PONTIFICIA UNIVERSIDAD CATÓLICA DE VALPARAÍSO

FACULTAD DE INGENIERÍA

ESCUELA DE INGENIERÍA INFORMÁTICA

Análisis de Algoritmos Pathfinding

Danilo Alejandro Cuevas Guzmán

INFORME FINAL DEL PROYECTO
PARA OPTAR AL TÍTULO PROFESIONAL DE
INGENIERO CIVIL EN INFORMÁTICA

Diciembre 2013

PONTIFICIA UNIVERSIDAD CATÓLICA DE VALPARAÍSO
FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERÍA INFORMÁTICA

Análisis de Algoritmos Pathfinding

Danilo Alejandro Cuevas Guzmán

Profesor Guía**: Silvana Roncagliolo de la Horra**
Profesor Co-referente: **Ricardo Soto de Giorgis**

Diciembre 2013

Dedicado mi familia, amigos, compañeros de carrera y profesores, quienes me acompañaron y apoyaron en mi paso por la universidad. Mención especial para mis mascotas y a mis tres amigos de la PUCV.

Resumen

Pathfinding es el área de la inteligencia artificial que busca encontrar el mejor camino de un punto a otro en mapas representados digitalmente. Se han desarrollado distintos algoritmos con el objetivo de aminorar el tiempo y los recursos utilizados en esta tarea. En este proyecto se busca probar el rendimiento de uno de estos algoritmos, JPS (Jump Point Search) desarrollado por Daniel Harabor y Alban Grastien. El algoritmo se compara con A*, utilizando tres distintas heurísticas.

Palabras Clave: Pathfinding, A, JPS.*

Abstract

Pathfinding is the artificial intelligence area that seeks to find the best path from one point to another in a digitally represented map. A large number of algorithms have been developed in order to minimize the time and the resources used in this task. This project seeks to test the performance of one of them, JPS (Jump Point Search), developed by Daniel Harabor y Alban Grastien. The algorithm is compared to A*, using three different heuristics.

Keywords: Pathfinding, A, JPS.*

Índice

Lista de Ilustraciones

Lista de Tablas

Lista de Abreviaturas

A*: A asterisco.

BG: Baldur’s Gate II.

IDE: Integrated Development Environment.

JPS: Jump Point Search.

RTS: Real Time Strategy.

1. Introducción

El encontrar caminos de un punto a otro es muy importante en la tecnología actual. Se le llama Pathfinding al área de la inteligencia artificial que se encarga de solucionar este problema. Pathfinding principalmente busca encontrar un camino de un punto a otro lo más rápida y eficientemente posible. Se pueden aplicar distintos enfoques y algoritmos, donde se logran resultados distintos, generalmente para resolver objetivos distintos, dependiendo si se busca un camino óptimo o cercano al óptimo.

En este documento se describirá Pathfinding aplicado a los videojuegos. Área en donde cada vez es más requerido encontrar mejores algoritmos que ayuden a ahorrar recursos y mejorar la experiencia del jugador, con mayor rapidez y naturalidad en el movimiento.

Para poder utilizar los distintos algoritmos de Pathfinding, primero, es necesario transformar o resumir el problema o el mapa con él que se trabajará a una estructura que sea fácil de procesar en un lenguaje de programación. Para lograr esto, existen los llamados mapas malla, o mallas mapa, que representan generalmente un grafo donde se definen los lugares por donde se puede o no cruzar, y cuan costoso es hacerlo. Para este trabajo se utilizará el mapa cuadriculado de costo uniforme, donde se dividirá el mapa en una cuadrícula, en la que los sub cuadrados, o nodos, tendrán todos un mismo costo para moverse en forma recta de uno a otro y un mismo costo para hacerlo en forma diagonal.

Se describirán los principales algoritmos utilizados en Pathfinding, como A*, que es la base de los métodos que se utilizan actualmente para encontrar el camino de un punto a otro. Junto con esto se explicará la importancia de la elección de la heurística a utilizar con A*, explicado cómo se diferencian los distintos enfoques utilizados en cada una de ellas.

En seguida, se describirá cómo funciona el algoritmo Jump Point Search, o JPS, que será comparado con A*, puesto que, según sus autores, parece ser superior o muy parecido a los algoritmos más populares que se utilizan en el desarrollo de videojuegos.

Finalmente, se explicará cómo se desarrollaron las pruebas para evaluar a JPS en comparación a A*, utilizando tres heurísticas (Manhatan, Distancia Diagonal y Romper Casillas), los mapas utilizados y los parámetros de comparación. Se evaluarán los resultados de las pruebas, y se concluirá cómo fue el desempeño de los algoritmos probados.

2. Descripción de Pathfinding

La Inteligencia Artificial (IA) o Artificial Intelligence (AI) en inglés, es una rama de la informática que estudia procedimientos automatizados para lograr que un autómata logre ser o parecer inteligente [1]. Éstos, tratan de imitar diversas áreas del comportamiento humano, con el fin de acercarse o llegar a superar a una persona común y corriente, por ejemplo, un experto en cierta materia, en la toma de decisiones o diversas tareas que un sistema computacional puede realizar mejor o más rápido. Es en este contexto que la inteligencia artificial se hace presente en áreas como la robótica y videojuegos, en donde se busca lograr un comportamiento inteligente en un robot o un personaje virtual. Un aspecto básico que se tiene que cumplir, en ambas áreas, es ser capaz de moverse de un punto a otro, esquivando, de forma inteligente y natural, obstáculos que se pueden presentar. Para resolver esta problemática existe un área de la inteligencia artificial llamada Pathfinding.

Pathfinding provee una base importante para gran parte de los sistemas de navegación, se utiliza en áreas de robótica, y especialmente en videojuegos. El principal objetivo es encontrar el mejor camino posible de un punto inicial a un punto final en representaciones del entorno llamadas mapas malla. [2].

Algoritmos que definan el movimiento de personajes de videojuegos o robots son esenciales para que éste parezca lo más cercano a la realidad posible, haciéndolo parecer inteligente. Estos algoritmos tienen que definir por donde dirigirse, basados en la información que se tiene del ambiente, representada en planillas para facilitar su procesamiento.

En el caso de los videojuegos, desde sus inicios ha sido esencial logar movimientos lo más real posible. Es así como en los personajes, lograr que encuentren un camino apropiado de un punto a otro, aunque parezca una tarea sencilla, puede llegar a ser muy compleja, ya que se tiene que esquivar obstáculos y elegir entre quizás cientos de distas rutas el camino más rápido posible en la menor cantidad de tiempo, haciendo parecer esta tarea tan simple y fácil como le puede resultar a un ser humano. En el caso de realizar esta tarea de forma incorrecta, el usuario puede verse frustrado por la lentitud o la poca inteligencia de los personajes que podrían chocar con paredes o girar sobre sí mismos, detalles que pueden arruinar la experiencia total del juego y llevarlo a ser un fracaso. [3]

Actualmente, aunque el hardware ha evolucionado bastante, cada vez se hace más necesario el investigar sobre mejores algoritmos de Pathfinding. Con el aumento de memoria y procesamiento, los mapas se van haciendo más grandes y complejos, con más procesos que hacer simultáneamente y más personajes que mover. Por lo que es muy importante no desperdiciar los recursos que se disponen en el desarrollo de software en ningún área, incluyendo Pathfinding, que resulta esencial, ya que en gran parte de los juegos existe la necesidad de moverse de un punto a otro como componente fundamental de ellos.

3. Definición de Objetivos

A continuación se presentan los objetivos del proyecto.

3.1. Objetivo General

Comparar el rendimiento de los algoritmos JPS (Jump Point Search) y A* con distintas heurísticas.

3.2. Objetivos Específicos

- Explicitar los conceptos principales en el área de Inteligencia Artificial, profundizando en Pathfinding.
- Especificar los aspectos más importantes de JPS y A*, junto con las heurísticas de Manhatan, Distancia Diagonal y Romper Casillas.
- Comparar los aspectos más importantes en los algoritmos JPS y A*, utilizando las heurísticas de Manhatan, Distancia Diagonal y Romper Casillas en cada uno.
- Concluir las ventajas y desventajas de los algoritmos JPS y A*, en base a los aspectos definidos anteriormente y los resultados de las pruebas realizadas.

4. Algoritmos de Búsqueda en Pathfinding

Existen técnicas que resuelven el problema de búsqueda de caminos simplificados en mallas compuestas por elementos similares entre sí, como cuadrados u otras figuras. Los algoritmos de Pathfinding de los textos de informática trabajan con grafos en el sentido matemático (un conjunto de nodos que se conectan entre sí). Un mapa cuadriculado de jugo puede ser considerado un grafo, con cada casilla siendo un nodo y los bordes son los conectores a los que están adyacentes. A continuación se presentan los enfoques más comunes sobre la forma de enfrentar este problema. Hay que tener en consideración que lo que se explicará a continuación considera sólo los mapas en dos dimensiones, puesto que los algoritmos con los que se trabaja en este proyecto están diseñados para ese entorno.

