

Chapter 1

Introduction

Real Time Strategy is a sub-genre of strategy video games. Real time strategy games require the player to manage the units given to them under some certain scenarios, the game usually tasks the player to take control of a unit (civilization, city, group, etc.) and must improve it until the end of the game. In an RTS, the players must adapt to the environment, location, and condition of the unit they control and must devise strategies to keep its survival. With how RTS are usually played, the games under it shine the most when played by multiple players due to its competitive nature. That said, playing RTS games alone does not mean that the experience deteriorates.

When playing alone, players must deal with opposing units that are controlled by Bots or AI (Artificial Intelligence). These bots are programmed to deter the players of their progress by attacking and depriving them of resources. This gives the player a different yet still entertaining experience when playing RTS games.

Pathfinding algorithms are used to give the AIs a certain distinction and understanding on their surroundings. It lets them choose a path that will reach the destination with the least distance covered.

HPA* consists of a build algorithm and a search algorithm. The build algorithm defines the hierarchy through a series of graphs, where each graph abstracts a higher resolution graph. After the hierarchy is prepared, the search algorithm finds a path at the highest level, and refines it into a series of segment paths along the lowest level. The utility of HPA* is that a great deal of computation can be done in the preprocessing stage making the actual pathfinding task much faster. When a path is requested between locations a and b, all that is needed is to temporarily connect them to the pre-calculated graph by making small Dijkstra searches on the original cost raster within the blocks containing a and b, then calculate a path between them on the graph using A*.

Existing Algorithm

The Non-Playable Characters in the game use HPA* as a pathfinding algorithm. But, the way the algorithm is implemented to the application lacks in quality which then influences the performance of the AI.

Algorithm:

Start:

Pre-processing

1. Divide the map into clusters Problem 1
2. For every set of adjacent clusters, identify an entrance connecting them.
3. For each entrance, define one or two transitions depending on the entrance width. Problem 3
4. For each transition, define an edge connecting two nodes called inter-edges.
5. For each pair of nodes inside a cluster, define an edge linking them as intra-edges. Problem 2
6. Store information on disk that the grid is ready for hierarchical search.

On-line search

1. Insert start and goal positions in the abstract graph.
2. Use A* to search for a path between start and goal.

- a. Provides an abstract path that moves the start to the border of the cluster containing it.
- b. Provides an abstract path to goal's cluster.
- c. Provides an abstract path from the goal's cluster to the goal.

Statement of Problems

The following problems have been observed in the given and current technology:

1. The algorithm only scans rectangular-shaped obstacles.

The algorithm divides the grid map and searches the best path using this method. The downside of this method is that the algorithm can only scan for rectangular-shaped obstacles, non-rectangular obstacles cause the algorithm to produce non-optimal solutions. The figures below show how a grid map sets a non-rectangular obstacle on the grid.

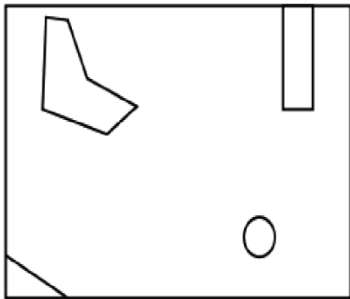


Figure 1 A terrain with non-rectangular obstacles

A sample terrain where non-rectangular obstacles are scattered and placed on it.

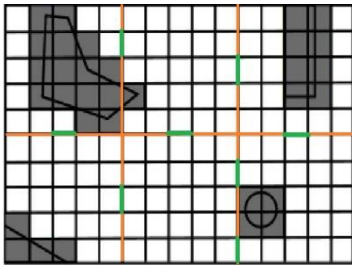


Figure 2 *HPA* applied on terrain from figure 1*

The terrain is split into grids and divided into clusters, the shaded area are the places where the algorithm has deemed the node to be untraversable.

The algorithm takes a lesser optimal path when encountering non-rectangular obstacle as shown by Table 1 The unit circles around the obstacle and traverses more nodes than what it needs to pass through.

Table 1 *Terrain with non-rectangular obstacle applied with HPA* algorithm*

| Trial | Open List | Closed List | Clusters | Nodes Passed | Total Weight |
|-------|-----------|-------------|----------|--------------|--------------|
| 1 | 12 | 8 | 4 | 8 | 80 |
| 2 | 18 | 8 | 4 | 8 | 80 |

2. Dynamic obstacles prove to be a problem for the algorithm.

Although the algorithm can handle reading obstacles, it has difficulty when handling dynamic obstacles. This causes a huge burden on the computation of the algorithm when trying to pass dynamic obstacles, an example of which are moving units. The figure below displays a unit trying to pass a group of units blocking a path. Units are considered as dynamic obstacles. The red and blue line is the possible optimal path; the green line is the path the unit traversed to get to its destination.

