International Journal of Engineering Trends and Applications (IJETA) – Volume 2 Issue 1, Jan-Feb 2015

RESEARCH ARTICLE OPEN ACCESS

Direction Based Heuristic for Path finding In Video Games

Geethu Elizebeth Mathew[1], G.Malathy[2]
II- ME-CSE
K.S.R Institute for Engineering and Technology- Tiruchengode
TamilNadu – India

ABSTRACT
Path finding has been one of major research areas in video games for many years. It is a key problem that most of the video games are confronted with. Search algorithms such as the A* algorithm and the Dijkstra's algorithm representing such regular grid, visibility graphs also have significant impact on the performance. This paper reviews the current widely used solutions for path finding and proposes a new method which is expected to generate a higher quality path using less time and memory than other existing solutions. The deployment of the methodologies and techniques is described in detail. The significance of the proposed method in future video games is addressed and the conclusion is given at the end.
Key words: Path finding, A*, A* optimization, Computer game

I. INTRODUCTION

Path finding is the plotting, by a computer application, to find the shortest path between two points. It starts from a start node and reaches the goal node by repeatedly searching for the same, for finding a path between these points. Finding the optimal path is a complicated scenario. There are significant difference between the terms path and shortest path. In graph theory, the problem of finding a path between two vertices in a graph such that the sum of the weights this path's edges is minimized is called a shortest path problem. Two primary problems of path finding are to find a path between two nodes in a graph and to find the optimal shortest path [4].Path finding in the context of video games concerns the way in which an object finds a path around obstacles; the best explained context is real-time strategy games in which the player leads units around a play area containing obstacles, but the variations of this approaches are found in many of the games.

Path finding has grown in importance as games and their environments have become more complex. Many Artificial Intelligence based platforms and the tools are developed for providing solutions to path finding problem. Many of them uses basic path finding and provides readymade plugins for users to incorporate in their games. Real-time strategy games sometimes contain large areas of open terrain which is often relatively simple to navigate through, although it is common for more than one unit to travel simultaneously; this makes it necessary to employ different, and often more complex algorithms and methods to avoid traffic problems and bottlenecks at some points in terrain. In strategy games the map is normally divided into sub-worlds and there are practical methods of applying some algorithms in the smaller problems to apply it to larger sets.

II A* ALGORITHM

A* is a generic search algorithm that can be used to find solutions to many problems, pathfinding is just one of them. Many problem in the engineering are related to pathfinding problems. The look ahead effort in searching trees are found to provide improved results in pathfinding. The base of the A* algorithm arises from a view that the information from the problem domain can be incorporated in a formal mathematical manner to analyse the problem. The method shows that this approach will always try to find out a path by exploring minimum number of nodes to offer a minimum cost solution. A* is the most popular and widely used AI pathfinding algorithm proposed by Hart, Nilsson and Raphael in the year 1967. Due to its simplicity it guarantees, A* is almost always the search method of choice. This is because A* is guaranteed to find a shortest path on the graph. The problem with A* is that a shortest path on the graph is not equivalent to a shortest path in the continuous environment. Another issue related to A* is that, when the map size is significantly larger, A* algorithm cannot find a minimum path to goal state in limited amount of time. Also for large maps A* uses memory extensively. Even though many other methods are emerging, A* and its variations are widely used in path finding. A* uses this heuristic to improve on the behavior relative to Dijkstra's algorithm.

When the heuristic evaluates to zero, A* is equivalent to Dijkstra's algorithm. As the heuristic estimate increases and gets closer to the true distance, A* continues to find optimal paths, but runs faster. When the value of the heuristic is exactly the true distance, A* examines the fewest of the nodes .However it is generally impractical to write a heuristic function that always computes the true distance. As the value of the heuristic increases, A* examines fewer nodes but no longer guarantees an optimal path. In many applications this is acceptable and even desirable, in order to keep the algorithm running quickly. In order to expand less number of nodes as possible while searching for an optimal path, A* constantly make informed decision as possible about which node to expand next. Otherwise it is a wasting effort to use that particular method. If the algorithm constantly avoids nodes on

optimal path, then there is less chance that the algorithm will come up with a solution. Suppose, we have some evaluation function f(n) to be calculated for some node n. The nodes with f values are considered for exploring next. The node with the minimum f value is explored then we can define the A* search algorithm as follows[8].