4.1. Algoritmos Estándar en Pathfinding

Los algoritmos descritos a continuación son la base para resolver el problema de encontrar un buen camino de un punto a otro en una malla o grafo de búsqueda. El algoritmo de Dijkstra es la base para A*, el algoritmo más conocido en Pathfinding.

4.1.1. Algoritmo de Dijkstra

A continuación se procederá a describir de forma simplificada el algoritmo de Dijkstra. La entrada es un grafo G = (V, E) y el nodo inicial es s que pertenece a V. V es el set de nodos y E es el set de caminos o conexiones entre cada nodo dentro de G. Adicionalmente se tiene una función para pesar los nodos w (u, v) ≥ 0 para todo (u, v) que pertenezca a E. El algoritmo divide los nodos, V, en dos conjuntos disjuntos (que no tienen elementos en común), V = R ∪ Q. El conjunto R, contiene los nodos donde los pesos de los caminos más cortos finales ya fueron determinados, y el conjunto Q = V-R. Cuando se mueve un nodo de Q a R, se dice ‘retirado’. Algunas descripciones del algoritmo llaman a Q el conjunto ‘abierto’ y R el conjunto ‘cerrado’. Para cada nodo, v, se guarda el valor estimado del camino más corto actual, llamado g(v). Ya que no es relevante encontrar el valor del camino menos costoso, sino los nodos que lo componen, también se guarda, para cada nodo, su nodo predecesor, llamado ed[v], que es la unión desde un vecino al nodo en el mejor camino actual al nodo en cuestión. Esto es utilizado para reconstruir el camino retrocediendo desde el destino. En caso de que exista más de un camino óptimo, sólo el que fue descubierto más recientemente será recordado en este esquema. [4]

El conjunto Q debería ser implementado como una cola con prioridad con las siguientes operaciones: Q.insertar (u), que inserta un nodo a la cola, Q.ExtraerMínimo(), que saca el elemento v que pertenece a Q con el menor g[v], W. DecrementarLlave(v, a), que decrementa g[v] para a, y actualiza la cola. [4]

El algoritmo funciona repetidamente extrayendo el nodo v ∈ Q con el camino más corto estimado mínimo, g[v]. Ya que todas las uniones son positivas, no se puede encontrar ningún camino extendiendo un camino desde cualquier otro nodo. Así se determina el camino más corto a v y se puede retirar de R. Entonces se puede comprobar si cualquiera de las extensiones de este camino óptimo a sus vecinos es un camino más corto que la estimación del

camino más corto actual para el respectivo vecino. Si así fuera, entonces se actualiza el camino más corto estimado y la conexión con el predecesor para los vecinos según corresponda. Debido a que repetidamente se selecciona el nodo menos costoso y se actualiza él y sus vecinos, el algoritmo se dice que es 'greedy', ya que se comprueba el mejor camino localmente. [4]

4.1.2. Algoritmo A*

Como el objetivo es encontrar solamente el camino más corto entre dos nodos, parece una pérdida de tiempo y recursos procesar todos los caminos desde un nodo a todos los otros como lo hace el algoritmo de Dijkstra. El conocimiento de saber la posición del nodo destino no se ocupa para nada durante la búsqueda. El algoritmo A*, se hace cargo de esto agregando una heurística para guiar la búsqueda hacia el destino, h[v], utilizando f[v]= g[v] + h[v] en vez de sólo g[v]. El algoritmo A* será explicado en detalle en el próximo capítulo. [4]

4.2. Heurísticas en Pathfinding

Una heurística estima el costo mínimo desde un nodo al objetivo. Dependiendo de cual se utilice, puede ayudar o perjudicar bastante la búsqueda del camino apropiado.

4.2.1. Heurística de Manhattan

Esta es la heurística estándar para mallas cuadradas. Si D es el costo mínimo para moverse a un nodo adyacente, la heurística es el valor absoluto de la coordenada X del nodo actual menos la coordenada X del nodo objetivo, sumado a lo mismo pero con la componente Y, con todo eso multiplicado por D.[4]

```
function heuristic(node) =
    dx = abs(node.x - goal.x)
    dy = abs(node.y - goal.y)
    return D * (dx + dy)
```

4.2.2. Distancia Diagonal

Esta heurística considera los movimientos diagonales. Acá se cuenta el costo del número de pasos que se toman si no se puede ir diagonalmente, a lo que se resta el costo de los pasos que se ahorraron usando el movimiento diagonal. El mínimo entre el valor absoluto de la resta de las componentes X del nodo actual y el objetivo, lo mismo con las componentes Y, es la menor cantidad de pasos diagonales que puede tomar, y cada uno de estos ahorra dos veces el valor de un movimiento recto, por lo que se restan.[4]

```
function heuristic(node) =
    dx = abs(node.x - goal.x)
    dy = abs(node.y - goal.y)
    return D * (dx + dy) + (D2 - 2 * D) * min(dx, dy)
```

4.2.3. Rompe Casillas

En A*, normalmente se tienen que expandir los nodos que no están cerca del objetivo puesto que estos tienen un menor costo f. Pero si el valor de la heurística se aumenta sólo un poco, los nodos que están más cerca de la meta serán escogidos primero, y por ende, la búsqueda sería más rápida. A esto se le llama Romper Casillas. Una manera de realizar este procedimiento, es subiendo levemente el valor de H, multiplicándolo por 1 + P, siendo P el valor del costo mínimo de dar un paso dividido por el máximo valor esperado del largo de un camino en el mapa. Cuando se utiliza este método y se encuentran obstáculos, el algoritmo tiende a funcionar igual que A*, pero cuando no los hay se tiende a mejorar el rendimiento.[4]

5. A* en Pathfinding

Debido a la importancia de A* en Pathfinding y a que se ocupa como base en el algoritmo que se comparará (Jump Point Search), se procederá a explicar detalladamente cómo funciona este algoritmo para encontrar un camino de un punto a otro en una malla o grafo representativo de un ambiente básico. Esta explicación está basada en el texto "A* Pathfinding for Beginners", por Patrick Lester del año 2005.

5.1. Área de Búsqueda

Se quiere ir desde el punto A hasta el punto B. Un muro separa estos dos puntos. El ejemplo queda ilustrado en el gráfico siguiente, donde el recuadro verde es el punto de inicio A, el rojo es el punto de destino B y la zona azul el muro que hay entre ambos, como se muestra en la Ilustración 5.1. [5]

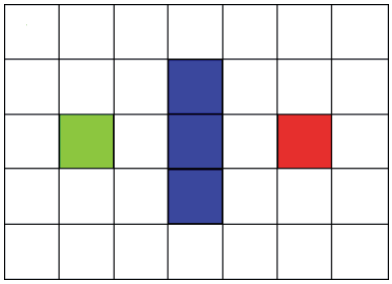

Ilustración 5.1 A* inicio

Se dividió el área de búsqueda en una rejilla o malla cuadrada. Simplificar el área, tal y como se muestra en la Ilustración 5.1, es el primer paso en un Pathfinding. Este método reduce el área de búsqueda a una matriz bidimensional. Cada elemento de la matriz representa uno de los cuadrados de la malla, y su estado se almacena como transitable o intransitable. El camino se elige calculando qué casillas se debieran cruzar para ir desde A hasta B. Una vez que el camino haya sido encontrado, lo que se quiera desplazar de A hasta B se moverá desde el centro de una casilla hacia el centro del siguiente hasta que el objetivo haya sido alcanzado. [5]

5.2. Inicio de la Búsqueda

Una vez que se ha simplificado el número de nodos en una malla como se muestra en la Ilustración 5-1, el siguiente paso es dirigir una búsqueda para encontrar el camino más corto. En Pathfinding A*, se comienza desde el punto A, comprobando las casillas adyacentes y generalmente buscando hacia fuera hasta que se encuentre el destino. [5]

Se comienza la búsqueda realizando los siguientes pasos:

1. El punto inicial A se agrega a la "lista abierta" de cuadrados a tener en cuenta. La lista abierta es como una lista de compra. Ahora mismo sólo se tiene un elemento en la lista, pero más tarde se agregarán más. La lista contiene los nodos o casillas de la Ilustración 5.1 que podrían formar parte del camino que

se quiere tomar, pero que quizás no lo hagan. Básicamente, esta es una lista de los cuadrados que necesitan ser comprobados. [5]

2. Todos los nodos adyacentes que son transitables, o que no son obstáculos, se agregan a la lista abierta. Por cada nodo guardado se tiene que guardar su 'padre' o de donde proviene.[5]
3. Se extrae el nodo inicial A desde la lista abierta y se agrega a una "lista cerrada" de nodos que ya no son necesarios para analizar.[5]

En la Ilustración 5.2, el cuadrado verde del centro es el nodo de inicio. Está bordeado el azul claro para indicar que el cuadrado ha sido añadido a la lista cerrada. Todos los cuadros adyacentes están ahora en la lista abierta para ser comprobados, y están bordeados con verde. Cada uno tiene un puntero negro que señala a su padre, el cual es el cuadro inicial. [5]

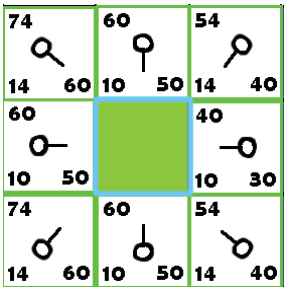

Ilustración 5.2 Nodo de inicio A* con punteros.

