Dynamic Map Pathfinding Using Hierarchical Pathfinding Theta-Star (HPT*) Algorithm

Irfan Darwin[1], Suryadiputra Liawatimena[1,2]

[1]Computer Science Department, BINUS Graduate Program – Master of Computer Science, Bina Nusantara University, Jakarta 11480, Indonesia
[2]Computer Engineering Department, Faculty of Engineering, Bina Nusantara University, Jakarta 11480, Indonesia

Abstract Theta-Star is an efficient algorithm that can be used to find an optimal path in a map with better performance compared to the A-Star algorithm. Combining the Theta-Star with Hierarchical Pathfinding further enhances its performance by abstracting a large map into several clusters. What this combination lacks is the capability to handle a dynamic element in the map. Without that capability, the agent could potentially collide with elements in the map that is undesirable in certain conditions, while adding that capability might reduce the pathfinding algorithm's performance. The proposed algorithm aims to provide the capability to handle dynamic elements without severe negative impact on the performance of the algorithm. The effectiveness of the proposed algorithm is verified in terms of execution time, number of nodes explored, final path length, and the number of collisions that occurred.

***Index Terms*—Artificial Intelligence, Dynamic Map, Grid-Based, Hierarchical Pathfinding, Theta-Star.**

I. Introduction

Pathfinding is one of the basic yet essential tasks for Artificial Intelligence. Pathfinding algorithms have many uses, such as navigation applications, AI in a game application, or even autonomous driving. There are several algorithms for finding the optimum path to travel from one position to another, each with its advantages and disadvantages. Most of the current algorithm, however, primarily deals with a static map. Meanwhile, a specific condition requires a pathfinding algorithm to consider dynamic elements. The navigation application may require viewing a different route due to changing traffic. AI may be required to consider a potentially dangerous area, and an autonomous car will need to consider other cars' locations. Therefore, adding the capability to navigate a dynamic map to a pathfinding algorithm is important to allow the algorithm to be applied in more challenging conditions.

Many researchers have continuously made improvements to create a fast and accurate pathfinding algorithm. They were starting from the simplest one, which is Dijkstra's algorithm, also known as the Shortest Path First (SPF) algorithm. A few improvements were made into what becomes the A-Star (A*) search algorithm, where a heuristic function is added to determine the optimal path. From these algorithms, another improvement was made to make the algorithm suitable for many nodes, which is then called the Hierarchical Pathfinding (HP) algorithm.

Based on the comparison between several pathfinding algorithms, the algorithm which has a good overall performance is the Hierarchical Pathfinding Theta-Star algorithm, which can calculate the optimum path to get from one point to another with better memory usage while maintaining the efficiency of the resulting path [1]. The algorithm works by combining the characteristics of the Hierarchical Pathfinding algorithm and the Theta-Star algorithm. The Hierarchical Pathfinding algorithm provides the capabilities to process a large number of nodes by separating them into several smaller grids. Some of the grids partitioned from the Hierarchical Pathfinding algorithm already offer the optimal path, which the Theta-Star algorithm will disregard. Then, the grids that do not yet have an optimal path will be further processed by the Theta-Star algorithm. This method allows the Hierarchical Pathfinding Theta-Star algorithm to provide an optimum path for a large grid with the minimum number of processed nodes.

While the Hierarchical Pathfinding Theta-Star algorithm provides a good result for a static map, improvements can still be made to enable the Hierarchical Pathfinding Theta-

Star algorithm to be used on a dynamically changing map. A dynamically changing map is defined as a map containing elements such as moving obstacles or, more generally, a map in which a path could become invalid at one time and valid at another. The main contribution of this paper is expected to be an algorithm that allows the Hierarchical Theta-Star algorithm to be used on a dynamic environment. Meanwhile, the performance of the algorithm should not be significantly worse than the original algorithm. Additionally, the paper also aims to provide additional insight by implementing the algorithm on a hexagon-grid instead of the usual square-grid.

II. LITERATURE REVIEW

A. *Static Map Pathfinding*

Pathfinding algorithm was developed as early as 1956, starting from Djikstra's algorithm, and has continued to create until now. Most algorithms primarily deal with a static map. A map is considered static if it contains only stationary obstacles and does not change while the agent deliberates which path is optimum [2]. This condition allows the pathfinding algorithm to plan the optimal path from the starting point to the destination only once and ensures that once found, the path will remain valid.