Search Algorithm A*:

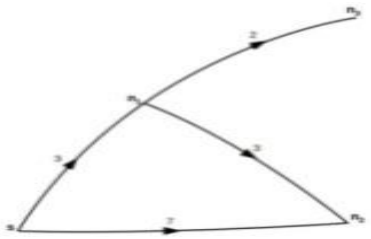
1. *Add the starting node to the open list.*
2. *Repeat the following steps:*
   a. *Look for the node which has the lowest f on the open list. Refer to this node as the current node.*
   b. *Switch it to the closed list.*
   c. *For each reachable node from the current node*
i. *If it is on the closed list, ignore it.*
ii. *If it isn't on the open list, add it to the open list. Make the current node the parent of this node. Record the f, g, and h value of this node.*
iii. *If it is on the open list already, check to see if this is a better path. If so, change its parent to the current node, and recalculate the f and g value.*
d. *Stop when*
i. *Add the target node to the closed list.*
ii. *Fail to find the target node, and the open list is empty.*
3. *Tracing backwards from the target node to the starting node. That is your path.*

Fig. Pseudocode of A* [3].

For any sub graph G and goal set T, let f(n) be the actual cost of an optimal path which go through n, from s to a preferred goal node of n. Determination of f(n) is the primary interest. In A* algorithm we see that, the function f(n) can be written as the sum of two functions:

$$f(n)=g(n)+h(n)$$

where g(n) is the actual cost from s to the node n, and h(n) is the actual cost from n to the preferred goal node of n. Let $\hat{g}(n)$ be the estimate of g(n). An excellant choice for $\hat{g}(n)$ is the minimum cost that has found so far. We observe $\hat{g}(n) \geq g(n)$.



Consider the subgraph shown in Fig. It consists of a start node s and three other nodes n1, n2 and n3. The arcs are shown with arrowheads and costs. Starting from s, we get successors n1, n2. The estimates $\hat{g}(n)$ and $\hat{g}(n2)$ are 3 and 7 respectively. Suppose A* expands n1 next with successors n2 ,n3. At this stage $\hat{g}(n3)=3+2=5$, and $\hat{g}(n2)$ is lowered to 3 + 3 = 6. The value of $\hat{g}(n1)$ remains equal to 3. Next we must have an estimate $\hat{h}(n)$ of h(n). Here we rely on information from the problem domain. Many minimum cost path problems through a paragraph has some information to estimate $\hat{h}$. In a city example where our paths are roads, we can use airline distance as the heuristic function. If h is any lower bound, then we can say that algorithm is admissible [8].There are variations of A* to optimize algorithm so that less memory is used .

The traditional A* algorithm has shortages as follows:

（1）It is slow in searching speed is slow and is poor applicability in the large scale path search environment. For example to get the optimal diagonal path in the 100*100 grid environment needs for searching 513 nodes at least.

（2）Due to the limitations of the traditional A* algorithm, the algorithm always lead to fall into failing situation in the unknown and complex grid environment.

（3）The traditional A* doesn't support path search operation between multi nodes at the same time which means generating different path from multi-starts to multi-target needs to retry.

2.1 HEURISTICS

Heuristics is a method used for experience based problem solving, which may or may not end up with an optimal solution. Algorithm's behavior based on the heuristic and cost functions can be very useful in a game.The trade off between speed and accuracy can be exploited to make your game faster. One way to construct an exact heuristic is to precompute the length of the shortest path between every pair of points. This is not feasible for most game maps. However, there are ways to ap proximate this heuristic:

- Fit a coarse grid on top of the fine grid. Precompute the shortest path between any pair of coarse grid locations.

- Precompute the shortest path between any pair of waypoints. This is a generalization of the coarse grid approach.

In a special circumstance, the heuristic can be exact without precomputing anything. If there is a map with no obstacles and no slow terrain, then the shortest path from the starting point to the goal should be a straight line.On a grid, there are well-known heuristic functions to use

- On a square grid that allows 4 directions of movement, use Manhattan distance

- On a square grid that allows 8 directions of movement, use Diagonal distance

- On a square grid that allows any direction of movement, might or might not want Euclidean distance.

- On a hexagon grid that allows 6 directions of movement, uses the Manhattan distance adapted to hexagonal grids.

2.2 MANHATTAN DISTANCE

The standard heuristic for a square grid is the Manhattan distance. Look at cost function and find the minimum cost D for moving from one space to an adjacent space. In the simple case, we can set D to be 1.The heuristic on a square grid where you can move in 4 directions should be D times the Manhattan distance:

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

Set D to the lowest cost between adjacent squares. In the absence of obstacles, and on terrain that has the minimum movement cost D, moving one step closer to the goal should increase g by D and decrease h by D. When we find f (which is set to g + h) will stay the same; thats a sign that the heuristic and cost function scales match.

