From Point A to Point B: Realistic Pathfinding in Games

Franklin Hartman - Ethan Marshall

December 13, 2018

1 Introduction

Frequently in modern games there arises a point where a non-player character (NPC) must navigate from its current position to an objective. As graphics continue to get more realistic, it becomes increasingly desirable for these behaviors to increase in realism as well. Thus, determining which algorithm most realistically moves a NPC from one point to another is of utmost importance to modern game developers. Furthermore, due to the reality of computational limitations in modern games, quantifying and comparing algorithmic efficiency is key as well.

The focus of this project is therefore to assess the organic authenticity and computational efficiency of three different pathfinding algorithms. The three algorithms to be discussed include Dijkstra's algorithm, A* Search, and a smooth-path variant of A* (referred to as Smoothed A* Search).

It was hypothesized that Dijkstra would be both the least realistic and efficient, A* would exhibit comparable realism but be the most efficient, and Smoothed A* would be the most realistic and have an efficiency in between the other two algorithms. Although the hypothesis was proven for the relative efficiency of each algorithm, it was concluded that the relative realism varied per environment. A detailed discussion of the scientific methods and data used to derive this conclusion follows. However, a description of the problem and each algorithm in this context is necessary first for a full understanding.

2 Problem Description

In order to assess the three pathfinding algorithms, the problem must include an environment, actor (also referred to as an agent), start position, obstacle (at least one), and end goal. In this context, an environment refers to the space in which the actor, start position, obstacle(s), and end goal exist within. Although there are numerous ways this can be represented, the specific implementation chosen for this paper will be a two-dimensional grid. In this grid, each "square" can be thought of as a node connected to all adjacent squares. Thus, this implementation allows for a natural extension of graph search algorithms, as it can be represented using a node-based graph. For this reason, a graph implementation was utilized instead of other data structures.

Within this environment, the actor is the artificially intelligent agent which navigates from the start position to the end goal. Using a specific pathfinding algorithm, the actor will first analyze the environment, then decide the optimal path to take. Although actors in video games are typically given visual representation (e.g. a human model) to aid in realism, the focus of this paper is the path taken, not the actor's appearance.

Because the simplest path between two points is always a straight line, the environment must also include obstacles to challenge the actor. The obstacles are nodes within the graph which the actor must avoid when traversing from the start point to end goal.

See Figure 1 for a simple representation of this problem. The path of the agent from starting node A to goal node B is represented by the sequence of red arrows, and the obstacles in the environment are represented with tan colored squares.

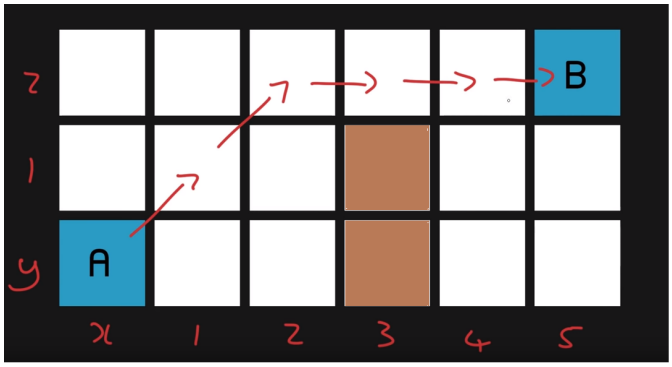


Figure 1: Simple Pathfinding Environment Example (Modified from YouTuber Sebastian Lague)

3 Algorithms

Search algorithms are required so that the actor may find an optimal path from the start point to the end goal. The first of which to be analyzed in this paper is Djikstra's algorithm.

3.1 Dijkstra's Algorithm

Dijkstra's algorithm, similar to the other two algorithms, can be used to find the shortest path between two specified nodes in a graph (although technically it is intended for finding the shortest path from a specified node to every other node). The algorithm begins by marking the path cost from the starting node to all others with infinity. Then, beginning at the starting point, the algorithm visits the node with the smallest path cost. At each iteration, all nodes directly connected to the current will have their path costs updated if the current cost is less

than what has been previously recorded. This continues until the desired end node is visited.

At this point, the shortest path distance from start to goal is found. In order to retrieve this path, the algorithm stores the parent of each visited node. Thus, starting from the goal node, the list of parents is compiled until the start node is reached.

It should be noted here that in this project's graph implementation, all path costs are equal. This is done to allow for more fine-grain movement than a typical graph abstraction of an environment. However, this means that the path cost from start node to goal node is determined by the number of nodes the path travels through.