Algorithms such as the A-Star provides a simple and efficient way to calculate the optimum path [3]. While other algorithms, such as the Theta-Star, improves the execution time and memory usage of the algorithm by optimizing the number of nodes that needed to be visited to determine the optimum path [4]. Further improvement to the execution time was also provided by the Lazy Theta-Star [5]. While there are some improvements such as the Cluster Theta-Star (C-Theta*) algorithm improves the Theta-Star algorithm's performance by dividing the map into several non-uniform clusters [6]. Other variation includes the combination of the Theta-Star with a hybrid A Star algorithm [7], using a visibility graph as a pathfinding method [8], and another variation that aims to improve the line-of-sight check's efficiency called the Batch-Theta-Star [9].

The accuracy of the path produced by both the A-Star and Theta-Star algorithm is proven to optimal given an accurate heuristic function. To determine whether the heuristic function used in the pathfinding algorithm is accurate, one of the criteria required is for the heuristic function to be admissible. For a heuristic function to be admissible, its estimated cost of reaching the destination must never exceed the actual cost [10]. Admissibility is more comfortable to achieve on a static map since the initial cost estimate will not change over time. Therefore, one of the improvements that can be made to both algorithms is by adding the algorithm's capability to provide an accurate path in a dynamic map.

B. *Dynamic Map Pathfinding*

One of the pathfinding algorithms that can provide an accurate result in a dynamic environment is the D-Star [11]. The D Star classifies the dynamic environment into several categories: a known dynamic environment, a partially known dynamic environment, and a totally unknown dynamic environment based on the observability of the environment. A known dynamic environment means that information such as the path of the moving obstacle is known. While a totally unknown dynamic environment means that only very limited information is initially provided, requiring the agent to refine its path on the go.

The D-Star algorithm provides the basis which could be used to allow a pathfinding algorithm to be used on a dynamic map. In a dynamic map, the agent should also take into account the movement of an obstacle to ensure no collision will occur while keeping the path as short as possible. A specific path in a dynamic map might be shorter than another but has a high chance of causing a collision between the agent and an obstacle. At the same time, a longer path might provide a much safer route to the destination cell. A dynamic map pathfinding algorithm should be able to choose which path is optimum.

Based on the environment's observability, the behavior of the algorithm itself should change to be able to perform optimally. This algorithm could be further improved by increasing the algorithm's performance when used in a larger known dynamic environment. This improvement can also be applied to a partially known dynamic environment, as long as one of the known information is the map's size. While for a totally unknown dynamic environment, this improvement might be challenging to achieve.

There are also other dynamic map pathfinding algorithms, most of them are based on the A-Star algorithm. Such as the Hierarchical Pathfinding Lifelong Planning A Star (HPLPA*) are algorithms that combine several algorithms to enable an A-Star algorithm to be implemented in a dynamic map [12]. Another dynamic pathfinding algorithm such as Dynamic Hierarchical Pathfinding A Star (DHPA*) [13], and there is also research on optimizing the graph to improve the performance of the algorithm used [14].

C. *Hierarchical Pathfinding*

Hierarchical Pathfinding provides a method to allow a pathfinding algorithm to deal with a large environment. By dividing a large environment into several clusters, a pathfinding algorithm can then be used locally on each cluster to find the optimum path. This method works similarly to a command structure, where the highest-level entity will decide the general strategy and delegate its implementation to a lower-level entity. The delegation will continue until it reaches the lowest level entity that will perform the actual action, determining the optimal path.

Hierarchical Pathfinding has been applied to both the A-Star and Theta-Star algorithms. Based on the test performed, the Hierarchical Pathfinding method combined with the Theta-Star algorithm is proven to be more efficient than Hierarchical Pathfinding combined with the A-Star

algorithm [15]. The variables compared to determine the performance of both algorithms are path lengths, number of node visits, and nodes in memory. Since both algorithms used are static pathfinding algorithms, improvements can be made to allow a combination of the Hierarchical Pathfinding method with dynamic map pathfinding. Precisely which algorithm will be chosen and combined will impact the performance and its ability to handle certain factors such as the environment's observability.

There is a constraint when combining the Hierarchical Pathfinding method with a dynamic map pathfinding algorithm. The constraint is that the agent's information should include the structure of the map that the agent will traverse. Using that information, the agent will be able to split the map into several smaller clusters. However, a dynamic map with a totally unknown dynamic environment classification cannot implement the Hierarchical Pathfinding method since the agent has little or no information about the map to split it into smaller clusters effectively.

