de datos, los cuales se pueden observar en el anexo 5.

A continuación, en la Tabla 6.1, se observan los tiempos en milisegundos obtenidos en las corridas de ambos algoritmos.

Ejecución	Algoritmo A*	Algoritmo HPA*
1	1044.00	52.00
2	151.00	31.00
3	3763.00	33.00
4	106.00	21.00
5	1636.00	65.00
6	2314.00	86.00
7	1020.00	49.00
8	143.00	33.00
9	3000.00	28.00
10	106.00	22.00
11	1512.00	60.00
12	2300.00	83.00

Tabla 6.1. Tiempos de ejecución de los algoritmos

6.1.6 Resultados de la experimentación

Supuestos para la aplicación de T-Student:

- Supuesto de Normalidad
- Supuesto de igualdad de Varianzas

Para verificar el supuesto de normalidad se utilizó la prueba de Kolmogorov-Smirnov, la cual permitió verificar la hipótesis de que los datos provienen de una distribución normal

Hipótesis 2

- Ho: *Los datos provienen de una distribución normal.*
- H1: *Los datos no provienen de una distribución normal.*

One-Sample Kolmogorov-Smirnov Test[a]		
		Tiempo
N		12
Normal Parameters[b,c]	Mean	1424.5833
	Std. Deviation	1228.82837
Most Extreme Differences	Absolute	.183
	Positive	.183
	Negative	-.142
Kolmogorov-Smirnov Z		.635
Asymp. Sig. (2-tailed)		.815

Tabla 6.3. Prueba K-S para tiempos de ejecución del A*

One-Sample Kolmogorov-Smirnov Test[a]		
		Tiempo
N		12
Normal Parameters[b,c]	Mean	46.9167
	Std. Deviation	22.66137
Most Extreme Differences	Absolute	.230
	Positive	.230
	Negative	-.126
Kolmogorov-Smirnov Z		.798
Asymp. Sig. (2-tailed)		.547

Tabla 6.2. Prueba K-S para tiempos de ejecución del HPA*

De los resultados obtenidos en la Tabla 6.3 y la Tabla 6.2, se observa que el P-valor (Asymp. Sig (2-tailed)), tiene el valor de 0.815 y 0.547 respectivamente; y como ambos son mayores que 0.05, se concluye que no hay evidencia para rechazar la hipótesis nula de la hipótesis 2.

Para verificar el supuesto de igualdad de varianzas, se utilizó la prueba de Levene., y se planteó la siguiente hipótesis:

Hipótesis 3

- Ho: *No hay diferencia significativa entre las varianzas*
- H1: *Hay diferencias significativas entra las varianzas.*

Independent Samples Test			
		Levene's Test for Equality of Variances	
		F	Sig.
Tiempo	Equal variances assumed	26.818	.000
	Equal variances not assumed		

Tabla 6.4. Prueba de Levene para los tiempos de ejecucuión del A*

Como el P-valor de la Tabla 6.4 es menor que la significancia (0.05), entonces se rechaza la hipótesis nula. De la tabla se observa que el P-valor es menor que 0.05, en consecuencia se rechaza la hipótesis nula, y no se asumen varianzas iguales.

Finalmente de la prueba T-Student, en la Tabla 6.5 se consideran los resultados la segunda fila, donde no se asumen las varianzas iguales, cuyo P-valor es 0.039, el cual es menor a 0.05. Entonces se concluye que hay suficiente evidencia para rechazar la hipótesis nula de la hipótesis 1, por lo que nos quedamos con la hipótesis alterna, la cual dice que la diferencia de las medias es significativa.

Independent Samples Test								
		t-test for Equality of Means						
		t	df	Sig. (2-tailed)	Mean Difference	Std. Error Difference	95% Confidence Interval of the Difference	
							Lower	Upper
Tiempo	Equal variances assumed	3.883	22	.001	1377.66667	354.79251	641.87203	2113.46130
	Equal variances not assumed	3.883	11.007	.003	1377.66667	354.79251	596.83837	2158.49496

Tabla 6.5. Prueba T-Student e para los tiempos de ejecucuión del A*

6.2 Prueba de Hipótesis 2: Rendimiento por uso de memoria

A continuación se presenta el segundo experimento, en el cual se buscó demostrar como el algoritmo HPA* propuesto, permitió reducir el uso de la memoria.

6.2.1 Definición del problema de la experimentación

En el segundo experimento, lo que se busca demostrar a través de la experimentación numérica, es la mejora en el uso de memoria para la búsqueda de un camino de parte del algoritmo HPA* sobre el A*.

