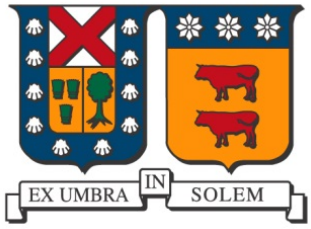
UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA
DEPARTAMENTO DE INFORMÁTICA
VALPARAÍSO - CHILE



“TÍTULO DE LA MEMORIA”

RICARDO GAMBOA ARIAS

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL EN INFORMÁTICA

Profesor Guía: Elizabeth Montero
Profesor Correferente: Nombre del correferente

Diciembre - 2018

DEDICATORIA

AGRADECIMIENTOS

RESUMEN

Resumen—

Palabras Clave—

ABSTRACT

***Abstract*—**

***Keywords*—**

GLOSARIO

Agente: Entidad que tiene que moverse por el tablero.
Heuristica: Método diseñado para resolver un problema mas rápido cuando los métodos clásicos tienen algún problema en el caso especifico. Normalmente sacrifica optimalidad, completitud, precision o validez para lograrlo.
Tablero: Representación computacional en forma de grafo que se en la representación del problema y la búsqueda de la solución.

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

INTRODUCCIÓN

CAPÍTULO 1
DEFINICIÓN DEL PROBLEMA

1.1. CONTEXTO

Durante años hemos necesitado encontrar el camino para llegar desde un lugar a otro, la velocidad con la que se obtiene esta información puede ser de gran ayuda en casos como encontrar el camino a un lugar con un GPS o encontrar un camino optimizado para un robot. Problema del camino más corto es un problema de la teoría de grafos que quiere encontrar un camino con una característica menor, en nuestro caso es tiempo de viaje. En casos donde el algoritmo no es lo suficientemente rápido se pueden usan atajos para obtener soluciones menos óptimas pero que son mas rápidas de obtener.

1.2. PROBLEMA

Pathfinding, es una rama del problema del camino mas corto, se basa en buscar el camino entre dos puntos en un espacio representado computacionalmente(Ver figura 1). Se busca una lista de posiciones por las cuales tenga que pasar el agente[1] de tal modo que se simplifique el movimiento a saltos entre posiciones como en un tablero o movimientos simples como lineas rectas que pueda seguir el agente.

1.3. DIFERENCIAS

1.3.1. TABLERO

Dependiendo del tipo de tablero en el que tiene encontrar un camino se encuentran distintos problemas, estos se separan en:

- Estático: El único movimiento en el tablero es el agente que esta buscando el camino, este es el caso mas simple porque no se tiene que ajustar el *pathfinding* durante el movimiento.
- Dinámico: Existen cambios en el tablero provocados por otras entidades o otros efectos. En este caso el problema es que el camino encontrado con anterioridad puede parar de ser funcional de un momento a otro y buscar el camino completamente después de cada movimiento es demasiado costoso.

[1] Entidad que tiene que moverse por el tablero.