3.2 A* Search Algorithm

The A* Search algorithm is another algorithm which can determine the shortest path between two nodes in a graph, although it is more commonly used in games than Dijkstra's algorithm. This is because it includes a heuristic function, making it typically more efficient at locating the shortest path from start to goal.

In A* Search, the heuristic function can be thought of as a straight line distance from a specified node to the goal node. Although obstacles within the environment prevent this straight line distance from being the accurate optimal path 100% of the time, it is a useful estimation when deciding which node will give the shortest path to the goal. Therefore, A* augments Dijkstra's algorithm by selecting the next node to visit based on both the path cost to that node and the estimated path cost from that node to the goal.

3.3 Smoothed A* Search Algorithm

One major disadvantage of both A* Search and Dijkstra's algorithm is that they will always follow paths in a straight line for as long as possible, making abrupt angular changes in direction when necessary. However, there exists a clever workaround to this issue which leads to more smooth, organic results[4].

In order to conceptualize this variant of A* Search, consider a simplification of paths in which all traversals in a straight line are combined. Thus, the only nodes of importance are those in which a turn (i.e. change in angle) occurs.

Since the algorithm has determined the path before the agent traverses it, it is able to anticipate these turning points. Therefore, the algorithm can begin to adjust the path towards the next turning point before the current turning point is reached. Functionally, the actual smoothing calculation is computed "live" using lerps (i.e. linear interpolations between two points) as the actor traverses the path. This creates a more natural curve, rather than a harsh rotation. See Figure 2 for a visual representation of this.

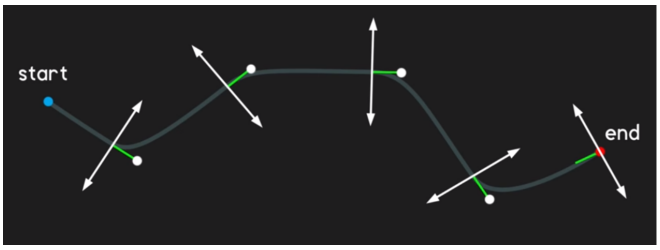


Figure 2: Smoothed A* Search Algorithm Example (From YouTuber Sebastian Lague)

4 Methods

In order to assess both the relative realism and efficiency of Dijkstra's algorithm, A* Search, and Smoothed A* Search, the project ran each algorithm on the exact same "environment" with the same start position and end goal. This was performed across three trials of increasing environment size and complexity. As stated in the Problem Description section, grids were used to create the environments. The three environments, start points, goal points, and obstacles were constructed as follows:

1. The first environment was small and included only a single obstacle between the starting point and the goal.
2. The second environment was larger and included three obstacles between the starting point and the goal.
3. The third environment was the largest with many obstacles between the starting point and the goal (akin to traversing a maze).

Figure 3 shows a visual representation of these three environments. In each environment, the small white dot on the left represents the start position, the larger white dot on the right represents the goal position, the green plane represents the "walkable" terrain, and the red rectangles represent obstacles. Although each environment is represented as the same size in this figure, environment 2 is five times the size of environment 1, and environment 3 is two times the size of environment 2.

(Environment 1) (Environment 2) (Environment 3)

Figure 3: The three environments used in analysis

At each trial, the realism of each algorithm was assessed. This was performed qualitatively. Because the goal in realistic pathfinding is to mimic the behavior of an

organically intelligent agent, asking the opinion of real life persons was a logical way to evaluate the realism of the three algorithms. Therefore, this assessment entailed a survey of 7 persons. Each participant was asked to choose which computationally chosen path represented the most realistic pathfinding behavior in each environment. The summation of the preferences in each environment was then calculated and used for analysis. Furthermore, the author's own opinion on the relative realism of each algorithm will be discussed as well.

Beyond the experience of realism that pathfinding algorithms can achieve, there are technical limitations to most game systems which must be considered. Since pathfinding is a frequent operation in games, it must be able to be calculated quickly. For this reason, the time each algorithm takes to find a path was recorded at each trial. Furthermore, modern game maps continue to increase in size which makes memory usage a concern when calculating long paths across complex terrain. Therefore, the amount of memory each algorithm occupies at each trial was also recorded. While the amount of time to run each algorithm was recorded in seconds, the amount of memory was recorded by counting the amount of nodes produced.