2.3 GRIDS

A grid map uses a uniform subdivision of the world into small regular shapes some- times called tiles. Common grids in use are square, triangular, and hexagonal. Grids are simple and easy to understand.If were using grids for pathfinding,units are not constrained to grids, and movement costs are uniform, We may want to straighten the paths by moving in a straight line from one node to a node far ahead when there are no obstacles between the two.If units can move anywhere within a grid space, or if the tiles are large, think about whether edges or vertices would be better choice for our application.A unit usually enters a tile at one of the edges (often in the middle) and exits tile at another edge.With pathfinding on tiles, the unit moves to the center of the tile, but with pathfinding on edges, the unit will move directly from one edge to the other. Obstacles in a grid system typically have their corners at vertices. The shortest path around an obstacle will be to go around the corners. With pathfinding on vertices, the unit moves from corner to corner. This produces the least wasted movement, but paths need to be adjusted to account for the size of the unit.This is referred as Vertex movement.

III. PROPOSED WORK

After performing the literature survey, it is clear that a lot more improvements and optimisations are possible in the field of pathfinding in video games. So in our paper , we are intended to work on the pathfinding and the path planning in games using the Artificial Intelligence principles.The proposed work is aimed at combining different approaches for pathfinding to effectively find out paths in video game environments using less resources by utilizing Artificial Intelligence Concepts.

In our paper , we put forward an Efficient Pathfinding Algorithm for video games by

- Optimizing search techniques.
- Effective terrain mapping.
- Improving Heuristics.

We here use the direction based approach for pathfinding. New approach is applied in a grid based environment. As most of the game worlds are divided into grids for simplicity, this method has scope in all of them. This method can also be extended to all types of grid based worlds.

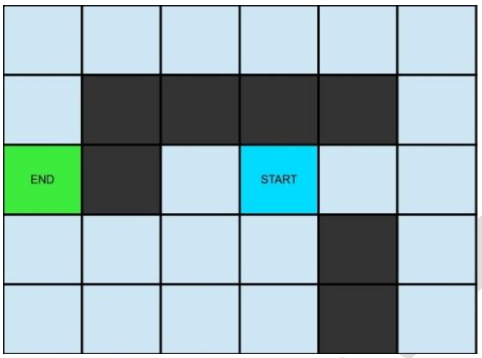
To find the shortest paths between two points in a map, is an important topic in mathematics and algorithm research. In the present gaming industry,pathfinding has its own requirements:

1. We do not really care too much whether a path is optimal in a mathematical sense, so long as it is virtually short enough.

2.. We do not want to devote too much resource for pathfinding in the gaming environment which which is usually resource constrained.

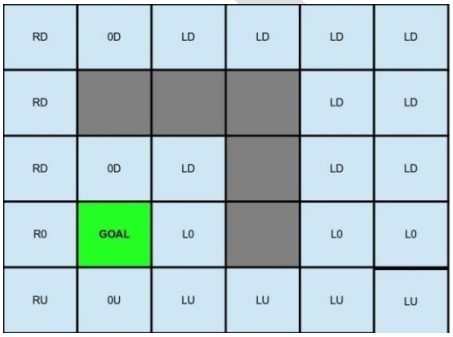
3.1 DIRECTION BASED HEURISTICS

In order to apply some sort of optimisation in pathfinding algorithms, the first thing needed is an efficient approach for pathfinding. The method I use here is applied on grid based maps. It uses the direction to the goal state as the heuristic function



A grid map example

In the above figure there is a starting and an ending point.The Heuristic Information needed is the relative position of a particular node to the end node. We start with the Start node.In order to calculate the information regarding the relative position of the goal node, In our approach, we use four letters(For explanation purpose). They are L, R, U, D which represents Left, Right, Up and Down respectively. Every node will have a Heuristic information which gives a clear idea about the relative position of that node with the goal node. In the above figure, the Heuristic Key can be recorded as L0 (Where 'L' represents left and '0' indicates that end state is in the same horizontal level as that of the start state.) The following figure gives more clear idea about direction based Heuristics



Grid showing direction based heuristics of all cell

As in the above figure, there are 8 Heuristic Keys applied in a pathfinding environment which are explained using table

Direction Key	Interpretation and direction of exploration
L0	Left
LU	Left – Up
0U	Up
RU	Right – Up
R0	Right
RD	Right – Down
0D	Down
LD	Left – Down