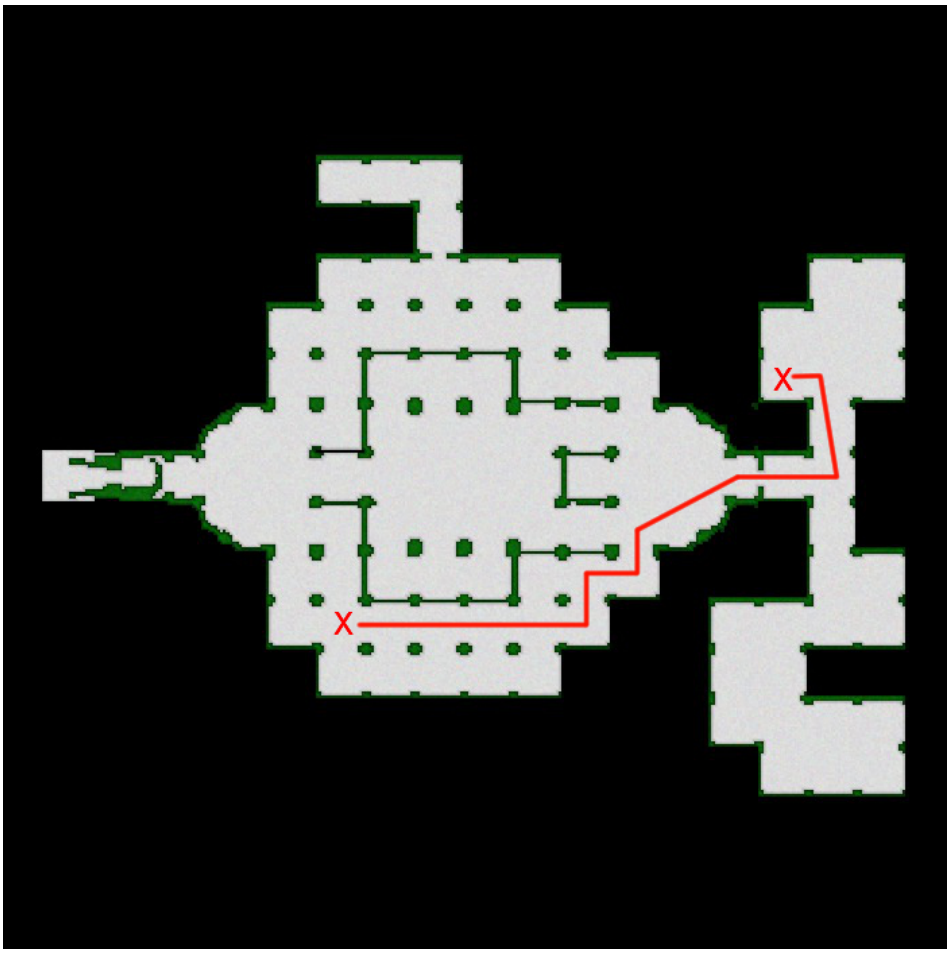


Figura 1: Ejemplo de *pathfinding*
Fuente: *Dragon Age: Origins*[Bioware(2009)].

También existen problemas donde se puede encontrar el camino para distintos tipos de agentes que se relacionan diferente con los distintos tipos de terreno, por ejemplo, agentes que puedan desplazarse sobre el agua no tienen que preocuparse de esquivarla.

1.3.2. VISIBILIDAD

Otro punto a considerar es el conocimiento previo del tablero y la visibilidad que se tiene de este, si el tablero es conocido, se pueden usar algoritmos que permiten optimizar el *pathfinding* con procesamiento previo, estos se separan en:

- Conocimiento total del tablero: El agente conoce el tablero y tiene visión de este en cada momento.
- Conocimiento del tablero pero visibilidad reducida: El agente tiene alguna especie de

cámara o visión periférica por lo cual conoce los cambios en el tablero que están cerca de el.

- Conocimiento reducido: No hay conocimiento del tablero y este se genera con la visión del agente mientras se esta moviendo.

1.3.3. MOVIMIENTO

El movimiento del agente puede ser variable dependiendo del terreno por lo cual puede variar el tiempo o el coste[2]. También el movimiento puede separarse en:

- Discreto: En juegos con tablero se usa este tipo de movimiento porque las posiciones posibles están dadas por enteros, como en el ajedrez.
- Continuo: En el trabajo de robótica y en juegos en tiempo real se usan movimientos continuos, para encontrar el camino el mundo continuo se reduce a un sistema discreto y luego con ese sistema se puede crear el movimiento continuo.

1.3.4. TAMAÑO

Cuando tienes que mover distintos agentes dentro de un tablero, pueden existir agentes que ocupan mas espacio que otros por lo cual sus caminos tienen que poder contenerlos. Casos como unidades que ocupan 1 celda del tablero y otras que ocupan 4 puedes ser mas complicados de tratar en un solo grafo.