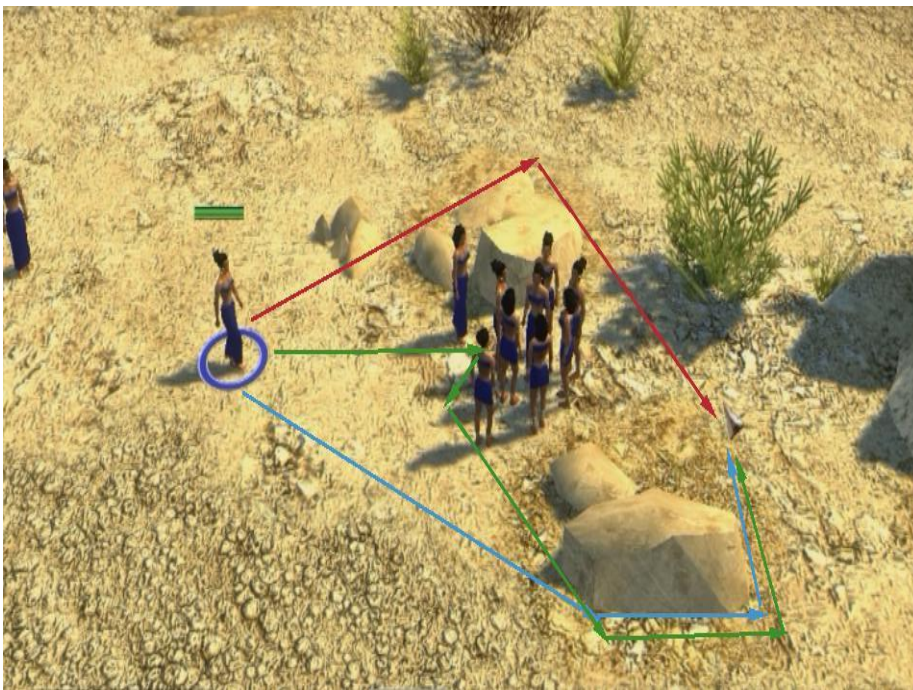


Figure 3. Interaction of the unit to dynamic obstacles (different unit) on HPA*

Figure 3 shows that the unit proceeds forward until it reaches the group of units which were considered as dynamic obstacle. The green arrow shows the path the unit takes while the red and blue arrows show the path where the unit should have traversed.

The performance of HPA* algorithm when encountering static obstacles is showed at table 2

Table 2. Performance of HPA* on static obstacles

| Trial | Open | Closed | Clusters | Nodes | Total |
|-------|------|--------|----------|-------|-------|
| 1 | 14 | 8 | 3 | 5 | 64 |

Dynamic obstacles make the algorithm calculate for more nodes than when it encounters static obstacles because it circles around the whole obstacle rather than considering the path it may have.

Table 3. Performance of HPA* on dynamic obstacles

| Trial | Open | Closed | Clusters | Nodes | Total |
|-------|------|--------|----------|-------|-------|
| 1 | 19 | 10 | 3 | 9 | 128 |

Objectives of the Study

The objective of the study is to help improve the Artificial Intelligence currently used by modern games such as Real Time Strategies. The use of pathfinding algorithms will help the AIs to make the best decision that will give the players a better experience in playing the game.

Specific Objectives

- 1.** To enhance the algorithm and allow it to scan for an optimal path when encountering non-rectangular obstacles on the map by using navmesh in scanning obstacles on the terrain.
- 2.** To help the algorithm compute both static and dynamic obstacles with the least possible work time and used resources by adjusting the pre-process of the algorithm using navmesh to calculate for the dynamic obstacles also present on the terrain.
- 3.** To find the best transition place based on the grid map and increasing the optimality of the path taken by the algorithm.

Significance of the Study

The study seeks to benefit the following people:

- To the **game developers**, this study will help them know the current issues of pathfinding Artificial Intelligence on real time strategy games and will help them advance and improve the quality of the AI.
- To the **gamers**, the study will let them know the existing problems found on RTS games which will help them identify the problem.
- To the **students**, that the study will let them be interested to games, as well as game development and help them expand their choices on their career.
- To the **other developers**, that the study will be useful for them on understanding the problems of pathfinding algorithms and knowing the solutions for it.
- Lastly, to the **future researchers**, that the study will help them on their own study and may help on furthering the study through finding out better solutions on recent game AI problems.