6.2.2 Variable de la experimentación

La variable utilizada para la experimentación es la siguiente:

Uso de memoria: Cantidad de memoria usada por el algoritmo para encontrar el camino solución entre dos nodos de un grafo.

6.2.3 Hipótesis Principal

La prueba pretende demostrar que la media del espacio de memoria usado para resolver el mismo problema varía de acuerdo al algoritmo utilizado

Hipótesis 1:

Sean:

X1: *La media del espacio de memoria usado por el algoritmo HPA**

X2: *La media del espacio de memoria usado por el algoritmo A**

- ***Ho: X1 - X2 = 0****: La media del espacio de memoria usado en la búsqueda de caminos utilizando el algoritmo HPA* es igual a la media del tiempo de ejecución usando el algoritmo A*.*
- ***H1: X1 ≠ X2:*** *La media del espacio de memoria usado en la búsqueda de caminos utilizando el algoritmo HPA* es diferente a la media del tiempo de ejecución usando el algoritmo A*.*

6.2.4 Estadístico a utilizar

Como el tamaño de la muestra es menor a 30 y se conoce la desviación estándar, se recomienda utilizar el estadístico T-Student, de tal manera que se verifique la diferencia de medias. Para este experimento, se eligió un con un nivel de significación de 5%, es decir hay una posibilidad de 0.05 de rechazar la hipótesis, siendo esta cierta.

T-Student:

$$t = \frac{X1 - X2}{\sqrt{\frac{S1^2}{n1} - \frac{S2^2}{n2}}}$$

Dónde:

S1: varianza muestral del tiempo de ejecución usando algoritmo HPA*

S2: varianza muestral del tiempo de ejecución usando algoritmo A*

n1: tamaño de la muestra 1

n2: tamaño de la muestra 2

6.2.5 Consideraciones para la experimentación

- Se tomaran dos muestras aleatorias con los siguientes tamaños
    - N1 tamaño de la muestra utilizando el algoritmo HPA* : 12
    - N2 tamaño de la muestra utilizando el algoritmo A* : 12

Los juegos de datos tomados como prueba, se pueden ver en el (anexo 4.

- Grados de libertad :

  n1 + n2 – 2 = 12 + 12 – 2 = 22

- Para el experimento se utilizó distintos juegos de datos, los cuales se pueden observar en el anexo 4.

- Valores de la muestra (en milisegundos):

Ejecución	Algoritmo A*(Mb)	Algoritmo HPA*(Mb)
1	37	3
2	6	3
3	3	3
4	121	4
5	3	3
6	0.24	0.20
7	35	5
8	8	4

9	142	4
10	118	4
11	3	3
12	0.23	0.22

Se utilizó el software IBM SPSS Statics para la realización de la prueba T-Student, los resultados obtenidos fueron los que se muestran en las siguientes ilustraciones.

6.2.6 Resultados de la experimentación

Supuestos para la aplicación de T-Student:

- Supuesto de Normalidad.
- Supuesto de igualdad de Varianzas.

Para verificar el supuesto de normalidad se utilizó la prueba de Kolmogorov-Smirnov, la cual permitirá verificar la hipótesis de que los datos provienen de una distribución normal.

Hipótesis 2

- Ho: *Los datos provienen de una distribución normal.*
- H1: *Los datos no provienen de una distribución normal.*

One-Sample Kolmogorov-Smirnov Test[a]

		Memoria
N		12
Normal Parameters[b,c]	Mean	39.7058
	Std. Deviation	54.38204
Most Extreme Differences	Absolute	.303
	Positive	.303
	Negative	-.234
Kolmogorov-Smirnov Z		1.051
Asymp. Sig. (2-tailed)		.219

Tabla 6.7. Prueba K-S para el uso de memoria del A*

One-Sample Kolmogorov-Smirnov Test[a]

		Memoria
N		12
Normal Parameters[b,c]	Mean	3.0350
	Std. Deviation	1.46330
Most Extreme Differences	Absolute	.324
	Positive	.171
	Negative	-.324
Kolmogorov-Smirnov Z		1.122
Asymp. Sig. (2-tailed)		.161

Tabla 6.6. Prueba K-S para el uso de memoria del HPA*

De los resultados obtenidos en la prueba Kolmogorov-Smirnov, se observa que el P-valor (Asymp. Sig (2-tailed) en la tabla), tiene el valor de 0.815 y 0.547 en las pruebas para las muestras del algoritmos A* y HPA* respectivamente, ambos son mayores que 0.05, por lo tanto no hay evidencia para rechazar la hipótesis nula de la hipótesis 2.