III. RESEARCH METHODOLOGY

A. Benchmark

The experiments are done using the benchmark grids from the Dragon Age computer game [16]. The grid is provided in a text file format, which will be interpreted by the simulator to form the environment. Each character represents a different type of grid.

```
type octile
height 49
width 49
map
TTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTT
TTT.............TTTT.TTT...TTTT.TTTT.............TT
TT..............TTT.........TTT..TTT..............TT
T.................................................T
T.................................................T
T.................................................T
T.................................................T
T.........................TT.......................T
T........................TTT......................T
T........................TTT......................T
T.................................................T
T.................................................T
T.................................................T
T.................................................T
T.................................................T
TTT.............TTTT..............TTTT.............TT
TTT.............TTTT..............TTTT.............TT
TTT.............TTTT..............TTTT.............TT
TT..............TTT................TTT..............TT
TT.................................................T
```

Fig. 1. Text Representation of the Benchmark Grid Used in the Experiment (Partial)

As shown in Fig. 1, the grids provided have several characters used to identify each cell's type in the grid. The experiments are conducted while keeping the original definition for the "." character, which represents a walkable cell. The changes made are on the definition for the "T" character. Initially, the "T" represents trees which is impassable. In this experiment, the "T" will represent a trapped cell, the grid's dynamic part. A trap cell will continuously switch between the "off" and "on" states. While the trap is in the "on" state, the agent moving to that cell will count as a collision.

Apart from the text representation of the map, a total of 130 scenarios are also provided benchmarking purpose.

```
version 1
0   arena.map   49  49  19  26  19  29  3.00000000
0   arena.map   49  49  44  30  43  28  2.41421356
0   arena.map   49  49  31  23  33  23  2.00000000
0   arena.map   49  49  30  22  31  21  1.41421356
0   arena.map   49  49  40  14  43  16  3.82842712
0   arena.map   49  49  11  1   13  1   2.00000000
0   arena.map   49  49  9   40  7   41  2.41421356
0   arena.map   49  49  12  6   15  4   3.82842712
0   arena.map   49  49  47  30  44  28  3.82842712
0   arena.map   49  49  45  46  44  43  3.41421356
```

Fig. 2. Text Representation of the Benchmark Scenario Used in the Experiment (Partial)

These scenarios place a different starting and destination point for the agent. By executing each of the pathfinding algorithm using the same map and scenario, a comparison can be made to determine the efficiency of each algorithm.

Fig. 2 displays the text representation of several scenarios from the 130 total scenarios for the map. The text is formatted in a certain pattern, the leftmost value indicates which bucket the scenario belongs to. Each bucket contains 10 scenarios which means there is a total of 13 buckets for this set of scenarios. The bucket value serves to categorize the scenario based on its complexity, with more complex scenario having a higher bucket value. The second value indicates which map this scenario is intended for. The third and fourth value indicates the width and height dimension of the map, respectively. The fifth and sixth value indicates the coordinate of the starting point, while the seventh and eight value indicates the coordinate of the destination point. Finally, the ninth value indicates the optimal length of the scenario, calculated by the square root of the diagonal cost of the distance between the starting and destination point given in the scenario.

B. Map Representation

In order to visualize the path formed by each of the pathfinding algorithm, the application developed for the simulation will convert the text representation into an image representation. While originally represented by a square grid, the application for this experiment will use a hexagon grid. A different color will be used to differentiate one type of cell from the other, with a blue-colored hexagon representing the starting position of the agent and a dark green-colored hexagon representing the destination that the agent needs to reach.

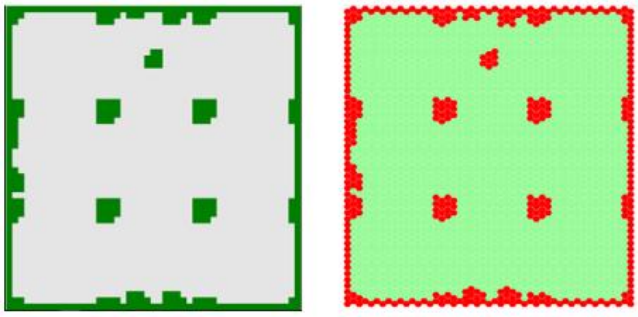

Fig. 3. Original Square-Grid Representation (Left) and Hexagon-Grid Representation Used in the Experiment (Right)

The specific map used in the experiment is the arena map. There are two types of terrains in this map, the "." and "T" cell. In the application used, the "." cell will be represented by a green-colored hexagon which is walkable, while the "T" cell will be represented by a red-colored hexagon which could cause a collision if stepped on by the agent at a particular time.