1.4. GRAFOS

La representación que se usara en la memoria son grafos, un conjunto de nodos unidos por arcos que permiten relacionar a estos. Estos nodos son las posibles posiciones en el espacio y los arcos representan las formas de movimiento entre ellos. Los arcos llevan un valor agregado que nos permite saber cuanto cuesta el movimiento desde un punto a otro, en un tablero de cuadrados existen 4 u 8 formas de movimiento dependiendo de como sea la decisión de diseño(Ver figura 2). Los movimientos cardinales tienen un coste 1 y los diagonales tienen un coste $\sqrt{2}$. Se explicara mas a fondo en el Marco Conceptual.

[2]En juegos con turnos, los agentes tienen una cantidad de puntos para sus movimientos que se reducen dependiendo del terreno por el que caminan

Figura 2: Ejemplo de movimiento entre Nodo central, sus cuatro direcciones cardinales y las diagonales en colores diferentes.

Fuente: Propio.

1.5. ENFOQUE

En esta memoria nos centraremos en el problema de *pathfinding* en juegos en tiempo real, por lo cual los dos problemas principales que se enfrentan son:

- Tiempo: En los juegos normalmente se le permite un tiempo a la búsqueda de caminos de algunos mili segundos para no entorpecer el funcionamiento del resto de las funciones del juego por lo cual la velocidad de el algoritmo se vuelve imperativa para que la calidad del juego no sea afectada.
- Calidad: Dentro del tiempo que es posible ocupar se tiene que entregar la mejor solución posible de tal modo que los agentes no tomen caminos que podrían ser denominados de poco humano para los jugadores.

Además nos estamos enfocado en mapas auto generados[3] no podemos usar pre-procesamiento para obtener un aumento de velocidad en el *pathfinding*, por lo cual nos enfrentamos a un problema que no permite ocupar pre-procesamiento directamente.

[3]Creados en el momento que se inicia la partida

1.6. OBJETIVO DE LA SOLUCIÓN

1.6.1. OBJETIVOS GENERALES

1.6.2. OBJETIVOS ESPECÍFICOS

1.
2.
3.

CAPÍTULO 2
MARCO CONCEPTUAL

2.1. PATHFINDING

Pathfinding es la creación de un plan de acción para cursar la ruta mas corta entre dos puntos, usando computación. El exponente inicial para la solución de este problema es el algoritmo de *Dijkstra's* que encuentra la ruta optima en un grafo con costos en sus arcos desde un nodo a todos los otros.

En el trabajo con videojuegos se acostumbra a usar estos *benchmarks* estandarizados [Sturtevant(2012)], estos han sido extraídos de juegos o creados para poner a prueba las distintas formas de encontrar la solución.

Es un punto importante del trabajo en videojuegos por la demanda de eficiencia, el trabajo desarrollado para el *pathfinding* regularmente tiene un tiempo alocado que no permite buscar indefinidamente los caminos por lo cual es necesario tener medidas que han ido cambiando con los años para obtener resultados en el tiempo esperado, muchas veces esto genera agentes que se mueven de maneras innaturales pero entregarle mas tiempo a esto no es posible porque afectaría el desempeño del juego.

Normalmente se tienen que mover varios agentes en tiempo real y para esto existen distintos algoritmos. Antes de entrar a hablar de los distintos algoritmos, veremos un poco de la historia y teoría ocupada.

2.2. TEORIA DE GRAFOS

2.2.1. DEFINICION DE GRAFO

Un grafo G consiste en un grupo no vació, finito de nodos, $V(G)$, y un grupo finito de arcos de pares de nodos, $E(G)$. Un vértice {v,w}, uno los vértices v y w. Se abrevia vw.