Para verificar el supuesto de igualdad de varianzas, se utilizó la prueba de Levene., y se planteó la siguiente hipótesis:

Hipótesis 3

- Ho: *No hay diferencia significativa entre las varianzas*
- H1: *Hay diferencias significativas entra las varianzas.*

Independent Samples Test			
		Levene's Test for Equality of Variances	
		F	Sig.
Memoria	Equal variances assumed	24.828	.000
	Equal variances not assumed		

Tabla 6.8. Prueba de Levene para el uso de memoria

Independent Samples Test								
		t-test for Equality of Means						
		t	df	Sig. (2-tailed)	Mean Difference	Std. Error Difference	95% Confidence Interval of the Difference	
							Lower	Upper
Memoria	Equal variances assumed	2.335	22	.029	36.67083	15.70443	4.10185	69.23982
	Equal variances not assumed	2.335	11.016	.039	36.67083	15.70443	2.11172	71.22994

Tabla 6.9. Prueba de T-Student para el uso de memoria

Como el P-valor de la Tabla 6.8 es menor que la significancia (0.05), entonces se rechaza la hipótesis nula, en consecuencia no se asumen varianzas iguales.
Finalmente, se considera la segunda fila de la Tabla 6.9, donde no se asumen las varianzas iguales, cuyo P-valor 0.039 es menor a 0.05. Entonces se concluye que hay suficiente evidencia para rechazar la hipótesis nula de la hipótesis 1, por lo que nos quedamos con la hipótesis alterna, la cual dice que la diferencia de las medias es significativa.

7. Construcción del módulo de Control de movimiento e interacción de los personajes

El presente capítulo se divide en dos partes, la primera es la implementación de los algoritmos y mecanismos para el control de movimiento de los personajes, la segunda se enfoca en la interacción de estos. Para esto, se describen las principales consideraciones que se tuvieron en cuenta al implementar dichas funcionalidades, buscando que se cumpla con lo descrito en la descripción sistematizada del capítulo 0.

7.1 Entorno de movimiento e interacción de los agentes

A diferencia de lo mencionado en el apartado 5.1.1, en donde se explica la utilización de un entorno de búsqueda estático para los algoritmos de búsqueda de caminos, en el caso de los algoritmos y mecanismos de movimiento e interacción de los agentes, se debe considerar un entorno dinámico, el cual varía constantemente conforme se desplacen los agentes.

7.1.1 Entornos dinámicos

El entorno para el desplazamiento es considerado dinámico, puesto que los agentes no mantienen una posición única, sino por el contrario, su posición es constantemente actualizada por el componente de simulación. Si bien, son varias las posibles causas que motivan a un agente a desplazarse, el mecanismo de movimiento es el mismo.

7.1.2 Sistema de coordenadas físicas

El sistema de coordenadas físicas está delimitado por la distribución de la imagen del mapa en el monitor, es decir, cada pixel representará una coordenada física, además el punto (0,0) u origen de coordenadas está ubicado en el extremo superior izquierdo del mapa. El eje de las ordenadas se incrementa hacia abajo de la pantalla y el eje de las abscisas se incrementa a la derecha de esta.

A diferencia de los algoritmos de búsqueda, los de movimiento no pueden basarse en el sistema de coordenadas lógicas ya que el desplazamiento de los agentes debe ser realizado a nivel físico, es decir píxel por píxel, para producir así un nivel de realismo en el videojuego.

7.1.3 Personajes como puntos en su entorno

Un personaje usualmente es representado como un modelo en tres dimensiones que ocupa algún espacio en su entorno. Los mecanismos para la detección de colisiones o evasión de obstáculos dinámicos suelen requerir de las dimensiones del personaje para la obtención de sus resultados; sin embargo, para los algoritmos de movimiento, estos pueden ser considerados simplemente un punto en el espacio. En la Ilustración 7.1 se puede observar un ejemplo de un personaje representado como un punto en un sistema de coordenadas físicas (MILLINGTON, 2009).