Scope and Limitations

The study covers the understanding on how an Pathfinding works in Nightmares, a Real-Time Strategy game. To improve the algorithm's ability on how it performs, how it acts based on its surroundings, and identifying the path it should take. The researchers will use Unity in developing the game that will be used to check the performance of the enhanced algorithm.

The study will not cover on how an AI works nor what an AI is. The study will only focus on how Pathfinding is used on Nightmares. Other Real Time Strategy games will not be a part of the study, such as: "Age of Empires", "Civilization", "Red Alert", "Warcraft", etc. Though the idea or logic on how pathfinding algorithms are used on mentioned games will be used by the researchers for reference purposes only. The study will not include how pathfinding is used on outside systems such as robotics.

Definition of Terms

Algorithm• A self-contained step-by-step set of operations to be performed.

Artificial Intelligence• Used to generate intelligent behaviors primarily in non-player characters (NPCs), often simulating human-like intelligence.

Closed List• List of nodes that has been listed off by the algorithm. These are commonly the nodes where the algorithm is.

Dynamic Obstacle• Obstacles that move along the surface.

Heuristic• A technique designed for solving a problem more quickly when classic methods are too slow, or for finding an approximate solution when classic methods fail to find any exact solution.

HPA*• Hierarchical Pathfinding A* algorithm.

Navmesh• An abstract data structure used in artificial intelligence applications to aid agents in pathfinding through complicated spaces.

Nodes• Multiple points used to traverse between paths.

Non-playable characters• Characters in a game that cannot be controlled by a player.

Open-source• Computer software with its source code made available with a license in which the copyright holder provides the rights to study, change, and distribute the software to anyone and for any purpose.

Open List• List of nodes that can be accessed by the algorithm from the current node.

Optimal path• Refers to the most desirable or most favorable path where the units can traverse the terrain.

Pathfinding• The plotting, by a computer application, of the shortest route between two points.

Player• The person the controls the character in a game.

Units• The characters controlled by both the AI and the player.

Chapter 2

Related Literatures and Studies

The HPA* Algorithm calculates its paths with the use of a generated grid system. As the algorithm sees the entire map as a grid with rectangular terrain, it will only be able to distinguish paths and obstacles that are within the bounds of a single grid square. This means that irregularly shape obstacles (like bodies of water, brush) may be miscalculated by the pathfinder in the sense that a tree branch that is protruding from an obstacle tree's grid may cross into the grid of a non-obstacle. In the study ***“Study and Development of Hierarchical Path Finding to Speed Up Crowd Simulation”*** by C.F. Paredes, November 2014, he uses extra methods to double-check the creation of grids to make sure that passable and non-passable terrain do not overlap each other's grids. To make things easier for the computer, he had to design his map elements to minimize the possibility of this overlapping. Failure to do so may cause glitches or bugs like bots' inability to pass through terrain that it thinks is passable.

At the beginning of the algorithm, a “pre-caching” is performed on the map. This means that the map will first be divided into a rectangular grid. Once the grid is determined, the algorithm looks for the “passable” entrances to the other grids. The problem in this is because it is the entrances that are given scanning emphasis rather than the insides of the

grid. This will cause a problem if, for example, a tiny edge of passable terrain is caught in a grid which is otherwise filled with impassable terrain. A likely solution to this is to provide a degree of advanced scanning. When a cluster is scanned for pass-ability, the nodes adjacent to it will be pre-scanned to see whether there are dead ends to each cluster. This may not only prevent AIs developing a false path, but may also reduce the frequency of “path-hugging” (e.g. “hugging”/moving right at the edge of every impassable terrain) and increasing the chances of finding the optimal path every time.

In the study by **A. Kring, A. J. Champandard, and N. Samarin** titled ***“DHPA* and SHPA*: Efficient Hierarchical Pathfinding in Dynamic and Static Game Worlds”*, 2011**, the authors mentioned the careful intentions of a programmer to minimize dynamics in their maps (e.g. destructible obstacles, environment alterations due to player actions) as map changes require a degree of rescanning. And since HPA* works in multiple threads that run new requests in queues, changes in the map will have to be queued for pathfinding which may cause lag due to possible bottlenecking in the CPU. The lag mentioned here isn’t of the frame rate nature (FPS dropping), but rather the quality of the AI response (the AI not knowing that a new, shorter path had just been opened up and choosing to take an old, longer path that was previously scanned.)