Los grafos que consideraremos serán mas simples porque no consideran mas de un arco entre los mismos dos nodos y no tienen arcos de un nodo a si mismo

La Figura 3 muestra el grafo G, donde $V(G) = \{u, v, w, z\}$ y $E(G) = \{uv, uw, vw, wz\}$

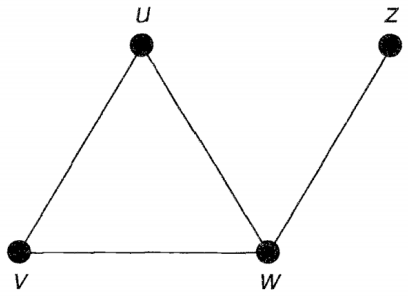


Figura 3: Ejemplo de Grafo
Fuente: *Introduction to Graph Theory* [Wilson(1970)].

2.2.2. CAMINO

Un camino en un grafo G es una secuencia finita de de arcos denotado por $v_o \rightarrow v_1 \rightarrow v_2 \rightarrow v_3 \rightarrow v_4 \rightarrow v_5 ...$

La Figura 4 muestra el grafo G, donde $V(G) = \{x, v, w, y, z\}$ y $E(G) = \{vx, vw, vy, wx, wy, yx, xz, yz, zz\}$. Se pueden generar distintos caminos entre v y z, de distintos largos, por ejemplo:

- $v \rightarrow w \rightarrow x \rightarrow z$ con largo 3.
- $v \rightarrow y \rightarrow z$ con largo 2.

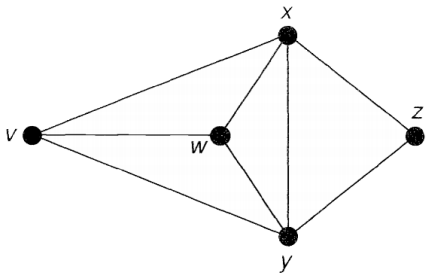


Figura 4: Ejemplo de Camino en un Grafo
Fuente: Introduction to Graph Theory [Wilson(1970)].

2.2.3. COSTO

Los arcos de los grafos que ocuparemos tendrán asociados un valor que llamaremos costo, donde el arco entre v y w se representa $\{v, w, X\}$ con $X \in \mathcal{R}^+$

La Figura 5 muestra el mismo grafo anterior con valores en los arcos y compararemos los caminos mostrados antes, ahora con el costo total asociado.

- $v \to w \to x \to z$ con costo 3.
- $v \to y \to z$ con costo 5.

En estos caminos podemos observar que el camino con menos arcos, no es mejor y esto es interesante cuando empezamos a trabajar en *pathfinding*.

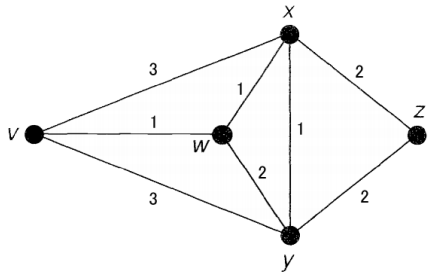


Figura 5: Ejemplo de Camino en un Grafo con Costo en sus arcos
Fuente: *Introduction to Graph Theory* [Wilson(1970)], con edición propia.

2.3. ALGORITMOS

Los métodos de *pathfinding*, comúnmente necesitan tres cosas. Un grafo para representar el mapa, un algoritmo de búsqueda y una heuristica para guiar la búsqueda.[Botea et al.(2013)Botea, Bouzy, Buro, Bauckhage, and Nau]

Una forma muy común de representar los mapas, es en casillas como el ajedrez, que se consideran de forma atómica[4]. Esta representación como grafo te permite aplicar el algoritmo de búsqueda.

Un primer acercamiento a un algoritmo para buscar un camino optimo es probar con todos los posibles caminos, esto se podría lograr con una búsqueda en profundidad. Pero se pueden eliminar caminos imposibles con anterioridad, o buscar en caminos mas problemas antes, por lo cual se usan heuristicas para acelerar la búsqueda.

Ahora entramos a hablar mas en detalle de los distintos algoritmos y heuristicas comúnmente ocupados para este problema.

[4]El movimiento hacia dentro y fuera de la casilla se considera una acción por la búsqueda, no hay movimientos mas pequeños.