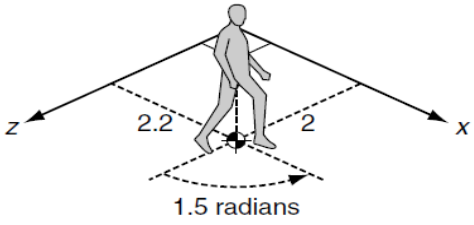

Ilustración 7.1 Personaje en un sistema de coordenadas (MILLINGTON, 2009)

7.2 Movimiento de los agentes

Como se ha mencionado en el apartado 3.2, el proceso de navegación de los personajes consta de varias etapas, por lo tanto, una vez que el componente de búsqueda de caminos ha determinado el camino que deben seguir los agentes para llegar a su destino, se procede a ejecutar el movimiento o en sí de los agentes.

7.2.1 Algoritmo de movimiento

Cada agente debe tener en todo momento una posición actual, tanto a nivel físico como lógico, y ocasionalmente puede tener asignado un camino a seguir. Además, se debe considerar que los agentes tienen características y atributos propios tales como la velocidad, rango de visión, entre otros, que influirán en la ejecución de los algoritmos.

Es el algoritmo que se encarga de que en cada iteración del bucle principal del juego, se actualicen las posiciones físicas y lógicas de cada uno de los agentes, lo realiza en función del nodo objetivo que tiene como destino.

7.2.2 Comportamientos de dirección

Los mecanismos de comportamiento de dirección son parte fundamental de los algoritmos de movimiento y se encargan de actualizar el atributo dirección que tienen los agentes, de tal manera que sigan el camino hallado con el algoritmo de búsqueda. Esta implementación cuenta con ocho direcciones en función de los puntos cardinales: Norte, noreste, este, sureste, sur, suroeste, oeste, noroeste, a cada dirección se le asignó un valor numérico del uno al ocho respectivamente como un atributo de la clase Personaje. Además, cada objeto instanciado en el sistema de objetos de la clase personaje tendrá como atributo la dirección actual del personaje.

7.2.3 Seguimiento del camino

Una vez que el componente de búsqueda halla el camino que debe de seguir un personaje, se procede con la ejecución del algoritmo de seguimiento del camino. Este consiste en que el componente de simulación calcule la futura posición del personaje en función de su posición y dirección actual así como de la celda destino

del personaje; luego se actualizan la posición del personaje y dirección del agente con la nueva posición hallada y le informa al componente gráfico del motor que dibuje al personaje en su nueva posición. Este proceso se repite de manera continua hasta que la posición lógica del personaje sea igual a la de su la posición destino, lo cual significa que terminó su recorrido.

7.3 Interacción de los agentes

La cuarta etapa del mecanismo de navegación autónomo descrito en el apartado 3.2.4, corresponde a la interacción de los personajes, al igual que el movimiento es parte del componente de simulación del videojuego. La interacción es un proceso que sucede permanentemente en la ejecución del juego, ésta se encarga de verificar cuál es la acción activa de cada uno de los personajes de ambos bandos y de procesarlas.

7.3.1 Mecanismos de interacción

Cada agente tiene un arreglo de acciones asignadas: Atacar, curar, reclutar, patrullar, detener, mover y mantener posición. Además cuentan con un arreglo auxiliar para la acción de patrullar. El rango de visión es un atributo fundamental en la interacción, pues cada acción se ejecutará justamente cuando el objetivo esté a una distancia menor al rango de visión del personaje.

Este algoritmo se encarga de que en cada iteración del bucle principal del juego, se verifique si cada personaje tiene asignado un objetivo en el atributo que contiene la acción del personaje, luego revisa su arreglo de acciones y la procesa.

7.3.2 Reacción del personaje

Durante la iteración del bucle principal del videojuego, se llama al método encargado de procesar el movimiento, el cual verifica el bando al que pertenece el personaje que se quiere procesar, luego se verifica el tipo del personaje y posteriormente se consulta su arreglo de acciones activas; una vez determinada la acción a ejecutar, se utilizan los métodos distancia y cerca de la clase de funciones geométricas, para verificar si existe o no un personaje a quien aplicarle la acción.

8. Pruebas

Siguiendo la metodología Extremme programming utilizada en el presente proyecto, toda funcionalidad implementada, previamente se debió definir un caso de prueba que permita luego verificar el correcto funcionamiento de lo programado. A continuación se describen los resultados de las pruebas unitarias que se realizaron.

8.1 Pruebas del módulo de búsqueda de caminos

Se describe el procedimiento seguido para la realización de las pruebas del módulo de búsqueda de caminos y posteriormente se indican los resultados obtenidos.