C. *Proposed Algorithms*

The following pseudocode describe the general idea about the initial idea to implement the capability to traverse a dynamic map in a hierarchical pathfinding algorithm. The first step is to convert the initial input, which is given in text form, into a graph format which the algorithm will use to plan the path. After that, following the procedure for the Hierarchical Pathfinding algorithm, the graph will be split into clusters. During the cluster creation, the cluster will be categorized into a dynamic cluster if it contains a dynamic

```
var solution = [];

function dynamicHierarchicalPathfinding(grid, clusterSize){

    // Convert the initial input into a graph format to traverse
    var graphToTraverse = convertToGraph(grid, clusterSize);

    // Split the graph into clusters
    var clusterGraph = splitIntoClusters(graphToTraverse);

    // Create the return path for each cluster
    for (let cluster of clusterGraph) {
        refinePath(cluster);
    }

    return solution;

}

function refinePath(cluster){

    // Traverse using a dynamic pathfinding algorithm for dynamic cluster and vice versa
    if(cluster.isDynamic){
        solution.push(traverseDynamically(cluster));
    } else {
        solution.push(traverseStatically(cluster));
    }

}
```

Fig. 4. Pseudocode for the Modified Pathfinding Algorithm

element and a static cluster otherwise.

Based on the cluster category, the appropriate pathfinding algorithm will be used. A dynamic cluster will be traversed using a dynamic pathfinding algorithm, while a static cluster will be traversed using a static pathfinding algorithm. This setup aims to minimize the performance impact of using the more complicated dynamic pathfinding algorithm by only using the algorithm when it is necessary.

The algorithm used for the dynamic traversal is based on the Theta-Star algorithm. The modification made to the algorithm is located on the visibility checking part of the algorithm. Originally, the algorithm would only consider

```
function dynamicVisibilityCheck(fromCell, toCell){

    // Determine other cells which determine the visibility between two cells
    var cellsToConsider = determineCellsToConsider(fromCell, toCell);

    for (let cell of cellsToConsider) {
        if(cell.isStaticObstacle || cell.isActiveDynamicObstacle){
            return false;
        }
    }

    return true;
}
```

Fig. 5. Pseudocode for the Modified Visibility Check Algorithm

static obstacle as something which could block visibility between two cells.

The modified algorithm adds another condition (highlighted in green on Fig. 5) that the algorithm also considers a dynamic obstacle as something that can block visibility if the dynamic obstacle is in an active condition. Otherwise, the cell is treated as a walkable cell since an inactive trap cell will not count as a collision.

IV. RESULT AND DISCUSSION

Six pathfinding algorithms are considered during the experiment. A-Star algorithm (A*), Theta-Star algorithm (Theta*), Modified Theta-Star algorithm (Mod Theta*), and the Hierarchical counterpart of the three algorithms (HPA*, HPT*, and Mod HPT*). Each of the pathfinding algorithm is executed to provide the return path for each one of the 130 scenarios provided with the map. The result of each execution is then stored and then further processed to determine the general performance of the algorithm on this particular map under varying scenarios. The following figure

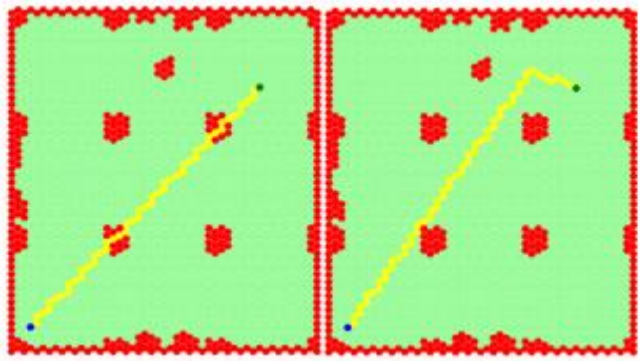

Fig. 6. Return Path Theta-Star (Left) and Modified Theta-Star (Right)

illustrates the different path produced by Theta Star Algorithm compared to the Modified Theta Star Algorithm when executed on one of the 130 scenarios.