As mentioned in the first one, the map is pre-cached. The algorithm processes and finds paths on the map before the player even gets to do anything. That means that if there were to be obstacles or terrain that move, the pre-cache would be completely irrelevant since a change in obstacle position can mean new paths.

In a study, **Pathfinding in Two-Dimensional Worlds by Anders Strand-Holm Vinther and Magnus Strand-Holm Vinther (2015)**, Pathfinding is used to solve the problem of finding a traversable path through an environment with obstacles. This is a problem seen in many different fields of study, which include robotics, video games, traffic control, and even decision making. These areas rely on fast and efficient pathfinding algorithms. This also means that the pathfinding problem appears in many different shapes and sizes. Applications in need of pathfinding will prioritize things differently and lay down different requirements on the algorithms. It is therefore worth exploring and comparing a wide variety of algorithms, to see which ones are better for any given situation. The problem of pathfinding is an easy one to understand. Planning a path, from where you are, to where you want to go, is something you do daily. The hard part of pathfinding comes when we want computers to do it for us. Applications nowadays often put strict requirements on running time, memory usage and path length. Add to that a large, may be even dynamic, environment, and you have a complex problem with many aspects to consider.

Synthesis:

With the recent popularity of video games, many developers are trying out new ways to reinvigorate the concept of old games. As technology improves every passing day, the games that are made with it improve as well. But game developers had problems on how to find the optimal path when using pathfinding algorithms. Many algorithms have been introduced and applied to various games, but all of which still has their own disadvantage. The problem mostly lies on finding and identifying the optimal path between two nodes.

The study revealed that Hierarchical Pathfinding A* (HPA*) algorithm has a short execution time which makes it easier and faster to discern the optimal path between two nodes. The way the algorithm works is that it divides the grid map into clusters connects the start node to the edge of a cluster containing it to the adjacent cluster that is closer the end node.

As grid map increases, so does the number of nodes the algorithm must consider. With HPA*, most nodes are removed from the process because a path, not an optimal one, between the start and end was already made during the pre-processing process of the algorithm. By using A* on the path that already exists, the optimal path can be found.

In conclusion, most of the nodes that are considered by another algorithm are not essential and can be removed early on. But this is not always the case, which is why the problem on pathfinding still persists.

Comparison Table of Pathfinding Algorithms

A comparison of different pathfinding algorithms when they are executed on a 64*64 grid. Table 4 shows that out of all the pathfinding algorithms, HPA* has the shortest execution time, least traversed nodes, and one of the algorithms that has the shortest length.

Table 4. 10% blocked node in grid map

| Type | Algorithm | Execution Time (ms) | Traversed Nodes | Length |
|------------|--------------------------|---------------------|-----------------|--------|
| Uninformed | Dijkstra | 1.89 | 496 | 23.36 |
| Uninformed | IDDFS | 9.64 | 423 | 23.36 |
| Uninformed | BIDDFS | 3.67 | 231 | 23.36 |
| Uninformed | BFS(Breadth) | 7.33 | 993 | 23.36 |
| Informed | Greedy Best First Search | 2.2 | 53 | 29.31 |
| Informed | Ida* | 5.232 | 312 | 28.54 |
| Informed | A* | 1.96 | 46 | 23.36 |
| Informed | Jump point search | 1.54 | 312 | 23.36 |
| Informed | HPA* | 1.11 | 36 | 23.36 |

Similar to Table 4, Table 5 shows the performance of different pathfinding algorithms on the same grid but with more blocked nodes. It is shown that HPA* still outclasses other pathfinding algorithms when applied on a grid map.

