CREATIVE MATH. & INF.
16 (2007), 135 - 142

Dedicated to Professor Ioan A. RUS on the occasion of his 70^{th} anniversary

Comparison of 3 implementations of the A* algorithm

LIVIA SÂNGEORZAN, KINGA KISS-IAKAB and MARICICA SÎRBU

ABSTRACT. This paper deals with a short presentation of the A* algorithm and the comparison of three different implementations of the algorithm. A* is a graph search algorithm that finds a path from a given initial node to a given goal node (or one passing a given goal test). It employs a "heuristic estimate" that ranks each node by an estimate of the best route that goes through that node. It visits the nodes in order of this heuristic estimate. The A* algorithm is therefore an example of best-first search, which optimizes the depth-first search.
What sets A* apart from best-first search is that it also takes the distance already travelled into account. This makes A* complete and optimal, i.e., A* will always find the shortest route if any exists. It is not guaranteed to perform better than simpler search algorithms. In a maze-like environment, the only way to reach the goal might be to first travel one way (away from the goal) and eventually turn around. In this case trying nodes closer to your destination first may cost you time. In addition to finding a path for a unit to move along, pathfinding can be used for several other purposes: exploration, spying, road building, terrain analysis, city building, puzzle solving. So the main application domain is game design.

1. SUMMARY OF THE A* ALGORITHM

In general, the A* algorithm is used in most games to realise the way finding of game characters. For this purpose, the whole game world is divided into little squares, and for each of these a value is stored, which defines whether the square is easy - more difficult - or impossible to be walked on.
The key to determining which squares to use when figuring out the path is the following equation:

$$F = G + H \tag{1.1}$$

where

(1) G = the movement cost to move from the starting point A to a given square on the grid, following the path generated to get there.
(2) H = the estimated movement cost to move from that given square on the grid to the final destination, point B. This is often referred to as the heuristic, the reason why it is called that is because it is a guess. We really do not know the actual distance until we find the path, because all sorts of things can be in the way (walls, hills, water, etc.). There are many ways to calculate H depending on the problem to solve.

Our path is generated by repeatedly going through our open list and choosing the square with the lowest F score.

(1) Add the starting square (or node) to the open list.

Received: 10.10.2006. In revised form: 21.02.2007
2000 *Mathematics Subject Classification.* 68T20, 91A90.
Key words and phrases. *Search algorithm, heuristics, game design.*

(2) Repeat the following:
  (a) Look for the lowest F cost square on the open list. We refer to this as the current square.
  (b) Switch it to the closed list.
  (c) For each of the 8 squares adjacent to this current square...
    (i) If it is not walkable or if it is on the closed list, ignore it. Otherwise do the following.
    (ii) If it is not on the open list, add it to the open list. Make the current square the parent of this square. Record the F, G, and H costs of the square.
    (iii) If it is on the open list already, check to see if this path to that square is better, using G cost as the measure. A lower G cost means that this is a better path. If so, change the parent of the square to the current square, and recalculate the G and F scores of the square. If we are keeping our open list sorted by F score, we may need to resort the list to account for the change.
  (d) Stop when we:
    (i) Add the target square to the closed list, in which case the path has been found, or
    (ii) Fail to find the target square, and the open list is empty. In this case, there is no path.
(3) Save the path. Working backwards from the target square, go from each square to its parent square until we reach the starting square. That is your path.

2. Examples of different A* search versions

Let us consider a map first on a grid of 20x20 cells and then on a grid of 40x40 cells. The cells with the highest difficulty to walk on are colored black, the cells with lower difficulty to walk on are colored different shades of gray, the cells easier to travel than the normal cells are colored orange. The cells without obstacles remain unchanged. We have applied on this map three different versions of A* search, every version on a rough grid and and on a fine grid. The starting cell is colored green and the endig cell is colored red. We can see a different behaviour of the three versions of the A* search depending on the map and the fineness of the grid.

2.1. **A* without heuristics.**

A* search without heuristics is equivalent to Dijkstra's algorithm. If the map on which we apply the algorithm is not large and the cost of the cells are not very different the A* search without heuristics shows good results meaning that it finds the shortest path relative fast. But in case of large and complicated maps the other versions of A* search have a better behaviour.

Example 2.1. hills.grd - 20x20 map

This map is on a grid of 20x20 cells with obstacles of different levels of difficulty. The starting point is in the left lower corner of the map and the ending point is in the right upper corner.

If we apply the A* search without heuristics (equivalent to Dijsktra's algorithm) we can observe that the whole map is explored between the starting point and the ending point. The determined path has the cost 43.

Figure 2.1

Example 2.2. hills.grd - 40x40 map
This example is the previous map on a grid of 40x40 cells.
If we apply the A* search without heuristics (equivalent to Dijkstra's algorithm) on the same map on a grid of 40x40 cells we observe that almost the whole map is explored with the exception of the left upper corner, right upper corner and right lower corner. This search determines another path with the cost 39 since the map is larger and finer.

Figure 2.2

2.2. **Classic A*.**
Classic A* search uses as heuristic function the Manhattan distance (between a node and the goal):

$$H(node) = D * (abs(node.x - goal.x) + abs(node.y - goal.y)) \tag{2.2}$$

where D is the minimum cost for moving from one space to an adjacent space.
The behaviour of this version of A* search is better than the A* search without heuristics, the runtime is better, the determined paths are indeed the shortest ones. But the path does not always look ”natural”.

Example 2.3. hills.grd - 20x20 map
This map is on a grid of 20x20 cells with obstacles of different levels of difficulty. The starting point is in the left lower corner of the map and the ending point is in the right upper corner.
If we apply classic A* search (with heuristics) we observe that it does not explore the whole area between the starting point and the ending point. The left upper corner and right lower corner is not explored anymore because classic A* sees that the path going through these regions would not be a short path. The determined path is the same as in case of A* search without heuristics and the cost is 43.