Se escoge el nodo adyacente con el menor costo o F(N).

5.3. Puntuación del Camino

Para determinar el camino a seguir se aplica la siguiente ecuación:

F(N) = G(N) + H(N).

En donde:

- G(N) es el costo del movimiento para ir desde el punto inicial A a un cierto cuadro de la malla, N, siguiendo el camino generado para llegar allí.[5]

- H(N) es el costo del movimiento estimado para ir desde ese cuadro de la malla, N, hasta el punto final B. Ésta es la llamada heurística, puesto que es una suposición. Se pueden determinar muchas heurísticas diferentes que pueden cambiar bastante el rendimiento de A*, pero en este texto se seguirá sólo un ejemplo (quizás el más sencillo).[5]

El camino se genera visitando repetidamente la lista abierta y escogiendo el nodo con el menor valor F(N). En este ejemplo se le asignará un costo de 10 a cada movimiento horizontal o vertical, y un costo de 14 a los movimientos diagonales. Se utiliza el número 14 porque la distancia para moverse en diagonal es raíz de dos, o aproximadamente 1.414 veces lo que se mueve verticalmente, en este caso 10, por lo que 10 multiplicado por 1.414 es 14.14, lo que aproximadamente es 14 para trabajar de forma más simple. [5]

Para calcular el costo G(N) de la ecuación nombrada anteriormente, se suma el costo G del padre, y entonces se suma 10 o 14 dependiendo si el movimiento es horizontal o vertical, o diagonal respectivamente. [5]

H(N) se puede estimar de muchas formas. El método que se utiliza acá es llamado el método Manhattan, donde se calcula el número total de nodos que se mueven horizontalmente y verticalmente para alcanzar el nodo objetivo del nodo actual, ignorando los movimientos diagonales, y los obstáculos que se puedan interponer. Entonces se multiplica el total por 10 (costo designado para movimientos horizontales y verticales). La heurística H tiene que ser menor igual al costo mínimo del camino real, en caso contrario no se garantiza que se pueda encontrar un camino óptimo y la heurística se declara inadmisible. La heurística de Manhattan es inadmisible en este caso, pero como la sobreestimación es mínima, y porque A* es más fácil de entender con este método se ignorará este detalle en este ejemplo. [5]

En la Ilustración 5.3 se puede ver el primer paso de la búsqueda que se está realizando. Los puntajes F(N), G(N) y H(N) están escritos para cada nodo. Como se indica en el recuadro inmediatamente a la derecha del nodo inicial de la Ilustración 5.3, F(N) está en la esquina superior izquierda, G(N) en la inferior izquierda y H(N) en la inferior derecha. [5]

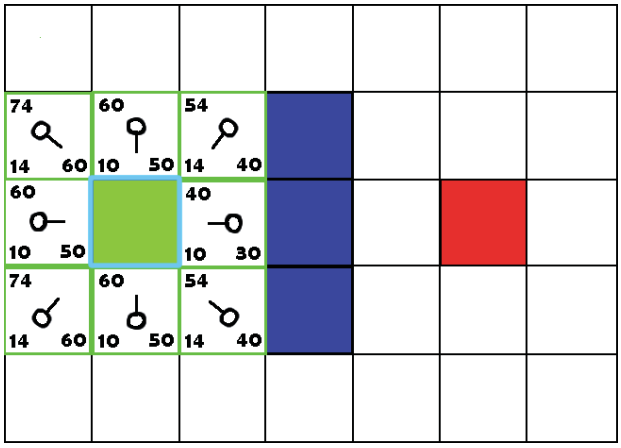

Ilustración 5.3 Primer ciclo A*.

Continuando con la búsqueda, simplemente se escoge el nodo con el puntaje F(N) más bajo de todos los que están en la lista abierta. En este proyecto, en el caso de que se existan varios nodos con un mismo F(N) mínimo, se elige el nodo que se agregó a la lista abierta primero. Con el nodo seleccionado se realiza lo siguiente:

1. Sacarlo de la lista abierta y agregarlo a la lista cerrada.[5]

2. Analizar todos los nodos adyacentes, ignorando los que están en la lista cerrada o son obstáculos, añadir nodos a la lista abierta si es que aun no lo están. Hacer del nodo actual el padre de los que se agregaron a la lista abierta.[5]

3. Si un nodo adyacente ya pertenecía a la lista abierta, comprobar si el camino que se está siguiendo con el nodo actual es un mejor camino al que ya estaba designado. En otras palabras, comprobar si el puntaje G(N) para ese nodo es

menor si se usa el nodo actual para llegar a él. Si no es así hay que hacer nada. Por otro lado, si el costo G(N) resulta ser menor, hay que cambiar el padre del nodo adyacente, donde se está comprobando G(N), como se observa en la Ilustración 5-3 cambiando el puntero para seleccionar el nodo actual. Finalmente, recalcular los puntajes F(N) y G(N) de ese nodo. [5]

Se repite el proceso hasta que se agrega el nodo objetivo a la lista abierta. En este caso se vería como se muestra en la Ilustración 5.4.

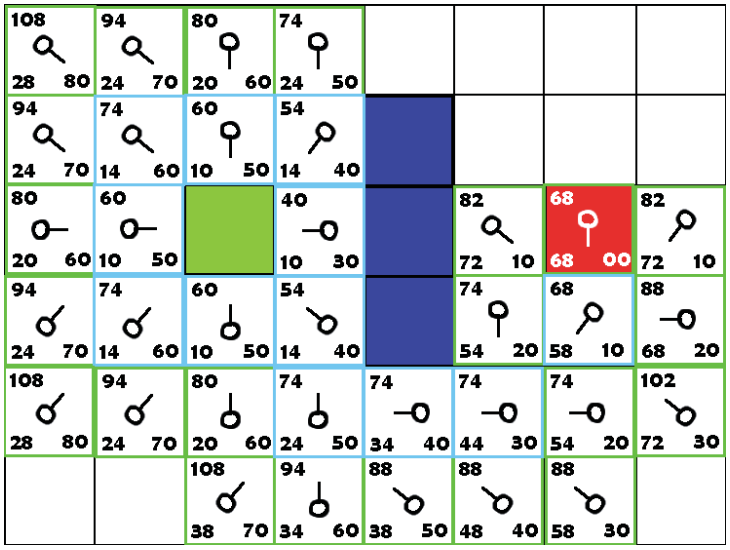

Ilustración 5.4 A* cuando el objetivo se ha encontrado.

Para lograr encontrar el camino, desde el cuadro objetivo (rojo en las ilustraciones), se sigue el puntero al padre de cada nodo, lo que llevará de vuelta al nodo inicial. Y los movimientos que se realizaron serían el camino a seguir. Camino que se aprecia en la Ilustración 5-5. [5]

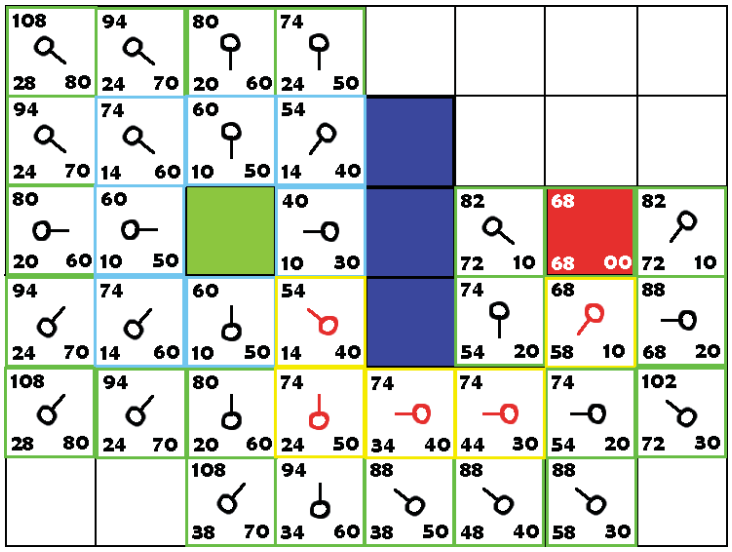

Ilustración 5.5 A* cuando se ha encontrado el camino.

Recapitulando, A* en Pathfinding se resume a lo siguiente [5]:

1. Se añade el nodo inicial a la lista abierta.

2. Repetir lo siguiente:

   a) Se busca el nodo con el costo F(N) más bajo en la lista abierta (nodo actual).

   b) Cambiarlo a la lista cerrada.

   c) Para cada uno de los 8 nodos adyacentes a la casilla actual:

   - Si no es transitable o si está en la lista cerrada, ignorarlo. En cualquier otro caso se realiza lo siguiente:
     - Si no está en la lista abierta, añadirlo a la lista abierta. El nodo actual pasa a ser el padre de este nodo. Se almacenan los costos F(N), G(N) y H(N) de la casilla.
     - Si ya está en la lista abierta, se comprueba si el camino para esa casilla es mejor usando el costo G(N) a través del actual. Un costo G(N) menor significa que éste es un mejor camino. Si es así, se cambia el padre del cuadrado al cuadro actual y se recalcula G(N) y F(N) del nodo.

   d) Se detiene la búsqueda para cuándo se añada el cuadro objetivo a la lista cerrada, en cuyo caso el camino ha sido encontrado, o no se encuentre la casilla objetivo y la lista abierta esté vacía. En este caso no hay camino.