2.3.1. DIJKSTRA'S

El algoritmo de *Dijkstra's* [Dijkstra(1959)] encuentra el camino mas corto entre dos nodos. Empieza con el nodo inicial y sus vecinos en un conjunto de opciones, en cada iteraccion selecciona el nodo con la menor distancia al inicial del conjunto de opciones, este queda marcado como visitado y sus vecinos que no han sido visitados entran al conjunto de opciones. Se repite este proceso hasta encontrar el nodo final o que este vacio el conjunto de opciones, en tal caso no existe un camino entre los nodos.

La primera version entregaba la distancia del camino, pero posteriores iteraciones han entregado el camino a todos los otros nodos desde el inicial o el camino a un nodo especifico entregado.

```
function Dijkstra(Graph, source):
    for each vertex v in Graph:          // Inicializar
        dist[v] := infinity              //Dist. Desconocida
        previous[v] := undefined         // Nodo desconocido
    dist[source] := 0                    // Nodo inicial
    Q := the set of all nodes in Graph   // Nodos a recorrer
    while Q is not empty:                // Main loop
        u := node in Q with smallest dist[ ]
        remove u from Q
        for each neighbor v of u:        // Vecinos faltantes
            alt := dist[u] + dist_between(u, v)
            if alt < dist[v]             // Usar Dist. menor
                dist[v] := alt
                previous[v] := u
    return previous[ ]
```

Figura 6: Pseudo Codigo de algoritmo de *Dijstra's*
Fuente: GITTA [Thaddeus A(2017)].

La Figura 7 se muestra un grafo al que se le aplica este algoritmo para encontrar todos los caminos mínimos a el resto de sus nodos.

Cabe destacar que el algoritmo de *Dijstra's* no funciona si existen arcos con costos negativos, pero esto no afecto a la búsqueda de caminos porque no tendría sentido que el agente se demorara un tiempo negativo en una acción.

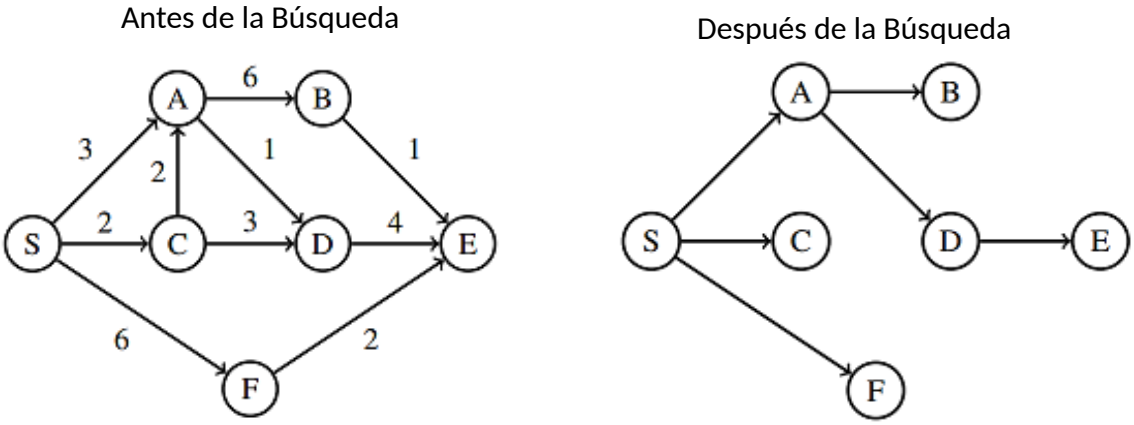


Figura 7: Ejemplo de la aplicacion del algoritmo de Djkstra's

Fuente: *Dijkstra's Shortest Path Algorithm* [Thaddeus A(2017)].

2.3.2. A*

A* es una variación de *Dijkstra's*, usa una heuristica que permite moverse a los nodos que están en dirección al objetivo. Logra esto sumando el peso del arco con una suposición de la distancia del nodo siguiente al nodo objetivo.