8.1.1 Procedimientos para las pruebas

Se desea verificar que los algoritmos implementados sean los adecuados, para lo cual, se verificará que cumplan con los criterios de optimalidad, tiempo de respuesta y ocupación de memoria que se establecieron en el apartado 5.1.4, por lo tanto a continuación se presentan los resultados de los distintos casos de prueba.

Los casos de pruebas utilizados, se encuentran es el anexo 5

8.1.2 Resultados de las pruebas por Historia de usuario

Se verificaron las historias de usuario 20 y 21:

- Para verificar que el algoritmo HPA* tiene menor tiempo de respuesta que el algoritmo A*, se realizó la experimentación numérica mostrada en el apartado 6.1.

- Para verificar que el algoritmo HPA* utiliza menos memoria que el algoritmo A*, se realizó la experimentación numérica mostrada en el apartado 6.2.

En las siguientes tablas se prueba que conforme la distancia entre el nodo inicial y final es mayor, las diferencias de rendimientos de ambos algoritmos es bastante mayor, por el contrario, para distancias pequeñas, el rendimiento de ambos algoritmos es casi el mismo.

A continuación se muestran las tablas Tabla 8.1 y Tabla 8.2 con los resultados de la experimentación para el mapa 1.

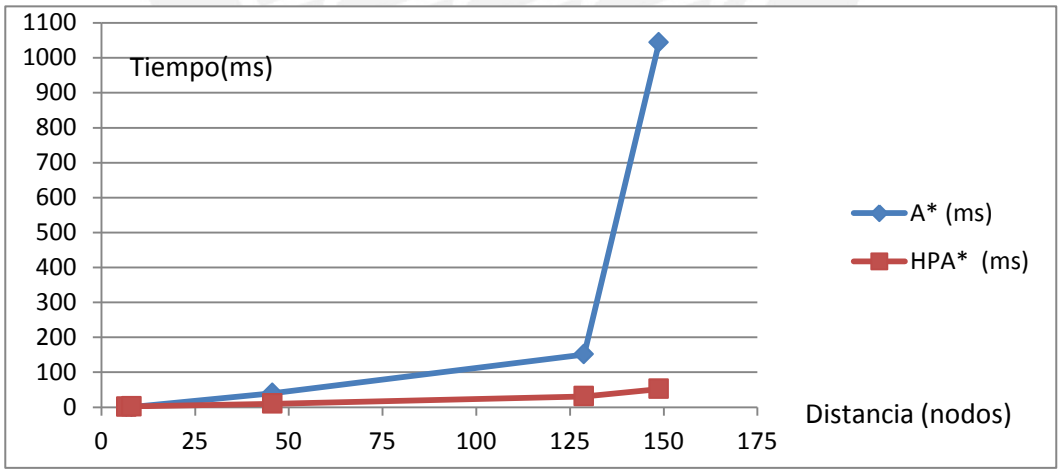

Tabla 8.1. Tabla comparativa de rendimiento del tiempo de respuesta

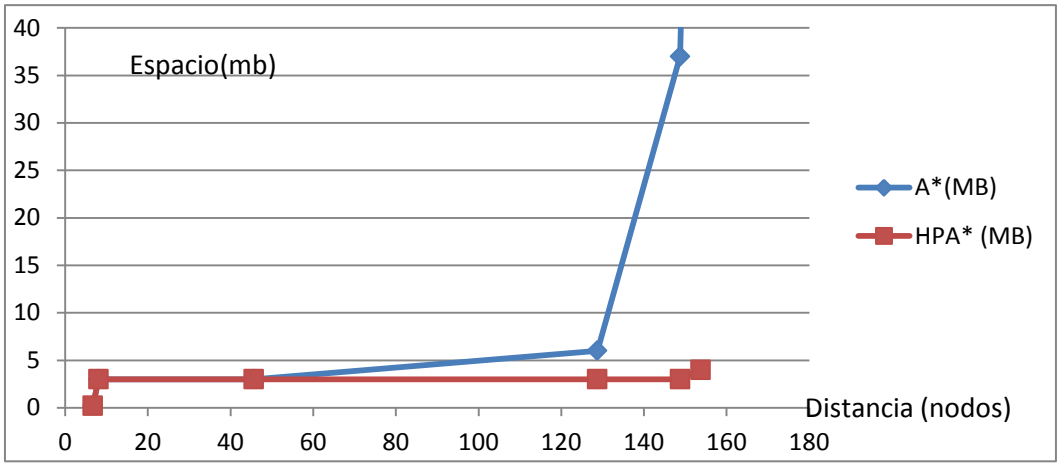

Tabla 8.2. Tabla comparativa de rendimiento del uso de memoria

En ambas tablas**,** se observa que el crecimiento de tiempo y el uso de memoria, aumenta exponencialmente, conforme los nodos se encuentran más alejados, además, para distancia cortas menores o iguales a los 10 nodos, las diferencias de los tiempos de ejecución es insignificante.