Table 5. 50% blocked node in grid map

| Type | Algorithm | Execution Time (ms) | Traversed Nodes | Length |
|-------------|--------------------------|----------------------------|------------------------|---------------|
| Uninformed | Dijkstra | 5.808 | 1535 | 16.49 |
| Uninformed | IDDFS | 56.6 | 1631 | 16.49 |
| Uninformed | BIDDFS | 35.41 | 971 | 16.49 |
| Uninformed | BFS(Breadth) | 13.335 | 1521 | 16.49 |
| Informed | Greedy Best First Search | 4.205 | 86 | 21.31 |
| Informed | Ida* | 10.632 | 734 | 20 |
| Informed | A* | 4.016 | 98 | 16.49 |
| Informed | Jump point search | 2.554 | 832 | 16.49 |
| Informed | HPA* | 2.170 | 82 | 16.49 |

Chapter 3

Design and Methodology

| Existing | Proposed |
|--|---|
| Existing Algorithm | Proposed Algorithm |
| <p><small>Problem 1</small></p> <ol style="list-style-type: none"> 1. Divide the map into clusters. These clusters form into a shape of a square or a rectangle. 2. For every set of adjacent clusters, identify an entrance connecting them. | <ol style="list-style-type: none"> 1. Divide the map into clusters. 2. Define entrances between two adjacent clusters. |
| <p><small>Problem 3</small></p> <ol style="list-style-type: none"> 3. For each entrance, define one or two transitions depending on the entrance width. 4. For each transition, define an edge connecting two nodes called inter-edges. 5. For each pair of nodes inside a cluster, define an edge linking them as intra-edges. | <ol style="list-style-type: none"> 3. These nodes that are traversed from each cluster are treated as entrances defined by the HPA* algorithm. 4. Clusters containing non-rectangular static obstacles are marked as “doubtful”. |
| <p><small>Problem 2</small></p> <ol style="list-style-type: none"> 6. Store information on disk that the grid is ready for hierarchical search. 7. The users will insert the start and goal positions on the abstract graph. | <p><small>Solution 1</small></p> <ol style="list-style-type: none"> 5. Return to each doubtful cluster and attempt to mask it with a global navmesh. 6. Store information on disk that the grid is ready for hierarchical search. |

8. Using A* calculations, the algorithm will search for a path between the start and the goal.

a. A* will provide an abstract path that moves the unit from the start to the border of the cluster containing it.

b. From the edge of the cluster before, provide a path that will lead to the cluster that contains the goal.

c. Provides an abstract graph that connects the edge of the goal's cluster to the goal position.

7. The users will insert the start and goal positions on the abstract graph.

8. Using A* calculations and minimal navmesh calculations, the algorithm will search for a path between the start and the goal. Solution 3

a. A* will provide an abstract path that moves the unit from the start to the border of the cluster containing it.

b. From the edge of the cluster before, provide a path that will lead to the cluster that contains the goal.

c. Provides an abstract graph that connects the edge of the goal's cluster to the goal position. Solution 2

9. Use a local navmesh to supplement HPA*'s lack of ability to track dynamic obstacles.

Existing Algorithm**Problem #1**

1) The algorithm only scans rectangular-shaped obstacles.

The table shows the existing algorithm's performance as it scans for the path between the start and the goal nodes. The unit passes through more clusters and traverses through more nodes.

(Existing for Problem 1. See Table 1 on Page 6)

Proposed Algorithm**Objective #1**

1.) To enhance the algorithm and allow it to scan for an optimal path when encountering non-rectangular obstacles on the map by using navmesh in scanning obstacles on the terrain.

In the table, the open list is greater than the existing algorithm's open list because it scans more nodes when calculating for the optimal path. The clusters traversed between the start and the goal are lesser than the existing, this shows that the enhanced algorithm traverses for lesser nodes, and extensively, less weight.

Table 6 Terrain with non-rectangular obstacle applied with navmesh on HPA* pathfinding algorithm

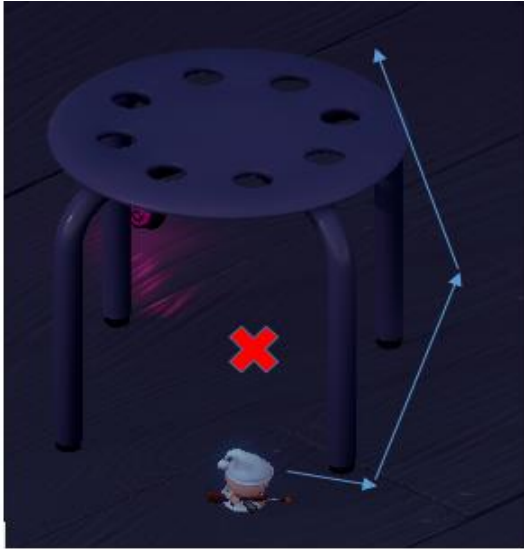
Existing**Computerized Simulation**

Figure 5 Interaction of HPA* algorithm to non-rectangular obstacles.