3. Se guarda el camino. Se mueve hacia atrás desde la casilla objetivo, se mueve desde cada nodo a su padre hasta que se alcance la casilla inicial. El camino seguido es el buscado.

5.4. Desventajas de A*

Hay situaciones donde A* podría no rendir bien por una variedad de razones. Los requerimientos en tiempo real de los video juegos, además de las limitaciones en memoria y tiempo de procesador en algunos de ellos, podrían hacer más difícil a A* funcionar bien. Un mapa grande podría requerir miles de entradas en las listas abiertas y cerradas, y podría no haber espacio suficiente para eso. Incluso si hay suficiente memoria para ello, los algoritmos usados para manipularlos podrían ser ineficientes. [6]

La calidad de A* depende de la calidad en la heurística estimadora H(N), si se acerca bastante al verdadero costo del camino restante, su eficiencia será alta, por otro lado, si es muy bajo, su eficiencia disminuye bastante. [6]

A* es la base del principal algoritmo que se analizará en este proyecto, algoritmo llamado Jump Point Search.

6. Algoritmo Jump Point Search (JPS)

Jump Point Search o JPS puede ser descrito como un macro operador que identifica y selectivamente expande sólo ciertos nodos del mapa cuadriculado, nodos que son llamados Jump Points. Este algoritmo fue elegido por lo que prometen los creadores acerca de su fácil adaptabilidad a otros algoritmos, y buen rendimiento. A continuación se describirá JPS basándose en un paper presentado por sus autores Daniel Harabor y Alban Grastien.

6.1. Trabajos Relacionados

Muy pocos trabajos explícitamente identifican y se enfrentan a la simetría en áreas del Pathfinding como los mapas malla.

'Empty Rectangular Rooms' es una técnica para romper las simetrías offline, que trata de enfrentar esto desde un punto de vista lejano. Descompone los mapas malla en una serie de rectángulos libres de obstáculos, y reemplaza todos los nodos del interior de cada rectángulo con un conjunto de bordes macro que facilitan el viaje óptimo. Este enfoque es especifico para mapas con 4 conexiones, y es menos general que la poda de JPS. También se requiere de un pre procesamiento.[8]

Los métodos **'Dead-End Heuristic'** y **'Swamps'** son dos técnicas de poda similares relacionadas a JPS. Ambas descomponen la malla en una serie de áreas adyacentes. Después, la descomposición es usada para identificar áreas que no son relevantes para resolver óptimamente una instancia particular de la búsqueda.[9] [10]

Un metodo diferente para podar el espacio de búsqueda es identificar las casillas 'muertas' y 'redundantes'. Desarrollado en el contexto de heurísticas de búsqueda basadas en el aprendizaje, este método acelera la búsqueda sólo cuando múltiples iteraciones de un algoritmo son realizadas. [11]

'Fast Expansion' es otro trabajo relacionado que acelera A* óptimo. Evita operaciones innecesarias en la lista abierta cuando encuentra un nodo sucesor tan bueno o mejor que el mejor nodo en la lista abierta. Los Jump Point son similares, pero fundamentalmente distintos, permiten identificar una gran cantidad de nodos que normalmente serian expandidos que pueden ser omitidos.[12]

En casos donde no se requiere un camino óptimo, los métodos de Pathfinding jerárquicos son populares. Éstos mejoran el rendimiento descomponiendo el espacio de búsqueda, usualmente offline, en aproximaciones mucho más pequeñas. Algunos algoritmos de este tipo como HPA* son rápidos y en cuanto a memoria son eficientes, pero no son óptimos.[13]

6.2. Simetrías

Ampliamente utilizado en áreas como la robótica, inteligencia artificial y los videojuegos, el mapa cuadriculado de costo uniforme es una forma muy popular de representar ambientes en Pathfinding. Las mallas son académicamente interesantes, ya que entre dos

lugares, usualmente existen un gran número de posibles caminos. A veces estos caminos representan formas alternativas de alcanzar un punto desde otro, pero a menudo son caminos simétricos en el sentido que sólo difieren en el orden en que los movimientos aparecen. [1]

Se entiende por camino en una malla al orden secuenciado de vectores, donde cada vector representa un solo pasó desde el nodo en la malla a su nodo vecino adyacente. En un grafo, un vector sería la unión entre dos nodos. [1]

Simetría en los caminos se entiende cuando dos o más caminos en la malla comparten el comienzo y el fin, y además de un camino se puede formar el otro cambiando el orden de sus vectores constituyentes. [1]

Por ejemplo si para llegar de un punto A a un punto B existen un camino 1 en el cual se realizaron los siguientes movimientos; Arriba, Arriba, Abajo, Abajo. Pero también existe el camino 2 que es Arriba, Abajo, Abajo, Arriba. Los camino 1 y 2 tienen ambos 2 pasos arriba y 2 pasos abajo en total, por lo que se dice que éstos son simétricos. Un camino 3; Abajo, Abajo, Abajo, Arriba no es simétrico ya que tiene tres pasos abajo y uno arriba, a diferencia del camino 1 y 2.

En presencia de la simetría, un algoritmo de búsqueda como A* es forzado a explorar cada nodo alrededor de cada camino óptimo. Se dan casos en que A* podría expandir cada nodo en el mapa entero antes de alcanzar el objetivo. Además, A* usualmente considera muchos otros nodos que parecen prometedores pero que no están en ningún camino óptimo. Por ejemplo, cada vez que A* expande un nodo del borde de la búsqueda, probablemente ya haya encontrado casi todos los caminos simétricos que llevan a ese nodo (dependiendo de la heurística ocupada cuando dos nodos son igualmente buenos). Pero este esfuerzo es en vano si estas expansiones no llevan a búsqueda cerca del nodo objetivo, sino que a un camino sin salida. Así nace la pregunta que JPS responde. ¿Cómo manejar la simetría en Pathfinding en mallas mapa? [1]

6.3. Métodos Existentes para Acelerar la Búsqueda

Un gran número de técnicas se han propuesto para acelerar Pathfinding. La mayoría pueden ser clasificadas como variaciones de los siguientes temas [1]:

a) Reducir el tamaño de la búsqueda: Algoritmos de este tipo son rápidos y ocupan poca memoria, pero los caminos resultantes usualmente no son óptimos y deben ser refinados con un segundo análisis.

b) Mejorar la precisión de la función o funciones heurísticas que guían la búsqueda: Los algoritmos de este tipo usualmente pre-procesan y almacenan las distancias entre pares de nodos en el ambiente. A pesar de ser rápidos y óptimos, estos métodos pueden implicar grandes usos de memoria, lo que a menudo no es deseable.

c) Detección de un camino sin salida y otros métodos de poda de espacios de búsqueda: Los algoritmos de este tipo usualmente apuntan a identificar áreas en el mapa que no necesitan ser exploradas para alcanzar el objetivo final. A pesar de no ser tan rápidos

como la alternativa a) o b), estos métodos usualmente tienen bajos requerimientos de memoria y pueden mejorar el rendimiento de la búsqueda de caminos en muchos factores.

JPS puede ser clasificado, a grandes rasgos, en el área de poda de espacios de búsqueda. Donde se difiere de los métodos existentes en que, en lugar de tratar de identificar áreas que no tiene que ser cruzadas durante la búsqueda, se identifican y cortan nodos simétricos que previenen la exploración rápida de un área. [1]

6.4. Notación y Terminología

Se trabajará con mapas mallas de costo uniforme. Cada nodo tiene 8 vecinos o menos, y cada uno puede ser atravesable o no. Cada movimiento derecho (vertical u horizontal), desde un nodo atravesable a uno de sus vecinos, tiene un costo de 1, el costo de los movimientos diagonales es √2. Los movimientos que involucran nodos no atravesables (obstáculos) no están permitidos. La notación $\underset{d}{\rightarrow}$ se refiere a una de las ocho direcciones disponibles para moverse. Se escribe y = x + k$\underset{d}{\rightarrow}$ cuando el nodo y puede ser alcanzado tomando k unidades de movimiento desde el nodo x en dirección $\underset{d}{\rightarrow}$. Cuando $\underset{d}{\rightarrow}$ es un movimiento diagonal, se denota los dos movimientos diagonales a 45 grados a $\underset{d}{\rightarrow}$ como $\underset{d_1}{\rightarrow}$ y $\underset{d_2}{\rightarrow}$. [2]

Un camino $\pi = (n_0, n_1, \ldots, n_k)$ es una caminata ordenada libre de ciclos que comienza en n_0 y termina en n_k. Se usará a veces el operador setminus en el contexto de un camino: por ejemplo π\x. Que quiere decir que x no aparece en el camino. Se hará uso de la función len para referirse al largo (o costo) de un camino y la función dist para referirse a la distancia entre dos nodos en la malla: por ejemplo len(π) o dist(n_0, n_k) respectivamente. [2]

6.5. Jump Points

JPS puede ser descrito en término de dos reglas simples de poda que son aplicadas recursivamente durante la búsqueda: una regla es específica para los pasos derechos y la otra para pasos diagonales. Lo básico en ambos casos es cortar el set de vecinos inmediatos alrededor de un nodo tratando de probar que un camino óptimo (simétrico o no) existe desde el padre del nodo actual a cada vecino y que el camino no involucra visitar el nodo actual. Como se explica en la Ilustración 6.1. [1]

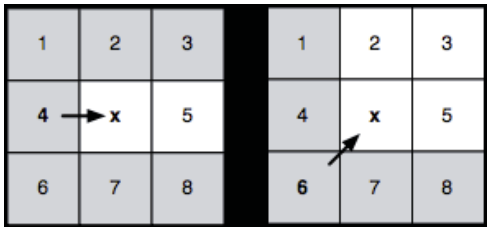

Ilustración 6.1 Poda de vecinos[1].