By utilizing these experimental methods, the relative efficiency and realism of each algorithm was determined. The discussion of the data which led to these conclusions follows in the next section.

5 Results

The experimental aspects to be analyzed across the algorithms include the qualitative realism, time complexity, and memory usage.

5.1 Qualitative Realism

In order to gain a deeper insight into the relative realism of each algorithm in each environment, the results of the aforementioned survey were summed across the individual algorithms per environment. A graphical representation of this can be seen in figure 4.

The first trend of interest apparent in figure 4 is the increase of participant preference in Smoothed A* Search. As the data suggest, the participants began to see the path derived from Smoothed A* as more realistic when the environment size increased. Viewing how Smoothed A* Search performed across the three environments in figure 5 helps to explain why this trend exists.

Viewing the chosen path in environment 1, the general opinion of the participants was that the curvature was exaggerated more than would be realistic. The common consensus was that persons would choose to walk closer to the wall than the algorithm's path because it would simply be more efficient.

However, as the environment size increased, the curvature inherit to Smoothed A* Search began to become

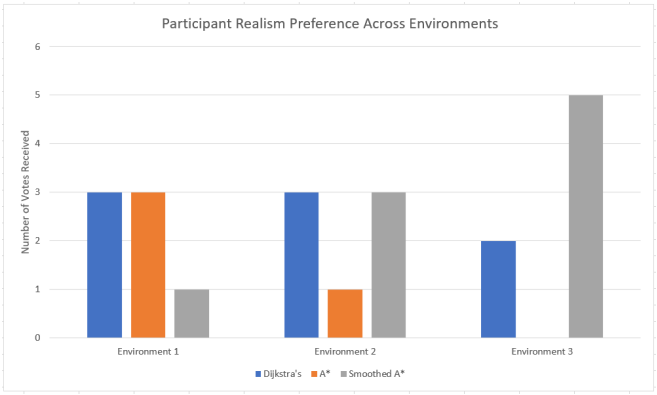
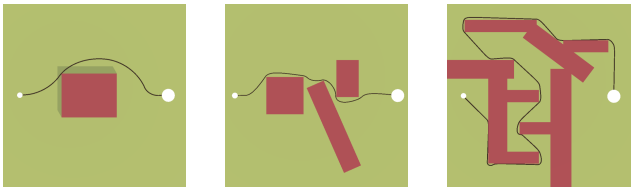


Figure 4: Summation of Participant Preferences in Realism

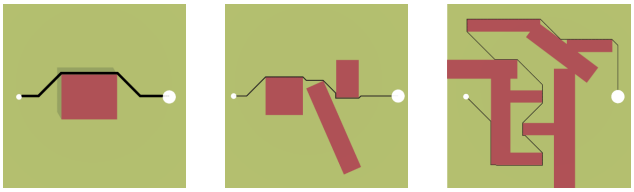


(Environment 1) (Environment 2) (Environment 3)

Figure 5: Smoothed A* Paths Across Environments 1-3

less pronounced. This is because the variable which represents the radius of the curvature (referred to as "turn distance") was held constant across each environment. As the smoothing of Smoothed A* became more subtle, participants began to recognize it as being the most realistic path. This would suggest that path realism is not necessarily determined by the binary existence of smooth turns, but is dependent on the degree of which the turns are smoothed as well.

In order to contrast the increasing realism with environment size that Smoothed A* exhibited, a comparison with regular A* Search is necessary. This is because participants generally saw the paths chosen by this algorithm as less realistic for larger environment sizes. Figure 6 shows the paths determined by A* for environments 1-3.



(Environment 1) (Environment 2) (Environment 3)

Figure 6: A* Paths Across Environments 1-3

Comparing the chosen environment 1 path between Smoothed A* and A*, it becomes even more apparent why A* was the preferred choice. Although A* does exhibit harsh angular changes for each turn, it more closely represents how a human would traverse this environment

than Smoothed A*. This is likely the case because a person would still attempt to traverse this environment with reasonable efficiency, and would therefore stick closer to the wall rather than walking further out unnecessarily.

The difference between the Smoothed A* and A* paths in environment 3 is less pronounced, however. As mentioned previously, this is because the turn distance was held constant across environments for Smoothed A*. Therefore, it is not necessarily the case that A* performed less realistically as the environment size and complexity grew, but it is more likely the case that Smoothed A* became comparatively more realistic. In environment 3, Smoothed A* shows slight rounding of turns that is not possible with regular A*. This slight imperfection more closely mimics human behavior than the tight turns A* executes.