The algorithm encountered a non-rectangular obstacle but it didn't treat it as one. It proceeded to treat it as a rectangular obstacle, this behavior made it unable for the algorithm to find the path between the obstacle. This made the unit circle around the obstacle instead of passing through it.

| Trial | Open List | Closed List | Clusters | Nodes Passed | Total Weight |
|-------|-----------|-------------|----------|--------------|--------------|
| 1 | 16 | 4 | 2 | 4 | 44 |
| 2 | 27 | 6 | 4 | 6 | 65 |

Proposed

Figure 6 Interaction of Proposed algorithm to non-rectangular obstacles.

In the figure, the unit found the path between the non-rectangular obstacle and made it possible for the unit to pass through.

Existing**Problem #2**

2) Dynamic obstacles prove to be a problem for the algorithm.

Manual Simulation

The table shows the performance of the algorithm when encountering static obstacles.

(Existing for Problem 2. See Table 2 on page 8)

While the following table shows the performance of the algorithm when encountering dynamic obstacles.

(Existing for Problem 2. See Table 3 on page 8)

Proposed**Objective #2**

2.) To help the algorithm compute both static and dynamic obstacles with the least possible work time and used resources by adjusting the pre-process of the algorithm using nav-mesh to calculate for the dynamic obstacles also present on the graph.

Existing Algorithm

Computerized Simulation



Figure 7. Unit interaction to Dynamic obstacles on HPA* Algorithm.

The unit stops moving whenever it encounters a dynamic obstacle.

The algorithm stored an information to the disk that there was no obstacle on the path that it traverses. Encountering an obstacle made it impossible for the unit to find a different path and circle around the obstacle.

Proposed Algorithm



Figure 8. Unit interaction to Dynamic obstacles on Proposed Algorithm.

The unit circles around the dynamic obstacles that it encounters. The enhanced algorithm refreshes the stored information and calculates for every obstacle that the unit may encounter while traversing the area.

Existing**Problem #3**

3) Placement of transition between entrances may deviate the unit from the optimal path.

Manual Simulation

The nodes traversed by a unit differs depending on the transition that it traverses.

Table 7. Unit traverse different transitions

| Trial | Open list | Closed List | Clusters | Nodes Passed | Total Weight |
|-------|-----------|-------------|----------|--------------|--------------|
| 1 | 14 | 12 | 3 | 13 | 194 |
| 2 | 11 | 10 | 3 | 10 | 155 |

Proposed**Objective #3**

3.) To find the best transition place based on the grid map and increasing the optimality of the path taken by the algorithm.

Chapter 4

Results and Discussion

HPA*, as introduced, is a pathfinding algorithm that has a unique process that differentiates itself from different pathfinding algorithm, this process is that it allows the algorithm to scan the grid area before the map loads. Another aspect of the algorithm is that it divides the grid into equally divided areas to make it easier to find the path and traverse between the nodes. With this process, the algorithm can find the optimal path between the nodes, but adding nav-mesh in the process, the algorithm can discern better which are passable or not when scanning obstacles. The following is the process with nav-mesh included:

Process:

1. A user or a computer assigns a grid map for the algorithm to scan.
2. Before the map loads, the algorithm divides the grid into areas that make it easier for the algorithm to discern the obstacles laid out on the grid.
3. Obstacles are then scanned and their respective locations are considered as unpassable.
4. Transitions are placed at the edge of the areas that were previously divided. These transitions are placed to grids that are still passable.
5. A user then input the starting and ending nodes.

6. The algorithm then uses A* to calculate the shortest path needed to take to traverse between the nodes.

7. As the agent (unit) moves along the generated path, a NavMesh is applied to this path. This NavMesh modifies the path and makes it possible for the unit to predict collisions that may be produced by dynamic obstacles (those of which are not included in the initial scan).

What navmesh does is that it scans the shortest path basing from the result of the HPA* pathfinding algorithm, but instead of using nodes, navmesh is used in its place.