En la Ilustración 6.1, el nodo x está actualmente siendo expandido por A*. La flecha indica la dirección de viaje desde su padre, ya sea recta o diagonal. En ambos casos se puede cortar directamente todos los vecinos grises ya que éstos puedes ser alcanzados óptimamente desde el padre de x sin pasar por el nodo x. Es en este paso donde se elimina la simetría. [2]

Al grupo de nodos que permanece después de la poda se le llama vecinos naturales del nodo actual. Estos son los que quedaron marcados en blanco en la Ilustración 6-1. Idealmente, sólo se considerará el grupo de vecinos naturales durante la expansión. Sin embargo, en algunos casos, la presencia de obstáculos podría implicar que un grupo adicional, de hasta k nodos ($0 \leq k \leq 2$). A estos nodos se les llama vecinos forzados del nodo actual. La Ilustración 6.2 explica esta idea. [1]

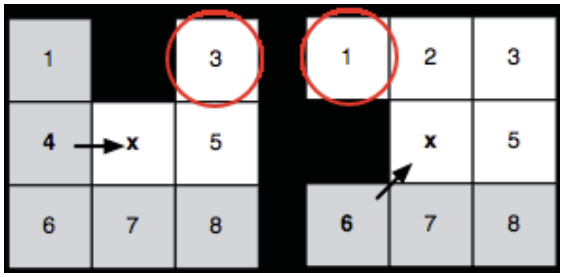

Ilustración 6.2 Vecinos forzados [1].

En la Ilustración 6.2, el nodo x está siendo expandido. La flecha indica la dirección de viaje desde su padre, ya sea recta o diagonal. Nótese que cuando x es adyacente a un obstáculo, los vecinos en el círculo rojo no pueden ser podados, cualquier camino óptimo alternativo, desde el padre de x para cada uno de estos nodos resaltados, está bloqueado.

Durante la búsqueda, se aplican las reglas de poda como se establece a continuación: en vez de generar cada vecino natural y forzado, se corta recursivamente el grupo de vecinos alrededor de cada nodo. Intuitivamente, el objetivo es eliminar las simetrías recursivamente "saltando" todos los nodos que pueden ser accedidos óptimamente por un camino que no pasa por el nodo actual. Se detiene la recursión cuando se da con un obstáculo o cuando se encuentra con un nodo llamado Jump Point, o puntos de salto, en español. Los Jump Points son interesantes, porque son nodos con vecinos que no pueden ser alcanzados por un camino alternativo simétrico, el camino debe pasar a través del nodo actual.[1]

Los detalles del algoritmo de poda recursivo son razonablemente constantes: para asegurar que se logren resultados óptimos, se necesita sólo asignar un orden en cómo se procesan los vecinos naturales (derechos antes de los diagonales). Las ilustraciones 6.3 y 6.4 dan dos ejemplos del algoritmo de poda en acción. En el primer caso se identifica un jump point con recursión derecha, y en el segundo caso se identifica en forma diagonal. [1]

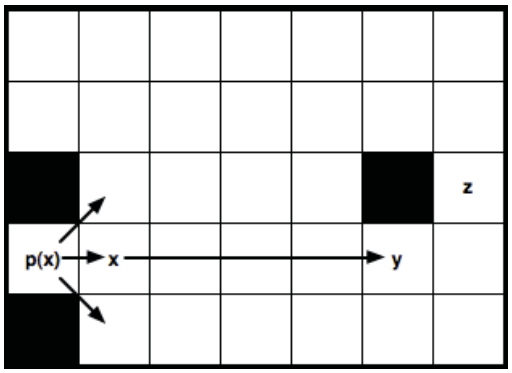

Ilustración 6.3 Salto derecho. [1]

En la Ilustración 6.3, recursivamente se aplica la regla de la poda recta y se identifica y como el sucesor jump point de x. Este nodo es interesante porque tiene un vecino z que no puede ser alcanzado óptimamente, a menos por un camino que visite x y después y. los nodos en el intermedio nunca son expandidos por A* o evaluados de otra forma.

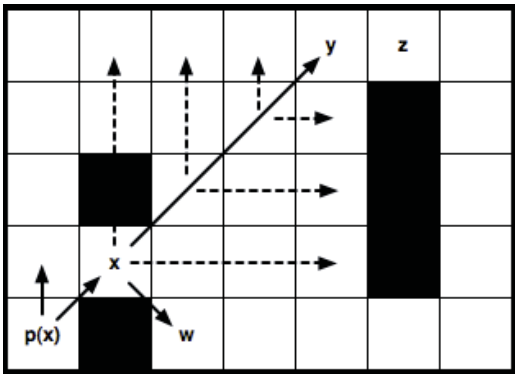

Ilustración 6.4 Salto diagonal. [1]

En la Ilustración 6.4 se aplica la regla de poda diagonal, y se identifica y como un sucesor jump point de x. Nótese que antes de cada paso diagonal, primero se realiza la recursión recta (líneas punteadas). Sólo si ambas recursiones fallan en identificar otro jump point, se realiza el paso diagonal nuevamente. El nodo w, que es un nodo forzado de x, se genera normalmente.

7. Formulación de Pruebas

Con el propósito de evaluar el algoritmo JPS, se realizaron pruebas con el lenguaje de programación C++. Los aspectos a destacar son los siguientes.

7.1. Consideraciones para Pruebas

En la ejecución de las pruebas se consideró lo siguiente:

- El lenguaje escogido para codificar el programa para ejecutar las pruebas fue C++, aunque éste no se dominaba, puesto que, gran parte de los programas de Pathfinding que se conocieron estaban escritos en este lenguaje.
- El IDE utilizado es CodeBlocks versión 12.11, una herramienta disponible gratuitamente.
- El computador donde se realizaron las pruebas tiene las siguientes características: Procesador Intel Core i7 de 2.40 GHz, con 8 GB RAM.
- Se trabajó en el sistema operativo Windows 8.

7.2. Parámetros a Evaluar

Al igual que Daniel Harabor y Albain Grastien en su presentación de JPS, se evaluará el tiempo y la cantidad de nodos expandidos. En ambos casos, mientras menor sea el valor mejor es el resultado.

7.2.1. Tiempo

El software mide el tiempo en segundos en que cada algoritmo logra resolver problemas determinados. Mientras menor sea el tiempo, mejor es el resultado.

7.2.2. Número de Nodos Expandidos

Con el fin de dar indicios de la cantidad de memoria utilizada en nodos, se determinan cuántos nodos se han expandido durante el proceso de búsqueda. Es por esto que todos los nodos que se expandan con A* y JPS serán contados .

7.3. Mapas a Utilizar

Los algoritmos con los que se trabaja, buscan el camino óptimo entre un punto a otro en un entorno determinado. Este entorno comúnmente suele ser la representación de un terreno real, en forma digital, por lo que en él se pueden encontrar, árboles, terreno plano, montañas mar, entre otros. En algunos de estos lugares se puede caminar fácilmente (terreno plano) y por otros es casi imposible atravesarlos a pie (montañas, mar u otro tipo de estructuras como las paredes).

Para poder trabajar con pruebas que entregan resultados lo más cercano a la realidad de

cómo se trabajaría en juegos verdaderos, y que aporten al estado del arte de los algoritmos de Pathfinding, se encontró disponible en internet, un grupo de mapas y problemas creado específicamente para este propósito.

Nathan Sturtevant, que pertenece a la Universidad de Denver, publicó y creó un grupo de mapas y problemas estándares disponible abiertamente para todos los investigadores que deseen utilizarlos. Esto, con el objetivo de disminuir la disparidad entre las distintas pruebas entre las variadas investigaciones.