Despite the results of the survey, it is the author's opinion that environment 2 exhibits the most realistic behavior for Smoothed A* Search. This is due to two reasons: The first reason is that the medium environment size seems to be the "sweet spot" for the smoothing radius which occurs at each turn. In environment 1, this radius is too large. In environment 3, the radius is perhaps too small to provide a significant difference from regular A*. The second reason is that the the decreased environment complexity in environment 2 (when compared to environment 3) allows the path to contain less "wall hugging" and less purely straight lines. Because there are only three obstacles in the way and the environment size is relatively small, there are no larger open spaces in which the algorithm will travel perfectly straight. Thus, all turns are congregated in a smaller area, allowing for more smoothing to take place. These two reasons combine to create an imperfect and realistic path.

It should be noted that the paths found by Dijkstra's algorithm were largely similar to those found by A* Search, and therefore it would be redundant to discuss the Dijkstra paths in this context. Although survey participants did choose Dijkstra over A* on average, they frequently expressed difficulty choosing between the two.

Furthermore, it should also be noted that the A* paths were traced over in Adobe Illustrator. This was in order to provide greater visual consistency, and did not change anything about the determined paths besides the aesthetics of the lines.

5.2 Time Complexity

Because pathfinding is a common operation within games, it must be performed quickly in both small, simple environments and large, complex environments. Figure 7 shows the runtime of each algorithm in the three environments.

This figure illustrates that although Dijkstra's algorithm always runs slower than both A* and Smoothed A*, the difference in run time becomes less as environment size and complexity increases. Although this trend

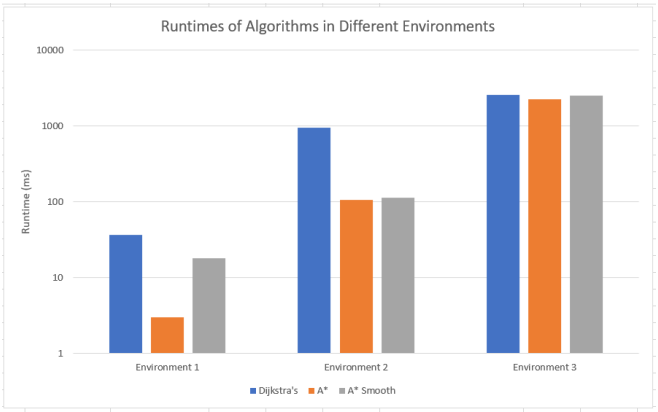


Figure 7: Algorithm Run Times Across Environments

is somewhat emphasized by logarithmic scale of the y-axis, it is worth noting that using Dijkstra's algorithm instead of A* Search is a viable choice in large complicated environments (although it would still not be preferable). As mentioned in section 3.2, the reason Dijkstra's algorithm typically performs worse than A* Search is because it lacks a heuristic function. Without this, the algorithm does not weight choices in the direction of the target over choices in the opposite direction.

It is possible that Dijkstra began to run quicker relative to A* and Smoothed A* because the increased environment complexity limited the amount of possible paths. As the amount of obstacles in an environment increases, freedom of movement decreases. Therefore, even though Dijkstra searches for paths more naively than A*, the size of the search space is inversely proportional to the amount of obstacles in the environment. Furthermore, the heuristic function in A* is less useful when multiple obstacles stand between the current node and the destination node.

Figure 7 also shows that Smoothed A* performs relatively better with respect to run time as environment size and complexity increases. This is because the calculations involved in simplifying the path down to only the turn points (which is necessary to smoothing the path, as explained in section 3.3) are far cheaper than the calculations involved in A* search. Since A* must traverse the grid of nodes, at worst it traverses all nodes. By contrast, simplifying the path found from A* will only traverse the nodes in the previously found path. In practice, this makes the path simplifying calculations relatively constant. Thus, Smoothed A* requires a small time overhead of roughly 3-6ms when compared to normal A*. This means that Smoothed A* is not preferable in environments with extremely small size and low complexity.

As stated in section 3.3, the actual smoothing in Smoothed A* takes place as the agent follows the path. Since this would require the additional time for the agent to travel from the start to end positions, the lerp calculations involved in this were not considered in these run times. Although these calculations do carry compu-

tational costs, it is typically the case that an actor following a path will do so at a speed which takes longer than the time necessary for these computations. Therefore, they be safely ignored in this data set, leaving only the calculations necessary to begin traversing a path.