Existing**Existing Algorithm**

Problem 1

9. Divide the map into clusters. These clusters form into a shape of a square or a rectangle.

10. For every set of adjacent clusters, identify an entrance connecting them.

Problem 3

11. For each entrance, define one or two transitions depending on the entrance width.

12. For each transition, define an edge connecting two nodes called inter-edges.

13. For each pair of nodes inside a cluster, define an edge linking them as intra-edges.

Problem 2

14. Store information on disk that the grid is ready for hierarchical search.

15. The users will insert the start and goal positions on the abstract graph.

16. Using A* calculations, the algorithm will search for a path between the start and the goal.

Enhanced**Enhanced Algorithm**

10. Divide the map into clusters.

11. Define entrances between two adjacent clusters.

12. These nodes that are traversed from each cluster are treated as entrances defined by the HPA* algorithm.

13. Clusters containing non-rectangular static obstacles are marked as “doubtful”.

Solution 1

14. Return to each doubtful cluster and attempt to mask it with a global navmesh.

15. Store information on disk that the grid is ready for hierarchical search.

16. The users will insert the start and goal positions on the abstract graph.

d. A* will provide an abstract path that moves the unit from the start to the border of the cluster containing it.

e. From the edge of the cluster before, provide a path that will lead to the cluster that contains the goal.

f. Provides an abstract graph that connects the edge of the goal's cluster to the goal position.

17. Using A* calculations and minimal navmesh calculations, the algorithm will search for a path between the start and the goal. ^{Solution 3}

a. A* will provide an abstract path that moves the unit from the start to the border of the cluster containing it.

b. From the edge of the cluster before, provide a path that will lead to the cluster that contains the goal.

c. Provides an abstract graph that connects the edge of the goal's cluster to the goal position. ^{Solution 2}

18. Use a local navmesh to supplement HPA*'s lack of ability to track dynamic obstacles.

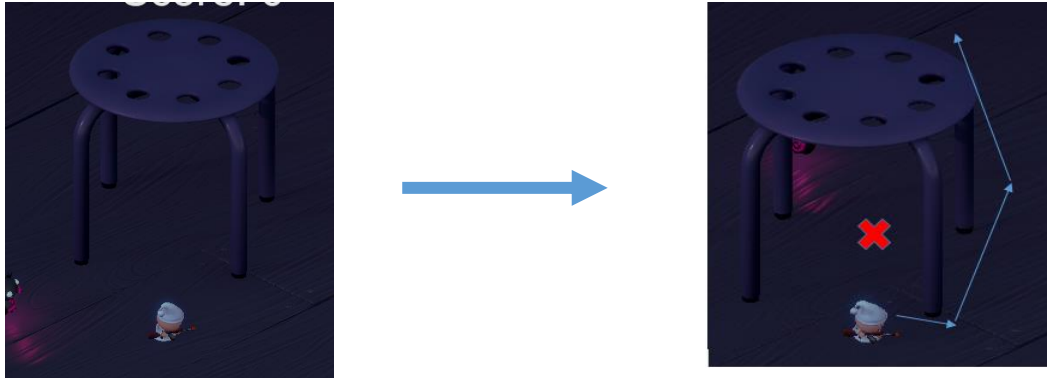


Figure 9. Unit Circles around the Non-Rectangular Obstacle

The unit circles around the obstacle instead of just passing through the area between.

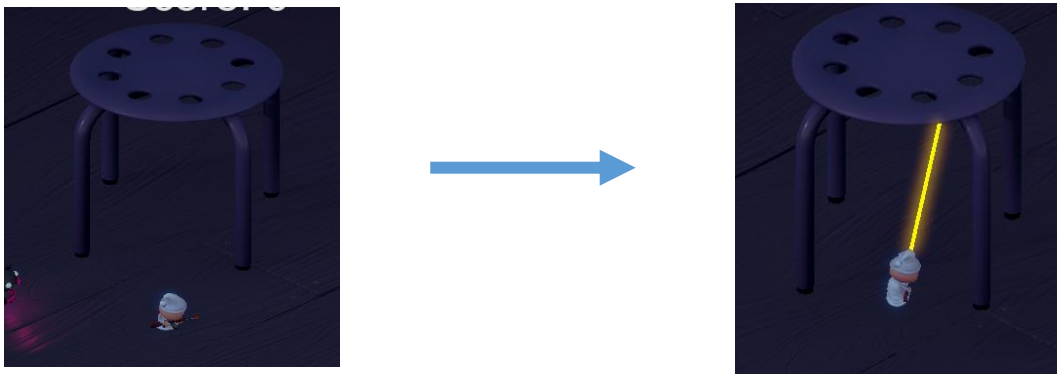


Figure 10. Unit passes through the area between the non-rectangular obstacle.

The unit is now able to find the path between the obstacle and makes the unit able to pass through it.