Por último, en distancias cortas, debido a que el algoritmo HPA* no es un algoritmo exacto, es decir trabaja con aproximaciones, tiene un comportamiento extraño.

8.2 Pruebas del módulo de control de movimiento.

A continuación, se describe el procedimiento seguido para la realización de las pruebas del módulo de control de movimiento y posteriormente se indican los resultados obtenidos.

8.2.1 Procedimientos para las pruebas

Para realizar las pruebas respectivas a las historias de usuarios de módulo de control de movimiento de los personajes, se utilizó el prototipo del videojuego implementado, y se probó cada una de las historias.

8.2.2 Resultados de las pruebas

Utilizando el prototipo del videojuego, se probaron la historias de usuario de los módulos de control de movimiento y el gestor de datos. Los resultados de las pruebas se encuentran en el Anexo 6.

9. Observaciones, conclusiones y recomendaciones

En este capítulo se describen las principales conclusiones y observaciones que se han podido concluir a lo largo del desarrollo del proyecto. Finalmente se realiza una discusión de los temas abordados y se plantean sugerencias y recomendaciones para futuros trabajos que puedan complementar lo desarrollado en este proyecto.

9.1 Conclusiones

- Se estableció la importancia de describir sistemáticamente las características y funcionalidades del componente desarrollado, pues gracias a esta, se pudo validar que cada uno de los resultados esperados cumpla con las principales características de un videojuego RTS.

- Se comprobó el correcto funcionamiento de arquitectura diseñada, basada el modelo de vistas 4+1 de Kruchten recomendado por la IEEE, junto a un modelo de arquitectura especializada en videojuegos del tipo RTS propuesto por Michael Doherty.

- Se demostró mediante un experimento de comparación de medias de dos muestras independientes, que la media muestral del tiempo de ejecución del algoritmo HPA* es menor que la del A*, lo cual permitió comprobar la veracidad de la hipótesis planteada. Además, se pudo verificar que para distancias mayores a 100 nodos, el algoritmo HPA* puede llegar a ser hasta 30 veces más rápido que el A*; sin embargo, en distancias pequeñas menores a los 10 nodos, la diferencia de los tiempos de ejecución es despreciable.

- Se demostró mediante un experimento de comparación de medias, de dos muestras independientes, que la media muestral del uso de espacio de memoria por parte del algoritmo HPA* es menor que la del A*, lo cual permitió comprobar la veracidad de la hipótesis planteada. Además, la diferencia de su consumo se incrementa exponencialmente para distancias mayores a los 100 nodos. Finalmente, se pudo verificar que para distancias cortas, el algoritmo HPA* utiliza una cantidad de memoria constante, cercana a los 3MB.

- Se comprobó que la utilización de mapas basados en grillas junto con la técnica de puntos de paso en un área de búsqueda jerárquica de dos niveles, mejora los tiempos de ejecución y uso de memoria, ya que se reduce la cantidad de nodos que se deben explorar para hallar la solución.

- Se demostró con éxito el acoplamiento del componente de búsqueda de caminos y control de movimiento con los componentes encargados de implementar la lógica y gestionar los gráficos del videojuego, los cuales fueron desarrollados en otros proyectos, Por último, se utilizó dicha adaptación en la construcción del videojuego ”1814, La Rebelión del Cuzco”.

9.2 Observaciones

- Se observó que el criterio de optimalidad del camino hallado en el caso del algoritmo HPA*, puede verse afectado en ciertos escenario específicos de búsqueda, ya que trabaja sobre aproximaciones, por lo tanto, se utilizó el algoritmo A* para distancias cortas menores a los diez nodos.

- El componente desarrollado contiene algunos de los mecanismos de comportamientos de dirección propuestos por Craig Reynolds, en particular el

seguimiento del camino y la detección de colisiones; sin embargo, es importante resaltar que existen varios mecanismos más; tales como la evasión de colisiones dinámicas o un sistema de Flocking; Los cuales podrían implementarse a futuro, para mejorar las características del componente desarrollado.