Esta suposición de la distancia es la heuristica, esta permite que no se revisen caminos posiblemente mas largos cuando existe la posibilidad de que exista un camino directo.

El Seudo Codigo de A* es similar a Dijkstra's, el cambio a destacar es la diferencia que genera la heuristica(Como se ve en la figura 8), en el ejemplo posterior estamos usando:

$$
\begin{aligned}
heuristic(v, f) &= |v.x - f.x| + |v.y - f.y| \\
v &= vertice\ actual \\
f &= vertice\ final \\
t.x &= posicion\ x\ en\ el\ tablero\ de\ vertice\ t \\
t.y &= posicion\ y\ en\ el\ tablero\ de\ vertice\ t
\end{aligned}
$$

Se ve en la figura 9, el numero sobre las celdas muestra el valor con el que se toma la decisión de movimiento, como A* intenta usar el camino mas directo va directo a la muralla pero aun con esto explora menos que Dijkstra's que explora todo el tablero.

Aquí se ve como el uso de heuristicas puede mejorar la magnitud de la búsqueda necesaria para encontrar el camino optimo.

```
function A*(Graph, source, heuristic):
    for each vertex v in Graph:           // Inicializar
        dist[v] := infinity               //Dist. Desconocida
        previous[v] := undefined          // Nodo desconocido
    dist[source] := 0                     // Nodo inicial
    Q := the set of all nodes in Graph    // Nodos a recorrer
    while Q is not empty:                 // Main loop
        u := node in Q with smallest dist[ ]
        remove u from Q
        for each neighbor v of u:         // Vecinos faltantes
            alt := dist[u] + dist_between(u, v) + heuristic(v)
            if alt < dist[v]              // Usar Dist. menor
                dist[v] := alt
                previous[v] := u
    return previous[ ]
```

Figura 8: Pseudo Codigo de algoritmo *A**
Fuente: *Introduction to A** [Red(2016)].

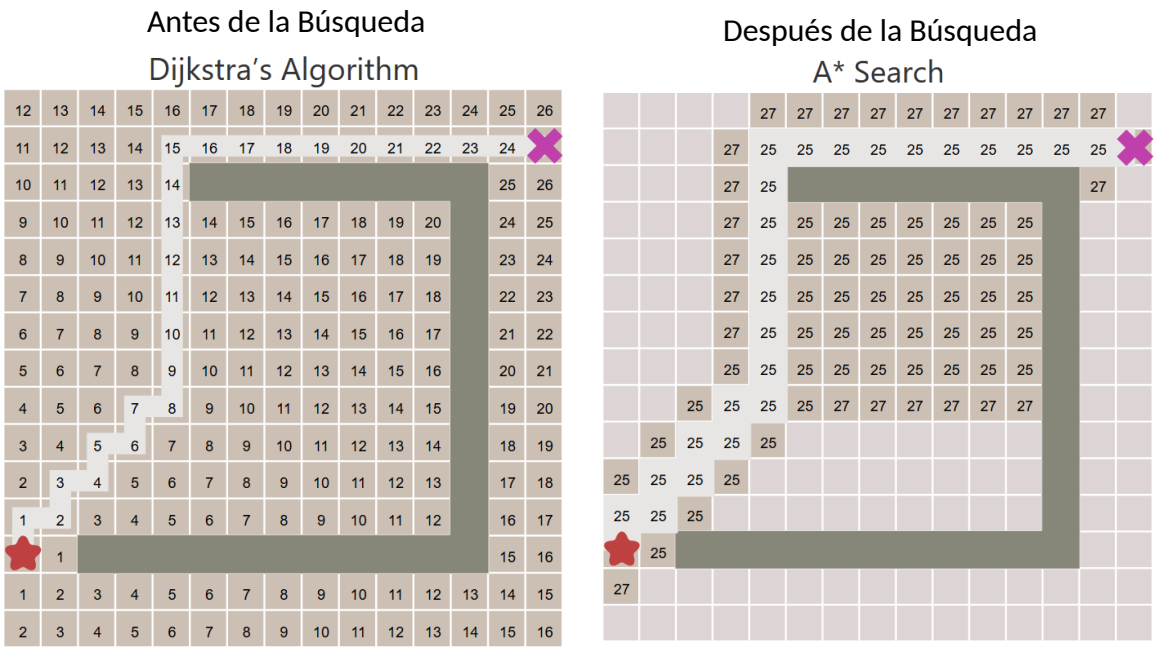


Figura 9: Comparacion entre Dijkstra's y A*
Fuente: *Introduction to A**[Red(2016)].