Figure 2.3

Example 2.4. hills.grd - 40x40 map
This example is the previous map on a grid of 40x40 cells.
If we apply classic A* search (with heuristics) we observe that it does not explore the whole area between the starting point and the ending point but it explores less in the right area of the map. The middle area of the map where are no obstacles of high difficulty level is not explored since classic A* sees that the path going through this area would not be a short one. This version explores less than in the case of A* search without heuristics on a grid of 40x40 cells. The determined path is different from the path in the previous case because the map is larger and the

algorithm determines a shorter path. The cost of this path is 39. The runtime is obviously higher.

Figure 2.4

2.3. **A* with modified heuristics.**
A* search with modified heuristics uses as heuristic function a vector cross-product for a better functioning in extreme cases. A way to determine "natural" looking path is to prefer paths that are along the straight line from the starting point to the goal:

$$dx1 = node.x - goal.x \tag{2.3}$$
$$dy1 = node.y - goal.y \tag{2.4}$$
$$dx2 = start.x - goal.x \tag{2.5}$$
$$dy2 = start.y - goal.y \tag{2.6}$$
$$cross = abs(dx1 * dy2 - dx2 * dy1) \tag{2.7}$$
$$H(node) += cross * 0.001 \tag{2.8}$$

This code computes the vector cross-product between the start to goal vector and the current point to goal vector. When these vectors do not line up, the cross-product will be larger. The result is that this code will give some slight preference to a path that lies along the straight line path from the start to the goal. When there are no obstacles, A* not only explores less of the map, the path looks very nice as well.
The behaviour of this version is better than the behaviour of A* search without heuristics and classic A*, the runtime is better, the determined paths are really short and the paths look more "natural".

Example 2.5. hills.grd - 20x20 map
This map is on a grid of 20x20 cells with obstacles of different levels of difficulty. The starting point is in the left lower corner of the map and the ending point is in the right upper corner.

If we apply A* search with modified heuristics then the difference to classic A* (with heuristics) on this map is not very significant. But we observe however that the left upper corner is explored less than in the previous case. The determined path is the same as in the previous case and the cost is 43.

Figure 2.5

Example 2.6. hills.grd - 30x30 map
This example is the previous map on a grid of 40x40 cells.
If we apply A* search with modified heuristics then the difference to classic A* (with heuristics) on this map is also not very significant. But we observe however that in the right lower corner the algorithm explores less than in the previous case. The determined path is not the previous path but is the same as in the case of classic A* (with heuristics) and the cost is 39.

Figure 2.6

2.4. **Runtimes.**
We have measured the runtimes for the three example maps from above, ran on our implementation of the three versions of A* algorithm. The results are contained in Table 2.1.

Average runtime						
	20x20 map			40x40 map		
	A* without heuristics	Classic A*	A* with modified heuristics	A* without heuristics	Classic A*	A* with modified heuristics
corners.grd	243	73	64	6039	878	71
hills.grd	123	129	72	3511	163	91
fails.grd	66	63	55	184	56	55

Table 2.1

We can observe that from the point of view of the runtime the best performances has A* search with modified heuristics then comes classic A* and finally A* search without heuristics. Important to notice is that A* search without heuristics cannot be taken in consideration when choosing a search algorithm for a game, for example. In the case of a practical map the runtime performances of the A* search without heuristics are almost 10 times weaker than the performances of other versions of A* algorithm.

3. CONCLUSIONS

The examples from Section 2 demonstrate that the best behaviour has A* search with modified heuristics. It determines the shortest path between the starting point and the ending point with minimal exploration of the map. And the average runtime is always the lowest.

In the case of a large map with different surfaces we can use larger cells on the grid. This consideration reduces the total number of verified nodes for the determination of the path. We can build even two search systems of the path which are used in different situations depending on the length of the path (two-tired pathfinding, multi-tired pathfinding). We can use large cells for a long path or we can switch to finer search using smaller cells when we are getting close to the target.

Depending on the game, collision between game units can be admissible or not. If we don not take in consideration collision avoidance then game units can pass right through each other. For adjacent units that are moving, we can discourage collisions by penalizing nodes that lie along their respective paths, thereby encouraging the pathfinding unit to find an alternate route.

While A* will automatically give us the shortest, lowest cost path, it would not automatically give us the smoothest looking path. While we are calculating the path we could penalize nodes where there is a change of direction, adding a penalty to their G scores. We have implemented this in A* search with modified heuristics with the help of the cross-product.

While A* is generally considered to be the best pathfinding algorithm, there is at least one other algorithm that has its uses Dijkstra's algorithm. Dijkstra's algorithm is essentially the same as A*, except there is no heuristic, H is always 0. Because

it has no heuristic, it searches by expanding out equally in every direction. So Dijkstra's usually ends up exploring a much larger area before the target is found. This generally makes it slower than A*.

REFERENCES

[1] Brackeen D., Barker B. and Vanhelsuwé L., *Developing Games in Java*, New Riders Publishing, 2004
[2] http://www.policyalmanac.org/games/aStarTutorial.htm
[3] http://www-cs-students.stanford.edu/∼amitp/Articles/AStar2.html
[4] http://www.generation5.org/content/2000/astar.asp

TRANSILVANIA UNIVERSITY OF BRASOV
DEPARTMENT OF MATHEMATICS AND COMPUTER SCIENCE
IULIU MANIU 50
500091 BRASOV, ROMANIA
E-mail address: livia.sangeorzan@gmail.com
E-mail address: kissjakab@gmail.com
E-mail address: sv.mari@gmail.com