A* Pathfinding Acceleration With use of Automatically Generated Waypoints for Grid Traversal in a Static Environment

Fredrik Olsson
Blekinge Tekniska Högskola
Karlskrona, Sweden

Magnus Nyqvist
Blekinge Tekniska Högskola
Karlskrona, Sweden

Figure 1: BTH-logo.

ABSTRACT

Pathfinding is a fundamental part of games and it is often supplemented by a waypoint graph to make traversal of a given region easier. The problem is that waypoints are usualy created during the development of the map and are placed manually by leveldesigners. In this paper we propose a method in which we automatically generate waypoints in key locations and accelerate the A* algorithm on tile based grid traversal. The 16200 tests done show that our method of using automatically generated waypoints are faster than using only A* grid traversal on large and complex maps.

KEYWORDS

pathfinding, static environment, games, automaticaly generated, waypoints, A*

ACM Reference Format:
Fredrik Olsson and Magnus Nyqvist. 2019. A* Pathfinding Acceleration With use of Automatically
Generated Waypoints for Grid Traversal in a Static Environment. In *Karlskrona '19: ACM Symposium on Pathfinding, March 03–05, 2019, Karlskrona, SE.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

Karlskrona '19, March 03–05, 2019, Karlskrona, SE

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Pathfinding is a fundamental part of games [3][4] and it is often supplemented by a waypoint graph to make traversal of a given region easier [3]. Every node in a waypoint graph is called a waypoint

and they represent key locations in the region [3]. Each waypoint has edges towards other waypoints to where an object can travel through without risk of colliding with the surroundings [3].

In this paper, we propose a method to reduce execution time of the A* pathfinding algorithm. We improve our previously implemented A* algorithm with automatically generated and connected waypoints in a two-dimensional grid coordinate system. The waypoint generation is done in two steps. First, we generate a waypoint for each corner of an obstacle. Second, we check connections for every waypoint by sending a ray towards all other waypoints in the region. The waypoints are connected if the rays path is unblocked. Our waypoint generation method is heavily influenced by the one suggested in the work of Weiping et al. [3].

Executing pathfinding in dynamic environments is more challenging than in static environments [3], and this study is therefore limited to completely static environments. The difference between the two terms path and shortest path is significant [1]. We conducted studies of several pathfinding combinations, with and without waypoints, but we decided not to measure and compare the time consumption of the shortest path with only A* pathfinding towards the shortest waypoint path. This due to the fact that the maps were so big and the paths so complex that the normal A* grid traversal was not even close to competing at any point with the shortest waypoint path. Differences in execution time of just finding a path is therefore our main focus in the results and we measured times with three different heuristics during the experiments. Different heuristics were used to reduce the chance that our maps did not favor waypoints just due to a poor choice of heuristic.

2 RELATED WORK

A waypoint graph is commonly used to facilitate pathfinding and Weiping et al. [3] suggests a method of how to automatically generate the waypoint graph. Waypoints are generated at every vertex and every waypoint is connected to all other visible waypoints. The connections between waypoints are called edges [3] and the creation of a connection is determined by drawing a line in between the waypoints. If no objects intersect with the line, then it is unblocked and the edge is created. This solution matched our purpose well and our work is therefore heavily inspired by it.

3 METHOD

The method for automatic generation of waypoints were developed with inspiration from the work done in [3]. The method was developed in a two-dimensional environment with rectangles as the only geometry. A maps geometry is determined by an algorithm that divides all blocked regions of the map into rectangular shapes. The algorithm starts by picking the first unselected blocked grid tile and continues by expanding outwards and downwards until it reaches an unblocked tile, or the end of the map. The blocks width is determined by the block depth. A block cannot contain any unblocked tiles and might therefore be adjusted to fit the desired depth rather than the possible width.

Every vertex of every geometry gets a waypoint, offseted by half a tile away from the vertex. If two vertices would create a waypoint in the exact same location then only one would be added to the collection of waypoints. Waypoints that are created outside of the map or in blocked tile locations are not added to the waypoint collection. Edges are then created in between the waypoints as follows. A ray is shot from every waypoint towards all other waypoints. Quadtree traversal is utilized to determine any collision with geometry and an edge is created if a ray reaches another waypoint without an intersection. Each edge gets assigned a cost that is equal to the squared distance between the waypoints.

When the waypoint creation is done the last step of the generation sequence begins. All unblocked tiles gets divided into regions. Each region belongs to a waypoint and each tile gets assigned to the region with the closest visible waypoint. The regions are later used to determine what waypoint to start the traversal from.