Direction Keys used for determining the direction of exploration

These information is used to determine which are the next nodes to be explored in order to reach the goal faster. For example, when the Heuristic key of a node is LU it has children in the top and in the left side. We then check whether those cells are free or not. A cell is said to be free if it is not previously explored or its not a non traversable cell. If there are no free children for a particular node, then we change the direction key by making L as R, R as L, U as D, D as U and '0' is kept as such.In usual practice with lightly obstructed map,this is not required in general. We continue to explore new children obtained through the changed Heuristic key. The method continues as such till it reaches the goal state.The above explained method is just a framework of the method. As it is very simple and straight, many optimizations are possible for the above mentioned method. In this work, We aim to implement the above direction concept so as to reduce the memory overheads and hence execution time.

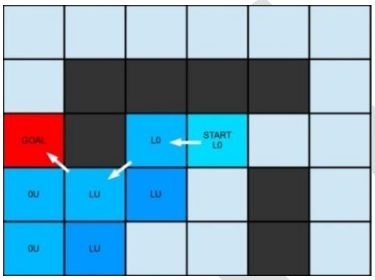
3.2.COMPARISON WITH OTHER ALGORITHMS IN A SMALL GRID

The new method is compared with other basic grid based pathfinding algorithms and results are analyzed. Here this analysis is made for some fixed grid environments. Preliminary analysis is conducted only to analyse the

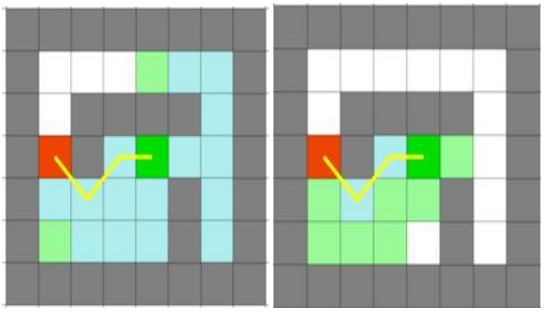
This algorithm is compared with other algorithms such as breath first search,,A* algorithm.Comparison results are shown below.Even though A* solves a given problem based on heuristics a necessary condition has to be satisfied. Basic comparison with the A* algorithm indicates that, this approach converges to the direction of goal as in the A* algorithm. Comparison with the other two algorithms also shows that the new approach converges to the direction of the goal state, even faster than Breadth First Search.These have to be proved mathematically.

3.3 CONVERGING BEHAVOIUR OF DIRECTION HUERISTICS

The method based on Direction Heuristic converges towards the goal state faster than Breadth First Search in tested environments. This makes it more usable. It also explores less number of nodes to reach final state. This is a good expected behaviour for a pathfinding algorithm. The proposed method explores less number of neighbors and hence the number of nodes processed each time is reduced. By using perfect data structures we can make it more appealing. So that the optimization for this particular work mainly concentrate on a data struture which can process nodes faster and which converges to the goal nodes so easily.

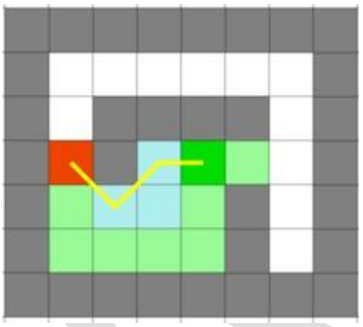


(a) Path obtained by direction heuristics



number of nodes explored in the Direction based Heuristics method and in other methods for a small graph.

(b) path obtained by breathfirst search and depth first serach



(c) Path obtained by A Star algorithm

The converging behaviour and thus the improvement of the work in obtaining a path in less time need to be theoretically proved. The converging behaviour of the algorithm of the algorithm can be explained as

- Compared to A* and other algorithms like Breadth First Search, the direction Heuristic approach explores less node each time. This is because of the fact that only those nodes in the direction of the goal node is considered in for expansion.

- Each time this approaches avoids a certain direction, in many of the cases the same direction is chosen. This implies that the portions in the graph which are not relevant is not searched

3.4 APPLYING THE CONCEPT OF DIRECTION BASED HEURISTICS ON A STAR ALGORITHM

Developed in the context of learning based heuristics our method speeds up search by expanding and evaluating only necessary nodes in the map. Further, this method eliminates redundant nodes from the graph which reduces additional memory over-heads. Using this heuristic function, we are able to identify a large set of cells which can be skipped. We have implemented the direction oriented Heuristic function normally in a simple grid based environment. The aim is to study the behaviour of the algorithm. Following are the scenarios taken under consideration:

- A square grid of typical size is used. The algorithm works on every kind of grid based maps. But for the ease of calculation of results and observing the performance, square worlds are considered.