7.3.1. Descripción del Formato de los Mapas

El formato de archivo “map” usado en el repositorio fue originalmente desarrollado por Yngvi Björnsson y Markus Enzenberger en la Universidad de Alberta. [7]

El formato comienza con el tipo, que siempre es ‘octile’, seguido el alto y ancho del mapa, uno en cada línea, en la siguiente está escrita la palabra ‘map’ y a continuación el mapa en sí mismo. Todas las celdas están bloqueadas o no bloqueadas. Sin embargo, para representar las características del mapa de una mejor forma, se agregaron algunos tipos de terreno. La tierra normal se representa por un punto (‘.’), y para aguas poco profundas se representa con un carácter ‘S’. Estos son los únicos terrenos transitables o pasables. Todo otro terreno es considerado intransitable o impasable, incluyendo árboles (‘T’), agua (‘W’) y terreno fuera de alcance (‘@’). [7] A continuación, en la Ilustración 7.1, en la parte (a) se muestra la forma textual de representar el mapa, y en la parte (b) se muestra una representación gráfica. [7]

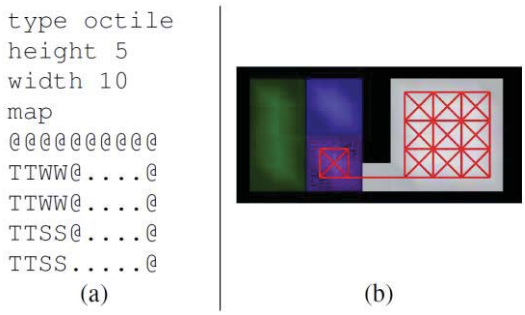

Ilustración 7.1 Representación del mapa escrita y gráfica.

7.3.2. Mapas Escogidos

De los mapas del repositorio elegidos para realizar las pruebas, dos pertenecen a al grupo del videojuego ‘Baldur’s Gate II’, otros dos mapas generados digitalmente del grupo ‘Mazes' (Mapas tipo laberinto) y dos del tipo 'Rooms' .

El primer grupo de escenarios es de Baldur’s Gate II (BG), éste es uno de los conjuntos de mapas más utilizados en papers publicados previamente. BG es un juego de rol publicado por la empresa de videojuegos BioWare Corp. en el año 2000. El juego utiliza mapas representados como malla internamente. Los mapas fueron escalados por Nathan Sturtevant a

un largo y ancho de 512, y generaron un grupo de pruebas con caminos al azar en el mapa. [7]

Los ‘Mazes’ son mapas artificiales. Son laberintos que se forman al azar, influenciado por una serie de parámetros. Todos los mapas de este grupo son de largo y ancho 512. Estos mapas están sub clasificados por el ancho de sus pasillos, pudiendo ser de 1, 2, 4, 8, 16 o 32.

Los 'Rooms', al igual que 'Mazes', son mapas artificiales. La diferencia es que estos mapas están constituidos por una serie de cuadrados o cuartos con salidas a otros cuartos contiguos.

7.3.3. Conjunto de Problemas

Junto con cada mapa, Nathan Sturtevant pone a disposición un conjunto de problemas Pathfinding. En el archivo de formato ".map.scen", en cada linea hay un problema, indicando, entre otras cosas, la posición inicial y final, junto con el largo del camino óptimo para la resolución del problema. Se utilizarán algunos de estos problemas para probar y comparar los algoritmos A* y JPS.

7.4. Pruebas Realizadas

El Software desarrollado, básicamente realiza los siguientes pasos:

- Lee el archivo del mapa y lo traspasa a un arreglo.
- Se lee el archivo de problemas y se extrae un escenario previamente escogido.
- Se resuelve el escenario con A* y se calcula el tiempo y los nodos expandidos.
- Después, se resuelve el mismo escenario con JPS y se calcula el tiempo y los nodos expandidos.
- Los resultados son guardados en un archivo de texto.

A continuación, sólo una parte de los resultados será mostrada. El resto de las tablas se encuentra en el Anexo.

7.4.1. Resultados Obtenidos

En la tabla 7.1, Se muestran los resultados de un conjunto de diez pruebas realizadas en el primer mapa de BG, en ella se muestran los nodos expandidos y el tiempo que demoró cada combinación de algoritmo en segundos, la columna final, en color verde, muestra el largo del camino óptimo para cada escenario. Se muestra cada algoritmo con cada una de las heurísticas, y por cada combinación de éstas se muestran la cantidad de nodos expandidos en la columna '*Nodos*' y el tiempo en segundos bajo la columna *'Tiempo(seg)*'. Por cada uno de los seis mapas utilizados existe una tabla como la 7.1.

En la Ilustración 7.2 se grafican los resultados de A*, contrastando con cada heurística, del mapa del primer mapa de BG, o BG 1. El eje horizontal representa el largo del camino en nodos y el eje vertical es el tiempo en segundos. En la Ilustración 7.3 se muestra lo mismo pero con JPS. Cada combinación del algoritmo con una heurística es representada con un color distinto, ambos gráficos de las ilustraciones 7.2 y 7.3.

Tabla 7.1 Resultados mapa 1 BG.

Prueba	A* Manhattan		JPS Manhattan		A* Distancia Diagonal		JPS Distancia Diagonal		A* Rompe Casillas		JPS Rompe Casilla		Largo Camino (nodos)
	Nodos	*Tiempo(seg)*	*Nodos*	*Tiempo(seg)*	*Nodos*	*Tiempo(seg)*	*Nodos*	*Tiempo(seg)*	*Nodos*	*Tiempo(seg)*	*Nodos*	*Tiempo(seg)*	
1	27	0,001001	5	0,010007	27	0,001	5	0,009006	27	0,001001	5	0,009006	34
2	356	0,019013	9	0,015009	1716	0,284189	8	0,01401	1716	0,288192	8	0,01401	129
3	117	0,005004	11	0,018013	272	0,13009	11	0,018013	272	0,013008	11	0,018012	164
4	765	0,064043	13	0,022016	3545	1,52602	9	0,017011	3545	1,53202	9	0,016011	176
5	5909	4,6351	12	0,017012	9607	12,4493	28	0,019013	9607	124944	28	0,018012	214
6	16424	36,3693	18	0,024016	18044	44,0435	21	0,026018	18044	43,7823	21	0,034024	258
7	3704	1,66711	12	0,023014	7813	8,41363	20	0,034024	7813	8,41263	20	0,029019	301
8	2626	0,82256	22	0,027017	8131	8,73884	37	0,03002	8131	8,7008	37	0,03002	312
9	236	0,018013	19	0,029019	4818	3,04203	20	0,029019	4818	3,04504	20	0,02902	316
10	17927	43,7823	39	0,021014	18304	45,4184	42	0,014008	18304	45,3443	42	0,01401	329

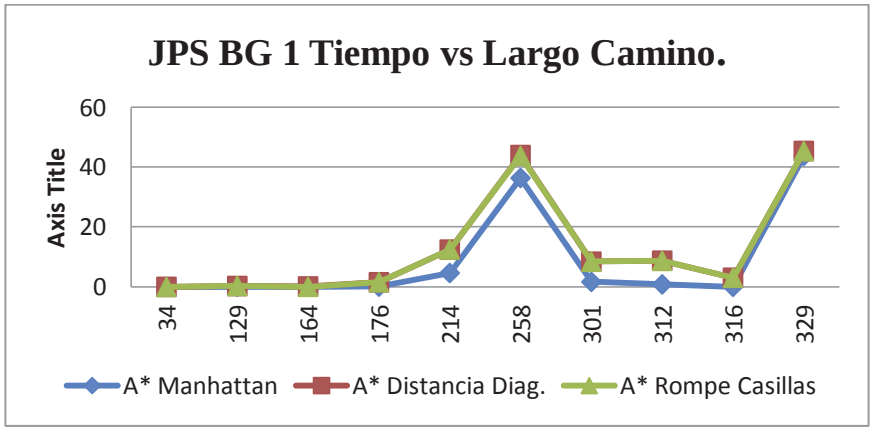

Ilustración 7.2 Gráfico JPS BG 1 tiempo vs Largo Camino.

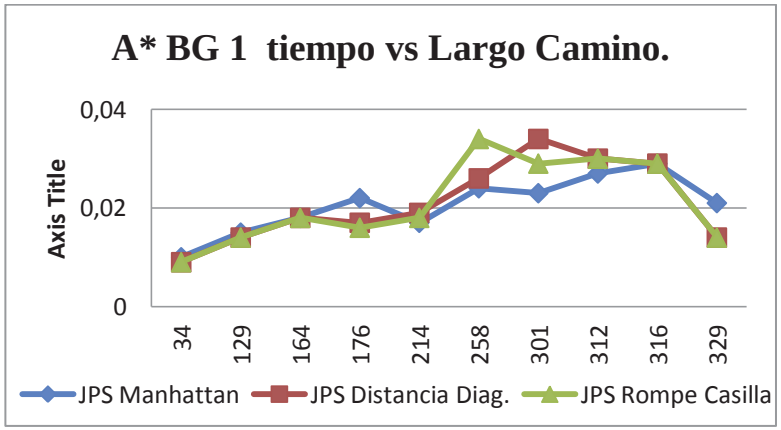

Ilustración 7.3 Gráfico A* BG 1 tiempo vs Largo Camino.

Tabla 7.2 Resultados horizontales A* distancia diagonal.