When a path is requested, the start and end position gets translated into tiles. The start and end region gets extracted from the tiles before the waypoint traversal begins. To traverse the waypoints a slight modification of the A* algorithm is used. The waypoint algorithm returns a chain of tiles that consist of all the waypoints that needs visit, from start to end. The final path is created by using A* pathfinding to construct short paths in between the tiles of the tile chain and then fusing them together into one long path. By doing so the total computation time of the A* pathfinding algorithm is reduced, at the cost of added overhead to create the tile chain.

The data is sampled bu 57 different tests on three different maps. These maps vary in size and structure. Edgy (Appendix A.1) is the smallest map with 9085 tiles and a blockrate of 46.5%. It has 214 waypoints with a total of 2228 connections. Edgy2 and UMAP2 (Appendix A.2 and Appendix A.3) are both greater in size than Edgy. Edgy2 has 26550 tiles with a blockrate of 48.4%. There are 658 waypoints with a total of 6458 connections. Edgy2 is built of several parts of Edgy repeated. The biggest map is UMAP2 and it is constructed as two tilted U:s facing away from eachother. The map has 35748 tiles and a blockrate of only 8%. UMAP2 has 179 waypoints and 2038 waypoint connections. The choice of three different maps were based upon how well A* performs on small maps compared to bigger ones with complex structures. UMAP2 is an edge case map and has very few blocked tiles, but the way the map is constructed shows how poor A* pathfinding can be on maps with big dead end sections.

Each map was tested with nine different paths and each path had six tests with different heuristics, with and without waypoints. Euclidean Distance [2], Manhattan Distance [2] and Stanford Distance [2], with and without waypoints. Each setting on each path were tested 100 times. The delta time between waypoint traversal and A* tile traversal performance is the avarage of the total time of each test. The paths of choice were not biased, they were chosen to favor both pathfinding methods. A few paths were chosen to be completely open to demonstrate the strength of A* in open areas, but also paths that had wall sections in between the start and the end to demonstrate the waypoint traversals strengths.

The limitation was made to only compare traversal time and not take into account the loading time for the map, memory consumption nor dynamic environments. After the experiments the results for the best waypoint path were removed, because there was no data to compare it to. The experiments does not take into account if the paths look natural to the viewer, as it is a subjective question. The research goal was to create an algorithm that could compute paths faster than the raw A* algorithm and the solution propesed

proves that the waypoint generation gives a pathfinding advantage on bigger maps. The waypoint traversal scales much better then the A* pathfinding algorithm does. Also, waypoint traversal works excelent in edge cases, where A* completely fails to keep low compute times. It is however hard to motivate waypoint traversal on smaller maps, due to the added overhead.

4 RESULTS

The results show that traversing the grid with the automatically generated waypoint approach, is more scalable than using the raw A* algorithm. This also holds true when changing the heuristics for the traversing algorithms. In the smallest map, Edgy (Appendix A.1),

Path	ED Delta	MH Delta	SD Delta
P0	0,561612	4,115806	4,307822
P1	-0,158349	1,111369	1,057934
P2	-0,055877	-0,066283	-0,057953
P3	-0,153236	-0,210622	-0,20186
P4	-0,055854	-1,120068	1,492422
P5	-0,075147	-0,075057	-0,066367
P6	-0,031801	1,573965	1,908289
P7	-0,160622	-0,957571	-0,753667
P8	-0,119794	-0,538933	-0,502129

Figure 2: Edgy Time Table
Total time gain using waypoints in milliseconds.

when the Euclidean Distance heuristic is used, it is cheaper to simply use raw A* directly on the grid to generate the path due to the added overhead when using waypoints. The other heuristics show no significant gain with waypoints added to the map, except the P0-path as seen in Figure 2, but this path is a worst case scenario.

While traversing the maps Edgy2 (Appendix A.2) and UMAP2 (Appendix A.3), which are greater in size compared to Edgy, the results shows that waypoint traversal is superior to raw A* pathfinding. On almost all the paths tested, to traverse the waypoints first and direct the A*, gets a much smaller total time spent calculating the path. This proves that the waypoint traversal added are improving the total time spent calculating the path. Though the results show that in some cases using waypoints, the total time is increased compared with raw A*. This is because these paths are short or straight, with no or few blocked tiles in the way of the pathing.