- The territory type used is spacious maps and lightly obstructed maps.The method is based on finding out the

cells to be expanded out of many neighboring cells. So this is perfectly applicable in the above mentioned types. Also it can be applied on mazes. But for experimental purpose I used maps with less obstacle density.

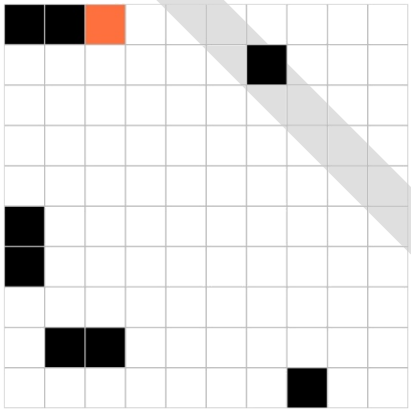
Every edge has given same weight as we are considering diagonal direction also. This is a mandatory step in the implementation. Expanding diagonal nodes improves the realistic movement which is very essential in game playing. There are a total of eight neighbors for a node under consideration. Sup- pose we are considering a particular node which can be denoted by (x,y) in a grid based environment. Then we can consider the neighboring nodes as a set:

$$\{(x-1,y-1),(x,y-1),(x+1,y-1),$$
$$(x1,y),(x+1,y),$$

$$(x-1,y+1),(x,y+1),(x+1,y+1)\}$$

Our particular aim is to eliminate some nodes from the above eight neighbors to optimize the result. Popular algorithms like the A star algorithm and Breadth First Search algorithm expands every node so that all nodes are to be processed.

Here Simple direction based Heuristics is applied for the implementation. We used the same datastructures as used by the A* algorithm and Breadth First Search algorithm to study the behaviour of the algorithm. From the above implementation, we could see that the proposed approach always finds solution without fail. A* algorithm can find out a solution pretty faster and the drawback is it eats up lot of memory. So an efficient optimization applied as this would greatly enhance the approach



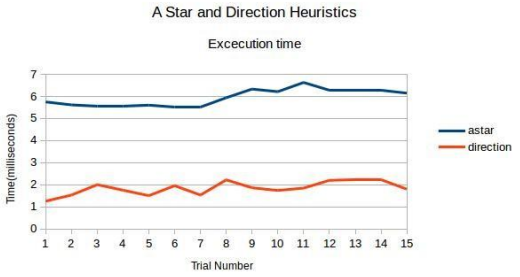
An example for game world simulated in browser.

IV. EVALUATION

Evaluating of the proposed system is perhaps the most important part of a our work. For precisely evaluating the results a common benchmark should be considered where all the algorithms under consideration can be executed. As we know A* algorithm is the most popular algorithm for pathfinding. Based on the time of excecution, Comparisons are made between a* algorithm and direction heuristic approach Every game world is represented as grids in the evaluation. For world representation a multi dimensional array is used. For getting the stable results in every platform,world representations in all such cases should be same.In most of the game worlds are represented using arrays. By using the same representation, results obtained are standard results which can be applied universally. A screen shot for running benchmark for the performance evaluation is given where it uses a 150 × 150 grid. The experiment is done in browser.

Map is represented using a 2D array for obtaining standard results.

First, both the algorithms are run several times and values are taken for execution time. The time thus calculated in milliseconds is plotted. An important observation is that the time taken by the proposed method is always less than that of A*. Both the algorithms, A* and Direction Heuristic approach are run thousand times and the average of the execution time is calculated for a grid of same size and obstacle density but randomly distributed obstacles. This result is given as the first instance. Fifteen such instances are calculated and result is plotted. By doing this we will get a result which is stable. Here also we have better performance for proposed algorithm



Comparision between A* and direction

Heuristics for 15 randon worlds of same size

V. CONCLUSION

In our work, we have introduced a new Heuristic method for speeding up Pathfinding on uniform cost maps which are represented using grids. The new approach selectively expands certain nodes from grid map so that minimum number of nodes are explored. The most important highlight of this work is identification of the nodes to be explored and proper usage of efficient data structure.we prove that unnecessary nodes are not expanded in this approach, or Direction Heuristics would try to minimize the number of nodes to be explored and yet come up with a solution in less time.