Largo	Mazes ancho 8		Mazes ancho 4		Rooms 8		Rooms 16		BG 1		BG 2	
Camino	*Nodos*	*Tmp(seg)*	*Nodos*	*Tmp(seg)*	*Nodos*	*Tmp(seg)*	*Nodos*	*Tmp(seg)*	*Nodos*	*Tmp(seg)*	*Nodos*	*Tmp(seg)*
36-46	442	0,01601	157	0,003002	203	0,005003	132	0,003003	27	0,001	41	0,001
496-503	16046	32,0344	5984	3,76852	38919	229,847	35652	186,052	36346	206,755	9535	12,5223

Tabla 7.3 Resultados horizontales JPS distancia diagonal.

Largo	Mazes ancho 8		Mazes ancho 4		Rooms 8		Rooms 16		BG 1		BG 2	
Camino	*Nodos*	*Tmp(seg)*	*Nodos*	*Tmp(seg)*	*Nodos*	*Tmp(seg)*	*Nodos*	*Tmp(seg)*	*Nodos*	*Tmp(seg)*	*Nodos*	*Tmp(seg)*
36-46	16	0,13009	13	0,016011	25	0,026017	9	0,012009	5	0,009006	5	0,010006
496-503	172	0,082055	189	0,088059	3827	0,575385	856	0,07705	160	0,044028	41	0,047032

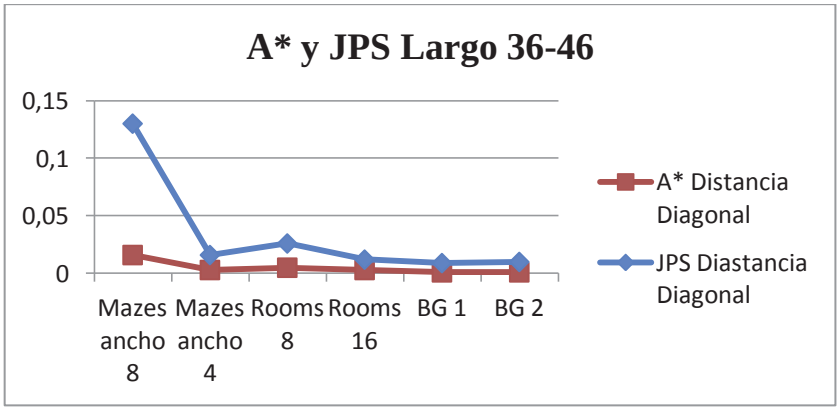

Ilustración 7.4 Gráfico Tiempo vs mapa de camino 36-46 horizontal.

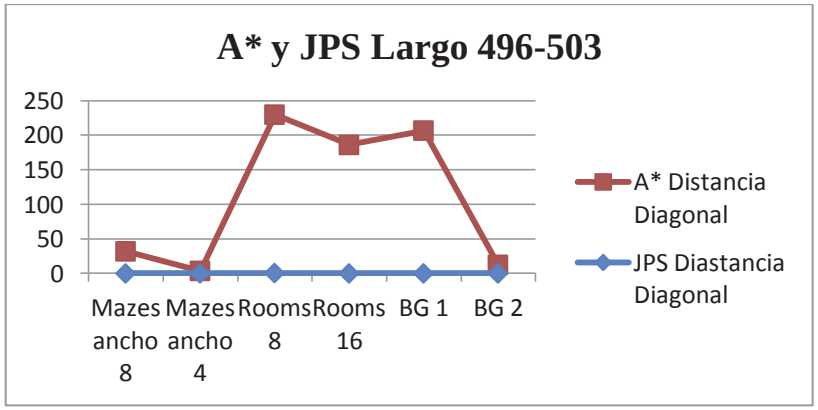

Ilustración 7.5 Gráfico Tiempo vs mapa de camino 496-503 horizontal.

En la Tabla 7.2, se presentan los resultados de A* con la heurística de distancia diagonal para todos los mapas, por cada mapa se muestran los nodos expandidos y el tiempo en segundos que demoró el algoritmo en ese mapa. La Tabla 7.2 tiene dos filas de datos; la primera 36-46, quiere decir que los largos del camino óptimo en ese escenario varía de 36 nodos a 46 nodos entre los mapas. Este largo representa el escenario con el camino más corto evaluado en cada mapa. Por ejemplo en la Tabla 7.1, del primer mapa de BG, el camino de menor largo evaluado es 34 nodos (correspondiente al mínimo de la Tabla 7.2 y 7.3), y los mínimos de los otros mapas varían hasta 46. En la segunda fila de la Tabla 7.2, los largos de los caminos de los escenarios en cada mapa varían entre 496 y 503 nodos, este rango, aunque no es el máximo en todos los mapas, lo es o se acerca en la mayoría de ellos. Aunque en la Tabla 7.1 no se puede observar los largos de caminos dentro del rango entre 496 y 503, estos datos están disponibles en la versión extendida en el Anexo de este documento. La Tabla 7.3 muestra lo mismo que la Tabla 7.2, pero con el algoritmo JPS con la heurística distancia diagonal.

En la Ilustración 7.4 se grafican los resultados con escenarios de camino óptimo de largo entre 36 y 46 nodos. Esto para A* y JPS con la heurística distancia diagonal. En el eje horizontal se muestran los mapas probados y en el eje vertical el tiempo en segundos. La Ilustración 7.5 muestra lo mismo que la 7.4, pero con caminos óptimos entre los largos de 496 y 503 nodos.

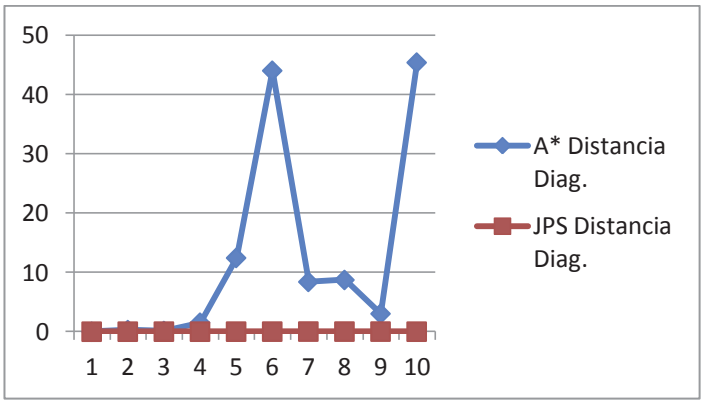

Ilustración 7.6 Gráfico tiempo vs escenario mapa BG 1 distancia diagonal.

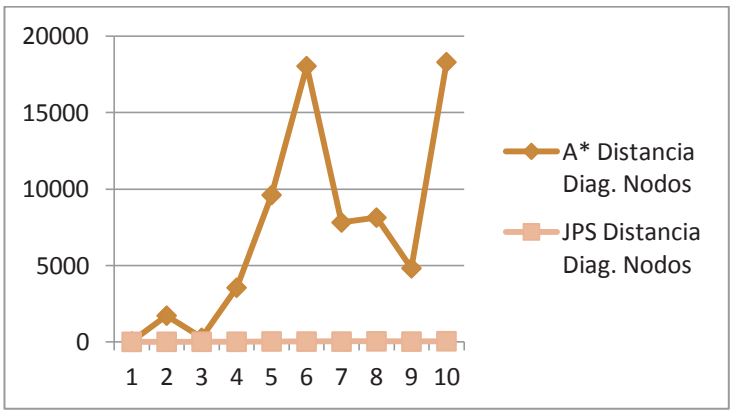

Ilustración 7.7 Gráfico nodos expandidos vs escenario mapa BG 1 distancia diagonal.

En la Ilustración 7.6, se muestra un gráfico que representa la evolución del tiempo en los distintos escenarios del mapa BG 1 para la heurística distancia diagonal. En el eje vertical se mide el tiempo en segundos y en el eje horizontal se enumera cada escenario.

Nótese que los escenarios están ordenados de menor a mayor según el largo en nodos del camino óptimo en cada uno, siendo 1 el escenario con el camino óptimo de menor largo y 10 el con mayor. La Ilustración 7.7, es similar a la Ilustración 7.6, pero la diferencia es que en el eje vertical se miden los nodos expandidos y no el tiempo. Para cada heurística y para cada mapa existen gráficos similares a los de las Ilustraciones 7.6 y 7.7.

7.4.2. Análisis de Resultados

En esta sección se analizarán los resultados obtenidos, los cuales están disponibles en su totalidad en el Anexo.

En todos los mapas y escenarios se puede apreciar un resultado similar comparando JPS y A*. Generalmente se observa que ambos algoritmos tienen un desempeño similar, de menos de un segundo, para caminos de largo menor a 150 nodos. Pero cuando se va amentando el largo del camino, el tiempo que demora JPS se mantiene alrededor del segundo, mientras que A* aumenta considerablemente. Por ejemplo, en el escenario 10 del mapa uno de BG, con un camino óptimo de largo 329, utilizando la heurística distancia diagonal, A* demoró 45.4184 segundos, mientras que JPS tardó sólo 0,014008 segundos, lo que quiere decir que JPS tuvo un desempeño 3242.319 más rápido que A*, mejora sustantiva a la hora de elegir un algoritmo para su implementación en un videojuego.