Fig. 6 shows the return path comparison between the Theta-Star algorithm and the Modified Theta-Star algorithm. The yellow-colored cell indicates the path which the algorithm produces. The Theta Star Algorithm creates a

path that passes through the red-colored grid, which results in some collision. Meanwhile the Modified Theta Star creates a longer path which evades the red-colored grid, choosing to minimize the number of collisions.

The total result for the execution time, path length, and number of visited nodes are averaged, while number of collisions is summed for the comparison between each of the pathfinding algorithm. Execution time indicates how long the algorithm requires to find the return path and is counted in milliseconds (ms), meanwhile explored node indicates how many cells the pathfinding algorithm need to consider before finding the final return path. Return path length

Fig. 7. Execution Time Comparison Chart

indicates how many steps the agent needs to take to reach the destination, meanwhile collision count indicates how many times the agent steps on an active trap cell. The following figures compares the execution result of each of the six algorithms.

Fig. 7 shows the execution time comparison between the six pathfinding algorithms. The average execution time for the A Star algorithm is lower compared to the Theta Star

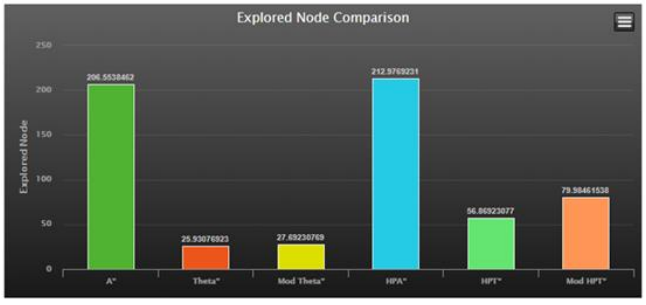

Fig. 8. Explored Node Comparison Chart

algorithm and the Modified Theta Star algorithm. Implementing the Hierarchical Pathfinding algorithm improves the execution time of each of the algorithm. In comparison, adding the dynamic pathfinding capability to the Theta algorithm does not negatively impact its execution time significantly.

Fig. 8 shows the number of explored graph nodes comparison between the six pathfinding algorithms. The average number of explored nodes for the A Star algorithm is higher compared to the Theta Star algorithm and the Modified Theta Star algorithm. Implementing the Hierarchical Pathfinding algorithm increases the number of

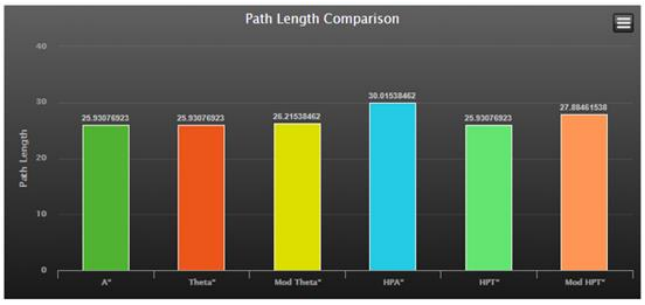

Fig. 9. Return Path Length Comparison Chart

nodes explored due to the preprocessing needed to split the map into clusters. In comparison, adding the dynamic pathfinding capability to the Theta Star algorithm further increases the number of nodes explored due to the need to find alternative path that will not cause a collision.

Fig. 9 shows the return path length comparison between the six pathfinding algorithms. The average path length for the three algorithms is relatively similar. Implementing the

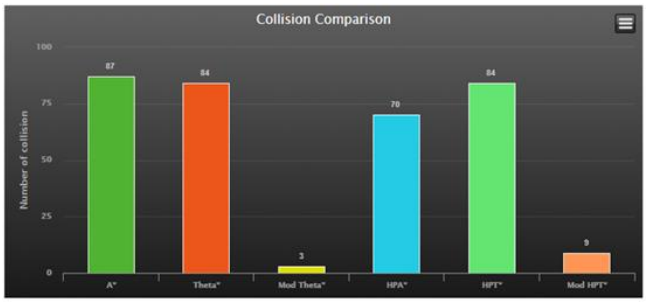

Fig. 10. Collision Count Comparison Chart

Hierarchical Pathfinding algorithm increases the length of the return path for the A Star algorithm due to the lack of path smoothing on the final hierarchical result. In comparison, adding the dynamic pathfinding capability to the Theta Star algorithm also increases the length of the return path due to the need to find alternative path that will not cause a collision.