2.3.3. HEUTISTICAS

Como vimos en A* usar heuristicas puede acelerar la búsqueda del camino, algunos ejemplos usados en la literatura son los siguientes.

1. *Manhattan*: Explicada en la sección de A*.
2. *Octile*: Para los mapas que permiten movimiento diagonal, se usa con A* en estos mapas.

$$Octile = \sqrt{2} * m + (M - m)$$
$$m = min(\Delta x, \Delta y)$$
$$M = max(\Delta x, \Delta y)$$

3. *Differential*: [Goldberg and Harrelson(2005)] Se hace un paso de procesamiento previo calculando la distancia real desde todos los puntos a unas *landmarks*[5]. Con esta información puedes calcular la heuristica para tener una estimación de la distancia entre el nodo n_1 y n_2 con todas las *landmarks* L.

$$Differential = max(|d(n_1, l) - d(n_2, l)|),\ l \in L$$
$$d(n_x, l) = distancia\ entre\ n_x\ y\ l$$

   Esto aproxima con la máxima distancia para evitar casos como que un *landmark* este en medio de ambos nodos.

4. *Dead-end*: [Björnsson and Halldórsson(2006)] Identifica areas que no tienen la solucion y les da un valor infinito.
5. *Gateway, canonical y portal*: [Björnsson and Halldórsson(2006), Sturtevant et al.(2009)Sturtevant, Felner, Barer, Schaeffer, and Burch, Goldenberg et al.(2010)Goldenberg, Felner, Sturtevant, and Schaeffer] Tres heuristicas que intentan separar el mapa en los cuellos de botella y precomputar movimientos entre los sectores creador

2.3.4. ABSTRACCIONES DE MAPA

Como vimos, lo normal es representar el mapa en un tablero, existen algoritmos que toman este tablero y hacen abstracciones posteriores para encontrar soluciones de mayor nivel rápidamente, esto permite trabajar en grafos mas pequeños y después cortar el grafo original

[5]Puntos ïmportante"dentro del mapa, pueden ser cuellos de botella, posiciones predefinidas u otros

para buscar una solución. También permite tener una dirección aproximada mas rápidamente, lo cual puede permitir que el agente empiece su movimiento sin tener la solución completa.

Se tiene que tener claro que al encontrar una solución a un nivel mas alto tienes que tener seguridad que es una solución inferior, porque hacer *backtracking* puede ser muy costoso.

1. *HPA**: [Botea et al.(2004)Botea, Müller, and Schaeffer] Separa el mapa en sectores cuadrados, las conexiones entre sectores se identifican y son agregadas como nodos. Luego que se arman todas las capas que se decidieran, se puede buscar conectando el nodo origen y final a sus sectores y hacer el *pathfinding* desde el nivel mas alto hacia abajo.

2. *PRA**: [Sturtevant and Buro(2005)] Recursivamente separa el mapa en bloques de $2x2$, hasta que el mapa esta representado por un solo nodo. Con el origen y el final conectados a una capa de abstracción se inicia el *pathfinding* y se va bajando por la pirámide de abstracciones.

3. *HAA**: [Harabor and Botea(2008)] Basado en HPA*, tiene dos cambios al anteriores, permite el uso de distintos tipos de terreno, como agua y bosque, que afectan a que unidades pueden atravesarlo. Ademas permite usar unidades de tamaño diferente, las unidades comunmente vistas son de una celda, en este caso se puede encontrar el camino para unidades que usan 4, 9 o mas, celdas.