En el caso de los nodos expandidos se observa un comportamiento similar. La diferencia está en que para todos los casos se da que A* expande más nodos que JPS. Pero la variación entre A* y JPS, aumenta significativamente conforme a que el largo del camino a encontrar es mayor. Por ejemplo, en el primer escenario del primer mapa de BG, con un camino a encontrar de largo 34, con la heurística distancia diagonal, A* expandió 27 nodos, mientras que JPS sólo 5. Para un camino de largo 214 (escenario 5), con la misma heurística anterior, A* expandió 9607 nodos mientras que JPS sólo 28. Con la misma heurística, pero esta vez en un camino de largo 329 (escenario 10), A* expandió 18304 nodos y JPS 42. En el primer caso JPS es 5.4 veces más rápido que A*, en el segundo esa proporción es de 343.18 y en el último caso JPS expande 435.81 veces nodos menos que A*. Diferencias significativas a la hora de tener en cuenta la memoria utilizada por los algoritmos. Aunque hay que considerar que JPS utiliza un algoritmo recursivo que ocupa más memoria que A*.

La diferencia entre las heurísticas utilizadas es variable y no sobresaliente. Aunque ésta es visible en los gráficos mostrados en las Ilustraciones 7.2 y 7.3. En la totalidad de los gráficos para cada mapa, se puede observar que para A* la heurística de Manhattan pareció funcionar mejor que las otras dos, aunque hay que tener en cuenta que ésta no es admisible, por lo que algunos casos el camino encontrado no era el óptimo, pero se asimilaba bastante. En el caso de JPS, se puede observar en los gráficos mencionados anteriormente que la heurística que en general tuvo mejores resultados fue la Rompe Casillas, pero en el caso de este algoritmo los resultados fueron mixtos, ya que en algunos casos Manhattan fue mejor, mientras que en otros lo fue la distancia diagonal.

La diferencia entre los tipos de mapas y sus configuraciones es poco notoria. Aunque en el caso de JPS y A*, en los mapas tipo Rooms, consistentes en una seria de cuartos conectados, mientras más pequeño es ese cuarto, más tarda el algoritmo, debido a que aumenta la cantidad de giros que tiene que tomar el camino, lo que hace más complejo encontrar el objetivo. Lo mismo en el caso de los mapas tipo Mazes o laberintos, pero en este escenario, las diferencias fueron menores para los distintos anchos del pasillo de

laberinto. En el caso de los mapas de BG, JPS tuvo un leve, aún así consistente, mejor desempeño que en los otros tipos de mapa, esto debido a la cantidad de espacios abiertos que existían, situación para la que JPS está diseñado a mejorar ignorando las simetrías. A* por el otro lado busca todos los caminos posibles por lo que en espacios abiertos muy grandes tiende a disminuir su efectividad.

Cabe destacar que existen en los resultados algunos casos en que no necesariamente el tiempo o la cantidad de nodos expandidos es proporcional al largo del camino objetivo. Esto se debe a que cada camino puede resultar mucho más complejo de encontrar por las dificultades que se encuentren por la estructura del mapa que el camino sigue. Teniendo que hacer muchos más cálculos para llegar a un nodo que puede no estar tan lejos.

En general se pudo observar un desempeño significativamente mejor de JPS, especialmente mientras más largo es el camino, todos los datos apuntaron a eso. Las distintas heurísticas no afectaron sustancialmente a este algoritmo, y en gran parte de los mapas se obtuvo un desempeño de alrededor del segundo.

8. Conclusiones

Pathfinding es un área de investigación que está en constante desarrollo, aunque la bibliografía que se puede encontrar no es muy extensa en cuando a definiciones generales. Existe una gran variedad de trabajos y desarrollos de distintos métodos y algoritmos que apuntan a mejorar lo que se utiliza hasta ahora en la navegación de videojuegos. Entre esos algoritmos está JPS.

Se cumplió con el primer objetivo específico. Se explicaron los conceptos básicos de Pathfinding, los enfoques que se pueden tomar, los algoritmos más importantes entre otros aspectos de la teoría. Se incluyó en este marco teórico una explicación del principal algoritmo a utilizar para comparado, JPS, creado por David Harabor y Alban Grastien, que básicamente, se utiliza en conjunto con A* para poder omitir la cantidad de nodos a expandir eliminando simetrías alrededor del camino que se está recorriendo.

Con respecto al segundo objetivo, se identificaron los aspectos a evaluar de los algoritmos JPS y A*, basándose en el trabajo de Daniel Harabor y Alban Grastien. El tiempo y los recursos parecen ser algo esencial en la evaluación de los algoritmos, puesto que ambos son recursos limitados a la hora de desarrollar videojuegos, y si estos parámetros tienen un buen desempeño, probablemente el algoritmo también lo tendrá.

En relación a los últimos objetivos específicos, estos se cumplieron desde el punto de vista que se logró realizar la comparación entre A* y JPS exitosamente, mezclándolos con las heurísticas de Manhattan, Distancia Diagonal y Rompe Casillas. De ello se lograron obtener más de 60 resultados. En un total de seis mapas, con diez escenarios cada uno, probando 6 combinaciones diferentes de los algoritmos mencionados anteriormente.

De acuerdo a los resultados presentados en la sección anterior y en la tabla general de resultados anexa, JPS presenta una mejora bastante significativa en el rendimiento. JPS, en todos los casos probados raramente supera el segundo de ejecución, y la cantidad de nodos expandidos es considerablemente menor a la de A*, con todas las combinaciones de las heurísticas.

En los casos más extremos, con los caminos más largos, la diferencia entre los dos algoritmos en el tiempo de ejecución llega a ser de aproximadamente 11.000 veces menor para JPS. Esto tipo de ventaja se produjo en todos los tipos de mapa.

En el caso de las heurísticas, en JPS existió muy pequeña diferencia entre cada una. Pero para el caso de A*, cuando se utilizaba la heurística de Manhattan, el tiempo que se tomaba la ejecución fue de alrededor de un tercio menor a las otras dos heurísticas, las cuales tenían siempre resultados casi iguales.

Dadas las pruebas realizadas, se puede concluir que JPS tiene una ventaja tremenda, frente a su competidor A*, el tiempo que demora en la búsqueda es significativamente menor, este valor es directamente proporcional al largo del camino que se quiere encontrar, si el camino es muy pequeño, los dos algoritmos demoran casi lo mismo pero si el largo del camino aumenta, la diferencia crece. No se recomienda cambiar la heurística de JPS puesto que, no se notaron grandes diferencias en los resultados de las pruebas comparativas entre ellas.

9. Referencias

[1] Harabor D. 2012, Fast Pathfinding via Symmetry Breaking. AIgamedev, Disponible vía web en http://aigamedev.com/open/tutorial/symmetry-in-pathfinding/. (Acceso en 30 /8/ 2013).

[2] Harabor, D. and Grastien, A.. Online Graph Pruning for Pathfinding On Grid Maps. 2011 . Proceeding of the national conference on artificial inteligence". 1114-1119

[3] Jeremy Christman, "General Pathfinding: Tables and Navigation", disponible vía web http://www.cse.lehigh.edu/~munoz/CSE348/classes/ChristmanPathfinding.pptx, (Acceso en 30/8/2013).

[4] Markus Jonsson, "An optimal pathfinder for vehicles in real-world digital terrain maps", The Royal Institute of Science, School of Engineering Physics, Stockholm, Sweden This paper is presented by the Department of Numerical Analysis and Computing Science

[5] Lester, P. (2005). A* pathfinding for beginners. online]. GameDev WebSite. http://www. gamedev. net/reference/articles/article2003. asp (Acesso en 30/08/2013).

[6] Stout, B. (1996). Smart moves: Intelligent pathfinding. Game developer magazine, 10, 28-35

[7] Sturtevant, N. R. (2012). Benchmarks for grid-based pathfinding. Computational Intelligence and AI in Games, IEEE Transactions on, 4(2), 144-148.

[8] Harabor, D., and Botea, A. 2010. Breaking path symmetries in 4-connected grid maps. In AIIDE, 33–38.

[9] Bj¨ornsson, Y., and Halld´orsson, K. 2006. Improved heuristics for optimal path-finding on game maps. In AIIDE, 9–14.

[10] Pochter, N.; Zohar, A.; Rosenschein, J. S.; and Felner, A. 2010. Search space reduction using swamp hierarchies. In AAAI.

[11] Sturtevant, N. R.; Bulitko, V.; and Bj¨ornsson, Y. 2010. On learning in agent-centered search. In AAMAS, 333–340.

[12] Sun, X.; Yeoh, W.; Chen, P.-A.; and Koenig, S. 2009. Simple optimization techniques for a*-based search. In AAMAS (2), 931–936.

[13] Botea, A.; M¨uller, M.; and Schaeffer, J. 2004. Near optimal hierarchical path-finding. J. Game Dev. 1(1):7–28

Anexo
Tablas y gráficos