In the comparison diagrams the color green, purple and red boxes represents traversal with waypoints with the Euclidian Distance , Manhattan Distance and Standford distance heuristic. The color blue, cyan and brown colors represent the raw A* tile traversal with the Euclidian Distance, Manhattan Distance and Standford distance heuristic. The X-axis is the nine different paths and the Y-axis is time in milliseconds spent calculating the path. On the map Edgy,

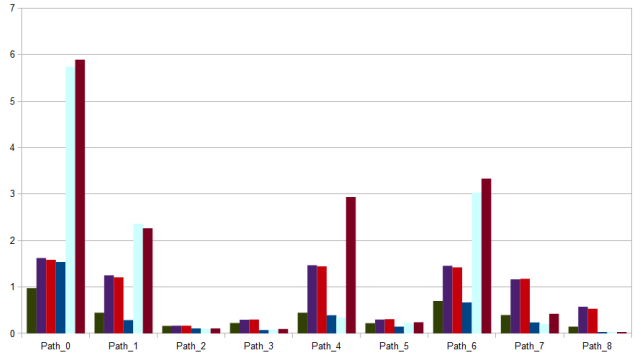


Figure 3: Edgy Compare Diagram

in Figure 3, all the tests are compared with each other. Its easy to see that in most cases, using only tile traversal is better than using the added waypoint traversal. The average gain time can be seen in Figure 2 and in almost all cases the total time increases when traversing with waypoints.

However it is clearly displayed that waypoint traversal is superior to raw A* when the map size and complexity increases. Both Figure 4 and Figure 5 prove that the gain increases as the map grows. Waypoint traversal was up to and above 40 milliseconds faster then raw A* pathfinding on UMAP2 and this is due to the maps edge case layout. In most cases the raw A* would get stuck in one of the U-shapes while searching for the goal, whereas the added waypoint traversal would quickly guide the A* to exit the rabbit hole and continue in the right direction. The exact avarge gain can be seen in Figure 6 and Figure 7.

All the experiments and tests made, in wich the results are used have a validity P value way below 0.05, mostly closely to 0.00.

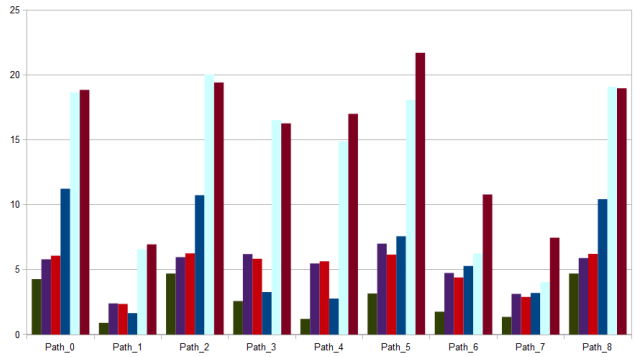


Figure 4: Edgy2 Compare Diagram

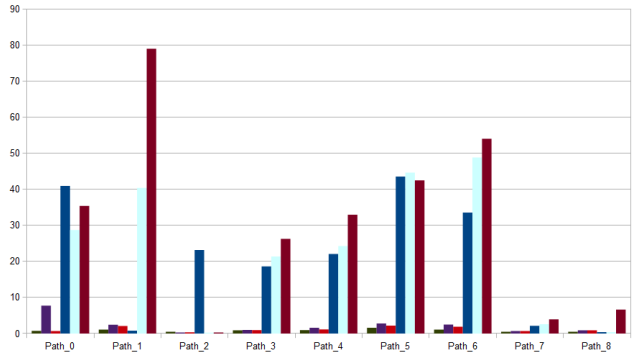


Figure 5: UMAP2 Compare Diagram

Path	ED Delta	MH Delta	SD Delta
P0	6,964149	12,862133	12,78077
P1	0,758243	4,178615	4,582088
P2	6,033441	14,063203	13,157186
P3	0,686925	10,322416	10,427871
P4	1,568094	9,408417	11,358409
P5	4,412479	11,073896	15,542987
P6	3,531597	1,501702	6,399459
P7	1,84863	0,923159	4,570182
P8	5,732686	13,197459	12,765844

Figure 6: Edgy2 Time Table
Total time gain using waypoints in milliseconds.

5 CONCLUSION AND FUTURE WORK

On smaller maps with a lot of waypoint connections, raw A* is quicker than with the generation of waypoints. But when the map starts to scale, waypoints are significantly better than just A* by itself. Our results are based on the first path the algorithm finds, not the shortest. In all our tests we also measured the time it took when using the shortest path for the waypoint traversal and this were proven to take more time than using raw A* with the first path found. It was only on UMAP2 we could get the shortest path for waypoints and still get less calculation time then just raw A*, but it were still not real time friendly. We got this result due to the real edge case layout of the map.

In the way the waypoints are generated there is no way of controlling how many there will be, it all depends on the complexity of the map. In future work there could be another approach of finding the waypoints based on the complexity of the map so that the average gain would always be positive.