- Se creó y verificó el correcto funcionamiento de los archivos para la configuración de las características y atributos de los personajes, de tal manera que de acuerdo a los valores establecidos en dichos archivos, varíe el comportamiento de los algoritmos y mecanismos de movimiento e interacción.

- Durante la adaptación, se observó que la arquitectura empleada, permitió el acoplamiento de los componentes, además, se logró que el tiempo de respuestas de los algoritmos en el videojuego sean eficientes y por lo tanto no causen retrasos.

9.3 Recomendaciones y trabajos futuros

Como se ha mencionado, el componente desarrollado en este proyecto ha sido adaptado a un videojuego que se viene desarrollando en la Pontificia Universidad Católica; sin embargo, el propósito de este trabajo es que pueda trascender a dicho videojuego, y sea visto como parte de la elaboración de un motor, que esté al alcance de quienes desean iniciarse en la industria de desarrollo de videojuegos de estas características.

A continuación, se presenta algunas recomendaciones para trabajos futuros que se relacionen al tema.

- Para la implementación de los algoritmos de búsqueda, se tuvo que elaborar dos archivos de texto para la configuración de un mismo mapa en dos niveles jerárquicos distintos, ver anexo 4; realizar dichos mapas significó un consumo de tiempo considerable; y si bien, se creó algunas fórmulas de conversión isométrica como herramienta de apoyo, la elaboración siguió siendo un procedimiento manual bastante tedioso. Por lo tanto, se recomendaría que en un futuro proyecto de fin de carrera se implemente un componente que pueda generar dichos archivos de configuración de manera automática.

- Respecto al módulo de movimiento e interacción de los agentes, se desarrollaron algunos de los comportamientos de dirección más básicos utilizados en los videojuegos RTS; sin embargo, existen muchos mecanismos del componente de IA que no pertenecen alcance de este proyecto, loso que podrían ser implementados en futuros trabajos, como son por ejemplo, los mecanismos del flocking.

- Como se ha mencionado, el objetivo del proyecto era implementar un componente de navegación autónomo, haciendo énfasis en solucionar el problema de la búsqueda del camino más corto en entornos estáticos e implementar un mecanismo de desplazamiento e interacción de agentes; sin embargo, dichos componentes son solo parte de la inteligencia artificial que puede llegar a tener un videojuego complejo. Es por esto que, se propone como un posible trabajo futuro, mejorar la inteligencia de la máquina mediante herramientas como: árboles de decisiones, probabilidades o lógica difusa,

10. Bibliografía

- **[1]SMED Jouni, Harri Hakonen, Algorithms and Networking for Computer Games**
  1995 *Prentice Hall-Inc*
  Segunda Edición ISBN – 0-13-103805-2
- **[2]RUSELL Stuart, Peter Norving, Artificial Intelligence A modern aproach**
  Wiley 2006 *University of Turku, Finland*
  Segunda Edición ISBN – 13: 978-0-047-01812-5
- **[3]RABIN Steve, AI Game Programming WISDOM**
  Charles River Media
  ISBN: 1-58450-077-8
- **[4]BUCKLAND Matt, Programming Game AI by example**
  2005 Wordware Publishing, Inc.
  ISBN: 1-55622-078-2
- **[5]REUTERS,** FACTBOX – A look at the $66 billion video-games industry
  Última visita: 28/10/2013
  Url: <http//in.reuters.com/article/2013/06/10/>
- **[6]ESA,** Entertainment software association
  Última visita: 05/10/2013
  <http://www.theesa.com/games-improving-what-matters/schools./schools.asp?degree=PhD >
- **[7]BLIZZARD Entertainment, Warcraft Reign of chaos**
  Última visita: 12/04/2013
  Url: <http/us.blizzard.com/en-us/games/war3/>
- **[8]MICROSOFT Games, Age of empires: The age of kings**
  Última visita: 12/04/2013
  Url: <http://www.microsoft.com/games/age2/>
- **[9]PEDERSEN** E,Roger Game Design Foundations
  Wordware Publishing, Inc.
  ISBN: 1-55622-973-9
- **[10] Stanford, Amits game programing**
  Ultima visita: 12/06/2012
  Url: http://theory.stanford.edu/~amitp/GameProgramming/