4. *Block A**: [Yap et al.(2011)Yap, Burch, Holte, and Schaeffer] Usando un tamaño predefinido de $m * n$, se precomputan todas las posibles combinaciones de terreno que pueden existir en ese tamaño de bloques, en ves de expandir por nodo, se expande cada bloque.

CAPÍTULO 3
PROPUESTA DE SOLUCIÓN

CAPÍTULO 4
VALIDACIÓN DE LA SOLUCIÓN

CAPÍTULO 5
CONCLUSIONES

ANEXOS

REFERENCIAS BIBLIOGRÁFICAS

[Bioware(2009)] Bioware. (2009) Maps of dragon age. Bioware. Revisado 2 de Diciembre del 2017. [Online]. Available: http://www.movingai.com/benchmarks/dao/

[Björnsson and Halldórsson(2006)] Y. Björnsson and K. Halldórsson, "Improved heuristics for optimal path-finding on game maps." *AIIDE*, vol. 6, pp. 9–14, 2006.

[Botea et al.(2004)Botea, Müller, and Schaeffer] A. Botea, M. Müller, and J. Schaeffer, "Near optimal hierarchical path-finding," *Journal of game development*, vol. 1, no. 1, pp. 7–28, 2004.

[Botea et al.(2013)Botea, Bouzy, Buro, Bauckhage, and Nau] A. Botea, B. Bouzy, M. Buro, C. Bauckhage, and D. Nau, "Pathfinding in games," in *Dagstuhl Follow-Ups*, vol. 6. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2013.

[Dijkstra(1959)] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[Goldberg and Harrelson(2005)] A. V. Goldberg and C. Harrelson, "Computing the shortest path: A search meets graph theory," in *Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics, 2005, pp. 156–165.

[Goldenberg et al.(2010)Goldenberg, Felner, Sturtevant, and Schaeffer] M. Goldenberg, A. Felner, N. Sturtevant, and J. Schaeffer, "Portal-based true-distance heuristics for path finding," in *Third Annual Symposium on Combinatorial Search*, 2010.

[Harabor and Botea(2008)] D. Harabor and A. Botea, "Hierarchical path planning for multi-size agents in heterogeneous environments," in *Computational Intelligence and Games, 2008. CIG'08. IEEE Symposium on*. IEEE, 2008, pp. 258–265.

[Red(2016)] B. G. Red. (2016) Introduction to a*. Revisado 2 de Diciembre del 2017. [Online]. Available: https://www.redblobgames.com/pathfinding/a-star/introduction.html

[Sturtevant(2012)] N. Sturtevant, "Benchmarks for grid-based pathfinding," *Transactions on Computational Intelligence and AI in Games*, vol. 4, no. 2, pp. 144 – 148, 2012. [Online]. Available: http://web.cs.du.edu/~sturtevant/papers/benchmarks.pdf

[Sturtevant and Buro(2005)] N. Sturtevant and M. Buro, "Partial pathfinding using map abstraction and refinement," in *AAAI*, vol. 5, 2005, pp. 1392–1397.

[Sturtevant et al.(2009)Sturtevant, Felner, Barer, Schaeffer, and Burch] N. R. Sturtevant, A. Felner, M. Barer, J. Schaeffer, and N. Burch, "Memory-based heuristics for explicit state spaces." in *IJCAI*, 2009, pp. 609–614.

[Thaddeus A(2017)] J. K. Thaddeus A, Hannah P. (2017) Dijkstra's shortest path algorithm. Revisado 2 de Diciembre del 2017. [Online]. Available: https://brilliant.org/wiki/dijkstras-short-path-finder/

[Wilson(1970)] R. J. Wilson, *An introduction to graph theory*. Pearson Education India, 1970.

[Yap et al.(2011)Yap, Burch, Holte, and Schaeffer] P. Yap, N. Burch, R. C. Holte, and J. Schaeffer, "Block a*: Database-driven search with applications in any-angle path-planning." in *AAAI*, 2011.