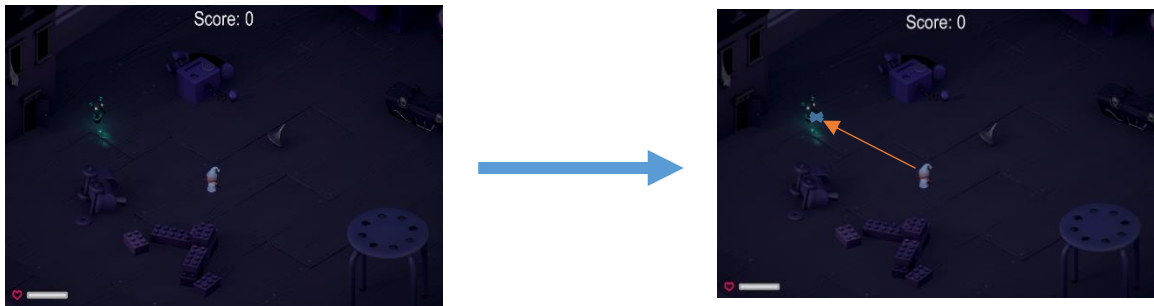


Figure 11. Unit stops when encountering dynamic obstacle

Because the information is stored on a disk once, the unit is given the data that there are no obstacles on that path, encountering an obstacle as it moves there makes the unit stop moving on that spot.

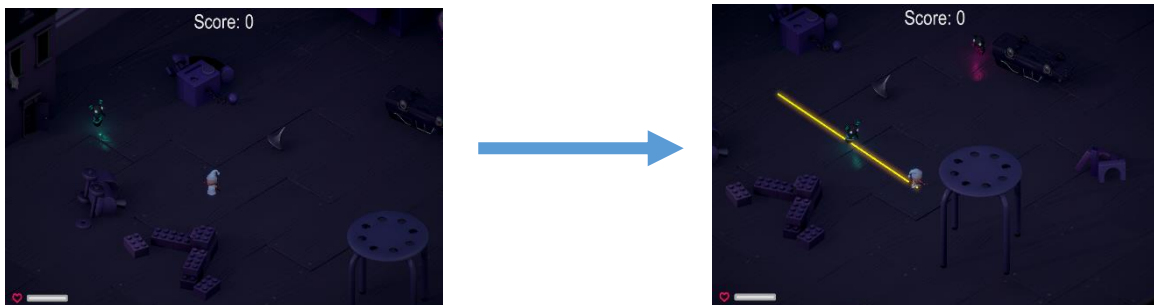


Figure 12. Unit passes towards a dynamic obstacle.

The unit now circles around minimally around the dynamic obstacle and continues on towards its destination.

Chapter 5

Conclusion and Recommendation

Conclusion

The importance of pathfinding algorithms in video games is made more apparent through the study. The ability of the algorithm to discern the optimal path has a great impact on the quality of the game and changes the experience of the player. HPA* may not be as popular as its preceding algorithm, the A*, but it does still have its own recognizable aspect, HPA* algorithm scans the whole area during its pre-processing process, this means that before the application or the game loads, the pathfinding algorithm has already calculated which path to take to traverse between two endpoints. This feature of the HPA* has given it a head start on how to compute for the optimal path.

Recommendation

We recommend using the enhanced algorithm from our research to improve the development of future games. Our enhanced algorithm may be applied on other applications such as way-finding and, extensively, robotics.

Bibliography

L. van Elswijk, (2013) "Hierarchical Pathfinding Theta**"

J. Vernet, (2012) "A Survey of Pathfinding Algorithms Employing Automatic Hierarchical Abstraction", University of Windsor, Ontario, Canada

R. Engman, (2014) "HPA* Used with a Triangulation-Based Graph", Blekinge Institute of Technology, Karlskrona, Sweden

B. Anguelov, (2011) "Video Game Pathfinding and Improvement to Discrete Search on Grid-based Maps", University of Pretoria, Pretoria

A. Strand-Holm Vinther, M. Strand-Holm Vinther, (2015) "Pathfinding in Two-dimensional Worlds", Aarhus University, Aarhus, Denmark

A. Noori, F. Moradi, (2015) "Simulation and Comparison of Efficiency in Pathfinding Algorithms in Games", Department of Computer at Technical and Vocational University, Tehran, Iran.

A. Kring, A. Champandard, N. Samarin, (2012) "DHPA* and SHPA*: Efficient Hierarchical Pathfinding in Dynamic and Static Game Worlds"

Z. Bhatena, (2012) "Near Optimal Hierarchical Pathfinding using Triangulations", University of Texas, Austin, Texas