- **[11] Avatar,PUCP**
  PUCP. 2012 “Grupo Avatar Pucp”. Lima, Perú.
  Consulta: 10 de Abril del 2013
  Url: <http://avatar.inf.pucp.edu.pe/index.php?seccion=quienes&id=0
- **[12]COIDEV Congreso Internacional de desarrolladores de Videojuegos en Lima**
  ultima visita: 03/08/2013
  Url: <http://www.coidev.net/>
- **[13] ORACLE:** Java sun technetwork
  Última visita: 10 de Septiembre del 2013
  Url: <http://www.oracle.com/technetwork/topics/>
- **[14]SNOOK, Simplified 3D movement and pathfinding using navigation meshes** Charles River Media.
- **[15] LOKI, Software Inc and Jhon R. Hall , Programming Linux Games**
  Linux Journal Press, San Francisco, CA, USA, 2001.
  Segunda Edición ISBN 1-886411-48-4
- **[16] GREGORY Jason, Game engine architecture**
  Wellesley Massachusetts: A K Peters Ltd, 2009.
  ISBN 978-1-4398-6526-2
- **[17] DOHERTY, Michael**
  A software architecture for games. University of the Pacific
  Department of Computer Science
  (RAPJ), vol 1. No 1. 2003
- **[18] IEEE,** Recommended Practice for Architectural Description for Software-Intensive Systems
  Última visita: 30 de Octubre del 2013
  Url: <http://standards.ieee.org/findstds/standard/1471-2000.html>
- **[19] KRUCHTEN Philippe**, **Architectural Blueprints- The ¨4+1 view¨**
  Rational Software Corp.
  Noviembre 1995, pag 44-52
- **[20] MILLINGTON Ian, Funge Jhon , Artificial intelligence for games**
  2009 Morgan Kauffman Publishers, Burlington MA, 01083 USA
  Segunda Edición ISBN 978-0-12-374731-0
- **[21] ADAMS, Ernest , Break into the Game Industry**
  McGraw-Hill/ Osborne 2100 Powell Street, Emeryville California
  Segunda Edición ISBN 0-07-222660-9

- **[22] ROUSE, Richard , Game Design: Theory and Practice**
  2001, Wordware Publishing, Inc. 2320 Los Rios Boulevard, Texas
  Primera Edición ISBN 1-55622-735-3

- **[23] ZIMMERMAN Eric, Salen Katie , Rules of Play: Game Design Fundamentals**
  2004 Massachusetts Institute of Technology
  Primera Edición ISBN 0-262-24045-9

- **[24] REESE Bjorn, Stout Brian , Finding a Pathfinder**
  Institute for Production Technology, University of Southern Denmark
  Proceedings of the AAAI 99 Spring Symposium on Artificial Intelligence and Computer Games

- **[25] Extreme Programming: A gentle introduction**
  Última visita: 05 de Octubre del 2013
  Url: <http://www.extremeprogramming.org/ >

- **[26] PMI, PROJECT MANAGEMENT INSTITUTE**
  2008 *Guía de los fundamentos para la dirección de proyectos.*
  Pennsylvania: Project Management Institute.
  4ta. Edición. ISBN: 978-1-933890-72-2

- **[27] UML, Unified Modeling Language**
  Última visita: 15 de Octubre del 2013
  Url: <http://www-01.ibm.com/software/rational/uml// >

- **[28] VISIO 2010, Microsoft Visio 2010**
  Última visita: 21 de Octubre del 2013
  Url:<http://www.microsoft.com/latam/gobierno/productos/visio2010.aspx>

- **[29] DE LOURA, Mark, Best of game programming gems**
  Boston 2008 Cengage Learning
  Primera Edición ISBN – 13: 978-1584505716

- **[30] HART, Peter y Nilson, Nils ,**
  A formal basis for the Heuristic Determination of Minimum Cost Paths
  IEEE transactions of systems science and cybernetics,
  vol ssc 4, 1998 pp: 100 – 107

- **[31] TIGRIS, Open Source Software Engineering Tools**
  Última visita: 04 de Noviembre del 2013
  Url:<http://tortoisesvn.tigris.org/>
- **[32] IBM SPSS Statics 20**
  Última visita: 04 de Noviembre del 2013
  Url:<www-01.ibm.com/software/pe/analytics/spss/products/statistics/>
- **[33] MICROSOFT EXCEL, Office 2010**
  Última visita: 09 de Noviembre del 2013
  Url:<http://office.microsoft.com/es-mx/excel-help/que-es-excel-HA010265948.aspx>
- **[34] XSTREAM**
  Última visita: 09 de Noviembre del 2013
  Url:< http://xstream.codehaus.org/ >