Path	ED Delta	MH Delta	SD Delta
P0	40,18396	20,98714	34,716981
P1	-0,304999	37,960368	76,931817
P2	22,676029	-0,056768	-0,037632
P3	17,772602	20,339802	25,31222
P4	21,121472	22,660484	31,813428
P5	41,919359	41,868362	40,338759
P6	32,468788	46,34481	52,146204
P7	1,663424	1,908057	3,255747
P8	-0,11025	-0,515442	5,743651

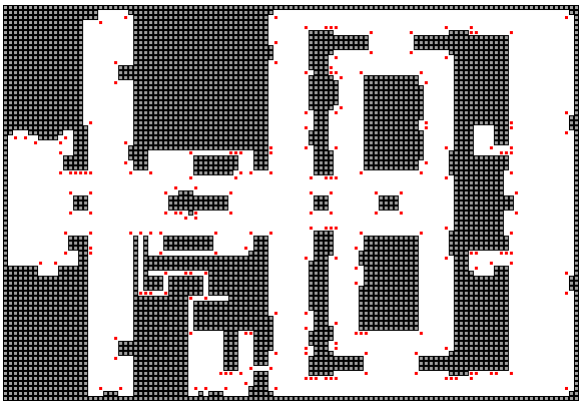
Figure 7: UMAP2 Time Table
Total time gain using waypoints in milliseconds.

REFERENCES

[1] Geethu Elizebeth Mathew, *Direction Based Heuristic For Pathfinding In Video Games*, Procedia Computer Science, vol. 47, pp. 262-271, 2015.
[2] Patel, Amit, *Amit's Thoughts on Pathfinding*, 2019-02-22, http://theory.stanford.edu/ amitp/GameProgramming/ Heuristics.html, (Accessed 2019-03-20).
[3] Weiping Zhu, Daoyuan Jia, Hongyu Wan, Tuo Yang, Cheng Hu, Kechen Qin, Xiaohui Cui, *Waypoint Graph Based Fast Pathfinding in Dynamic Environment*, International Journal of Distributed Sensor Networks, vol. 2015, Article ID 238727, 12 pages, August 2015. https://doi.org/10.1155/2015/238727.
[4] Zeyad Abd Algfoor, Mohd Shahrizal Sunar, Hoshang Kolivand, *A Comprehensive Study on Pathfinding Techniques for Robotics and Video Games*, International Journal of Computer Games Technology, vol. 2015, Article ID 736138, 11 pages, 2015. https://doi.org/10.1155/2015/736138.

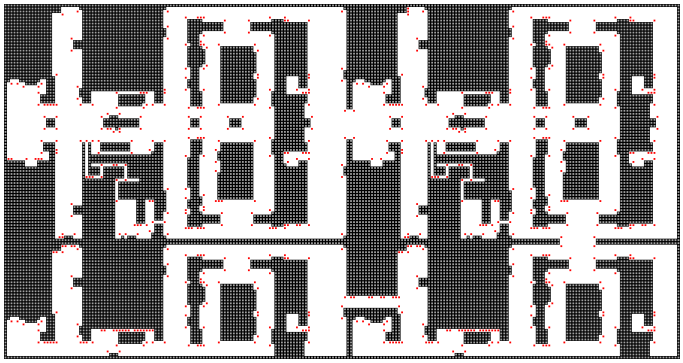
A MAP LAYOUTS

A.1 Map Edgy



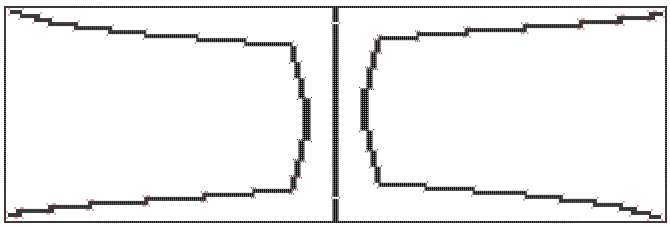
The image shows the map Edgy that consist of 9085 tiles, 4226 blocked (46.5%) and 214 waypoints with 2228 connections. Gray boxes indicates blocked tiles, red boxes indicates waypoints and white space are open tiles.

A.2 Map Edgy2



The image shows the map Edgy2 that consist of 26550 tiles, 12854 blocked (48.4%) and 658 waypoints with 6458 connections. Gray boxes indicates blocked tiles, red boxes indicates waypoints and white space are open tiles.

A.3 Map UMAP2



The image shows the map UMAP2 that consist of 35748 tiles, 2890 blocked (8%) and 179 waypoints with 2038 connections. Gray boxes indicates blocked tiles, red boxes indicates waypoints and white space are open tiles.