One of the important observation is that applying the new logic will not affect the solution.the solution is guaranteed by the new heuristic.it is simple yet highly efficient ,it reduces the amount of the memory needed by wisely choosing the nodes to be explored. It does not require any pre-processing and therefore it is a very fast method. Due to its simplicity it can easily be combined with other pathfinding methods, speedup methods and path smoothing methods to get a solution faster and there is scope for research in all these combinations. Direction based heuristic approach is highly competitive to other works from literature especially when dealing with large maps. When compared to the most popular A* algorithm and its optimized variation which is using priority queue for implementing data structures, Direction Based Heuristics is found to show improve ments.

The process speed up the path finding process by implementing an efficient data structure to handle the nodes for the evaluatation.This reflects in improvement in time when compared to traditional list based linear data structures. By efficiently deciding the number of nodes that are to be explored, new approach needs to process less number of nodes than other. This results in a memory efficient solution, Memory efficiency obtained by using Direction Heuristics is a key highlight when compared to A* algorithm which uses more memory. An interesting direction for further work is to extend Direction Based Heuristic approach to other type of grids like hexagonal and triangular grids. This can be easily achieved by employing proper direction keys for obtaining goal information pretty faster. Utility libraries and the pathfinding plug-ins can be developed by employing this idea so that it is avaliable as a package for the pathfinding. Much remains to be done in the field of Artificial Intelligence and pathfinding. Most of the research is oriented towards other areas like robotics and very few has been done towards their application in tile based games. We hope that our work would certainly benefit the game industry

VI. REFERENCES

[1] E. P. Hart, N. J. Nilsson;B. Raphael. A formal basis for the heuristic determination of minimum cost paths. IEEE Trans. Syst. Sci.Cybern:Vol. SSC-4, 1968,No.2:100 - 107 ,.

[2] Korf,R. Depth-first iterative-deepening: An optimal admissible tree search. Artificial Intelligence, 1985,27:97–109.

[3] Björnsson, Yngvi; Enzenberger, Markus; Holte, Robert C. Fringe Search: Beating A* at Pathfinding Game Maps; IEEE 2005 Symposium on Computational Intelligence and Games, 2005, 125-132.

[4] Björnsson, Yngvi;Vadim Bulitko ; Nathan Sturtevant. TBA*: Time-Bounded A*. Twenty-first International Joint Conference on Artificial Intelligence (IJCAI-09);2009，431-436.

[5] Rahul Kala, Anupam Shukla, Ritu Tiwari. Fusion ofprobabilistic A* algorithm and fuzzy inference system for robotic path planning, Artificial Intelligence Review, 334, 2010.

[6] Cen Zeng, Qiang Zhang, Xiaopeng Wei. GA-based Global Path Planning for Mobile Robot Employing A* Algorithm[J]. Journal of Computers, 72, 2012.

[7] Qing Xue, Yung-Ping Chien. Determining the path search graph and finding a collision-free path by the modified A* algorithm for a 5-link closed chain. Applied Artificial Intelligence, vol. 92, 1995.

[8] B. Stout, “The basics of A* for path planning,” in Game Programming GEMS, pp.254-262, Charles River Meida, America, 2000.

[9] A.Botea, M.Mueller, and J.Schaeffer, “Near optimal hierarchical path-finding,” J. GD, vol.1, no.1, pp.7-28, 2004.

[10] N.Nilsson, Artificial Intelligence: A New Synthesis, Morgan Kaufmann Publishers, San Francisco, 1998.

[11] P. Hart, N. Nilsson, and B. Raphael, “A formal basis for the heuristic determination of minimum cost paths,” IEEE Trans.Syst.Sci.Cybernet., vol.4, no.2, pp.100-107, 1968

[12] B. Stout, “Smart moves: intelligent path-finding,” in Game Developer Magazine, pp.28-35, 1996.

[13] P. Tozour, “Building a near-optimal navigation mesh,” in AI Game Programming Wisdom, pp.171-185, Charles River Media, America, 2002.

[14] S. Rabin, "A* speed optimizations," in Game Programming GEMS, pp.264-271, Charles River Media, America, 2000.

[15] N. R. Sturtevant and M. Buro, "Partial pathfinding usingmap abstraction and refinement," in *Proc. 20th Nat. Conf. Artif. Intell.*, 2005, pp. 1392–1397.

[16] N. R. Sturtevant, "Memory-efficient abstractions for pathfinding," in *Proc. Conf. Artif. Intell. Interactive Digit. Entertainment*, 2007, pp.31–36.