5.3 Memory Usage

Since game environments are continually increasing in size and complexity, the amount of memory a pathfinding algorithm uses is an important aspect to consider. Figure 8 shows the memory usage of the algorithms in the three environments (represented by the amount of nodes created).

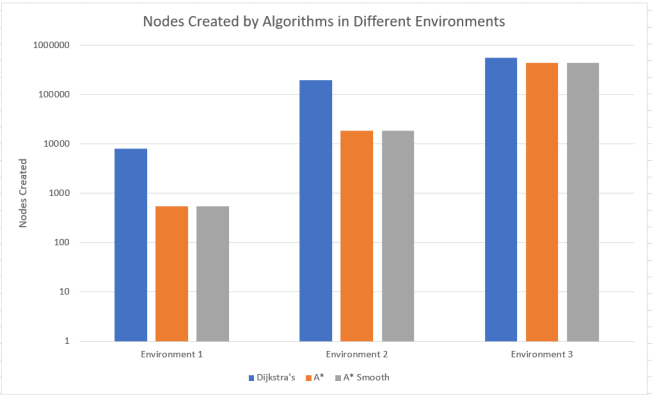


Figure 8: Algorithm Memory Usage Across Environments

This figure illustrates that both A* Search and Smoothed A* Search occupy the same amount of memory regardless of environment size and complexity. Since Smoothed A* is using normal A* to find a path and is then performing computations on that path, this conclusion is unsurprising. However, this is useful information for game developers with the desire to find paths across large, complex terrain. Memory usage should not be a deciding factor between choosing A* or Smoothed A*, since they perform identically in this regard.

Figure 8 also shows that Dijkstra performs significantly worse than the other two algorithms with regards to memory usage. Similar to the run time, however, this disparity becomes smaller as the environment size and complexity increases (likely for the same reasons discussed in section 5.2). Therefore, if memory is a primary concern, Dijkstra may still be viable in large complex environments. This being said, it is still not the preferred option when compared to A* Search.

6 Conclusion and Future Work

From the results outlined in section 5, it was found that realism with regards to pathfinding is more difficult to define than initially hypothesized. Before beginning the experiments, it was thought that Smoothed A* would always produce the most realistic path because it contained no harsh turns. However, the qualitative analysis suggests that too much smoothing can create exaggerated curves which are arguably less realistic than sudden angular changes. Therefore, it can be concluded that in order to create a realistic path, one must take care to find the correct amount of imperfection. Too much may seem illogical, while too little may seem robotic.

Furthermore, the analysis of the time complexity and memory usage suggests that there is never a reason to choose Dijkstra over A* Search when efficiency is a concern. If one is deciding between implementing A* or Smoothed A*, run time is the only consideration to make in terms of computational cost. However, the difference in run time becomes negligible as environment size and complexity increases.

Since it was found that size of the curves created by Smoothed A* have an impact on path realism, further experimentation with different turn distances (the variable which determines curve size, as mentioned in section 5.1) would be desirable.

In addition to turn distance, experimenting with other aspects that pertain to path realism would provide deeper insight into the problem. For example, implementing path weighting would be a possibly prevent the algorithms from "hugging" walls. By adding movement penalties close to obstacles, the algorithms would be less likely to move directly along them.

Finally, further experimentation which examines environment size and complexity separately would be useful. Although section 5 does distinguish between the effects which environment size had from environment complexity, examining these variables in isolation would lead to an even greater depth of analysis.

References

[1] Hernandez, Asin, Baier."Reusing Previously Found A* Paths for Fast Goal-Directed Navigation in Dynamic Terrain," *Proceedings of the Twenty-Ninth AI Conference on Artificial Intelligence*, vol. 1, no. 1, pp. 1158-1164, 2015.

[2] Cui, Xiao, Shi, Hao. "A*-based Pathfinding in Modern Computer Games," *IJCSNS International Journal of Computer Science and Network Security*, vol. 11, no. 1, pp. 125-130, 2011.

[3] http://www.gameanim.com/2009/12/22/the-ai-systems-of-left-4-dead/ *Online, Reactive Path Following in Left 4 Dead*

[4] https://github.com/SebLague/Pathfinding *Online, Smoothed A* implementation in Unity*

[5] `https://www.youtube.com/channel/UCmtyQOKKmrMVaKuRXz02jbQ` *Online, The YouTube channel of Sebastian Lague*