Fig. 10 shows the collision count comparison between the six pathfinding algorithms. Both the A Star and Theta Star algorithm has a similar number of collisions. Implementing the Hierarchical Pathfinding algorithm does not significantly change the number of collisions. In comparison, adding the dynamic pathfinding capability to the Theta Star algorithm successfully reduces the number of collisions that would otherwise occur.

V. CONCLUSION

Based on the total number of collisions shown in Fig. 10, the modified Hierarchical Theta Star algorithm can reduce the number of the collision of the final path in most scenarios. This result is achieved while maintaining the efficiency that hierarchical pathfinding provides by abstracting the map into several clusters, reducing the

number of explored nodes, and reducing execution time as shown in Fig. 7 – 8. Using the Theta Star algorithm as the main pathfinding algorithm ensures that the abstraction will not cause the final path to become significantly longer as shown in Fig. 9. Based on the result shown from Fig. 7 – 9, a Hierarchical Theta Star algorithm could be extended to be able to handle a hexagon grid and dynamic elements without significant penalty on its performance.

REFERENCES

[1] L. van Elswijk, "Hierarchical Path-Finding Theta*," pp. 11–13, 2013.

[2] S. J. Russell and P. Norvig, in Artificial intelligence: a modern approach, Upper Saddle River: Prentice-Hall, 2010, pp. 42–44.

[3] X. Cui and H. Shi, "A*-based Pathfinding in Modern Computer Games," International Journal of Computer Science and Network Security, pp. 125–130, 2011.

[4] K. Daniel, A. Nash, S. Koenig, and A. Felner, "Theta*: Any-Angle Path Planning on Grids," Journal of Artificial Intelligence Research, vol. 39, pp. 533–579, 2010.

[5] A. Nash, S. Koenig, and C. Tovey, "Lazy Theta*: Any-Angle Path Planning and Path Length Analysis in 3D.," Proceedings of the AAAI Conference on Artificial Intelligence, vol. 24, no. 1, 2010.

[6] P. Mendonca and S. Goodwin, "C-Theta*: Cluster Based Path-Planning on Grids," 2015 International Conference on Computational Science and Computational Intelligence (CSCI), 2015.

[7] В. Г. Михалько and І. В. Круш, "Effective pathfinding for four-wheeled robot based on combining Theta* and hybrid A* algorithms," ScienceRise, vol. 7, no. 2 (24), p. 17, 2016.

[8] N. B. Abdul Latip, R. Omar, and S. K. Debnath, "Optimal Path Planning using Equilateral Spaces Oriented Visibility Graph Method," International Journal of Electrical and Computer Engineering (IJECE), vol. 7, no. 6, p. 3046, 2017.

[9] V.-H. Dang, N. D. Thang, H. H. Viet, and L. A. Tuan, "Batch-Theta* for path planning to the best goal in a goal set," Advanced Robotics, vol. 29, no. 23, pp. 1537–1550, 2015.

[10] R. E. Korf, "Recent Progress in the Design and Analysis of Admissible Heuristic Functions," Lecture Notes in Computer Science, pp. 45–55, 2000.

[11] F. A. Raheem and U. I. Hameed, "Heuristic D* Algorithm Based on Particle Swarm Optimization for Path Planning of Two-Link Robot Arm in Dynamic Environment," Al-Khwarizmi Engineering Journal, vol. 15, no. 2, pp. 108–123, 2019.

[12] Yan Li, Wenju Zhao, Zhenhua Zhou, and Cai Chen, "Hierarchical and Dynamic Pathfinding Algorithms in Game Maps," International Journal of Advancements in Computing Technology, vol. 5, no. 11, pp. 87–98, 2013.

[13] A. Kring, A. Champandard, and N. Samarin, "Dhpa* and shpa*: Efficient hierarchical pathfinding in dynamic and static game worlds," Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, vol. 5, no. 1, 2010.

[14] W. Zhu, D. Jia, H. Wan, T. Yang, C. Hu, K. Qin, and X. Cui, "Waypoint Graph Based Fast Pathfinding in Dynamic Environment," International Journal of Distributed Sensor Networks, vol. 11, no. 8, p. 238727, 2015.

[15] A. Botea, M. Muller, and J. Schaeffer, "Near-optimal hierarchical pathfinding," Journal of Game Development, vol. 1, no. 1, pp. 1–30, 2004.

[16] N. R. Sturtevant, "Benchmarks for Grid-Based Pathfinding," IEEE Transactions on Computational Intelligence and AI in Games, vol. 4, no. 2, pp. 144–148, 2012.