A star Search Algorithms in Video game Pathfinding Comparison

Alexa Heidgerken

July 11, 2023

Abstract

Pathfinding in video games is a topic that has been heavily researched as movement is one of the most basic requirements for any video game. There are many different ways to go about pathfinding such as different algorithms, accounting for different variables and more. This document includes a literature review of previous work done in this area and then also experiments with four different algorithms: Dijsktras, breadth first, depth first, and A star. The experiment uses a game of snake that compares the algorithms using a scoring system.

1 Description

In video games, pathfinding is an important concept. Pathfinding has been a major research area in video games for many years. Many different MMOs(massively multiplayer online), MOBAs(multiplayer online battle arena), and phone apps use pathfinding. From League of Legends to Clash of Clans, most games have some sort of AI minion character that the player uses or kills. A lot of the times, these characters need to utilize the shortest path from one place to another. It also needs to account for barriers, other enemies in their path, or even use a teleportation device. In order for these games to work correctly, the characters need to travel the shortest distance in a predictable path for the players to understand.

This experiment focuses on the shortest distance comparing different algorithms rather than the obstacles or other factors. Since there are so many different video games, many use different algorithms for pathfinding such as Dijsktra's, breadth first search, depth first search, and A star search. It then applies it to a maze, which the algorithm then attempts to solve. The code used in this experiment is online readily available code from github that compares the four different algorithms using a game of Snake. The snake game is then scored by the number of food eaten and how many moves the snake made from the algorithms choices. The algorithms have to choose the paths between the fruits to maximize their score while also choosing the lowest cost path. Because so many movements are involved, the game of Snake is a great example game.

2 Review

Pathfinding in video games uses many different search algorithms to find the best and shortest path. There may be many obstacles that the algorithm has to take into account, such as water, enemies, or walls you may have to break to enter. Clement Mihailescu has done a project very similar to this, with comparing various different search algorithms and taking in different maps and obstacles. [Mih]

In 2005, Sven Koenig and Maxim Likhachev focused on the adaptive A star algorithm, in which the adaptive A star algorithm repeatedly finds the cost-minimal paths to a given set of goal states in a state space. The adaptive A star search was then compared to the original A star search and it was found that the adaptive A star search was more efficient during the experiment. As well as it could be made more efficient through other means of heuristics. [KL05]

In 2010, Xiao Cui and Hao Shi published A star based Pathfinding in Modern Computer Games, the article focuses on how an A star search algorithm is used, and how it applies to video games. An A star search algorithm is an informed search algorithm which makes choices based on the estimated cost and the actual cost of the move. Xiao Cui and Hao Shi go into detail about A star optimizations and how using optimizations can make it easier to take in various barriers the character may come across when moving. The optimizations described are search space, hierarchical pathfinding, navigation mesh, memory, and heuristic function. In terms of the search space, the smaller it is, the faster the A star search is. Hierarchical pathfinding changes the way that the A star searches as a whole, but uses smaller parts. Hierarchical pathfinding will break down the search by taking the larger picture and finding a path for that, then will search the paths between paths to find the complete route. Take for example travelling from one city to another. It will search for the fastest path between cities, then it will search how to travel through each city to get to the next city that was already determined. Because A star keeps track of every movement, reducing the memory is a great optimization. By reducing memory waste, pre-allocating a minimum amount is the best way to go about it. In a heuristic approach, only the better looking options are searched, meaning the search will be faster, but it could possibly skip over the fastest path if the first step is more costly than the others. Age of Empires is the example given that applies an A star search algorithm that is poorly done and the authors articulates what could be done to improve the pathfinding by using NavMesh. [CS10]

In 2015, Parth Mehta, Hetasha Shah, Soumya Shukla, and Saurava Verma wrote a review on algorithms for pathfinding in computer games. This review compares Dijsktra's algorithm, greedy best-first-search and A star, which is very similar to what this project does. Dijsktra's algorithm examines nodes and grows in an outward fashion to find the shortest path to the goal, very similar to breadth first search but it also keeps track of the weights and will keep checking the shortest path not already checked until it gets to the target. The three different algorithms are examined and explained. Very similar to Xiao Cui and Hao Shi's paper, heirarchical pathfinding, NavMesh, search space, memory, and the Heuristic function are all explained and talked about how they can optimize the three different algorithms in this review. This review cites Hao Shi and Xiao Cui's review as well as uses some of the same visuals. The A star algorithm is visually compared to Dijsktra's algorithm and greedy best-first-search and

illustrates how the A star search algorithm takes the advantages of both and eliminates the drawbacks of both algorithms. [MSSV15]

In 2019, Clement Mihailescu published his project online along with a video explaining it. This website lets the user create a map with obstacles and then has 8 different search algorithms that the user can run and compare the results. The main search algorithms used are breadth-first, depth-first, A star, Dijkstra's, and greedy best-first-search, which are what both the articles above have written about, but this website shows it in a visual, making it easier to understand. [Mih] His YouTube video then explains how the algorithms work and searches the maze the user inputs. [you19]

In 2020, Azyana Kapi published a review on informed search algorithms for video game pathfinding, which again, like the other two reviews brings up NavMesh and heuristic functions and how they can optimize searches. This review cites both of the articles above as well. The main focus of this article is combining different searches and comparing the results, resulting in A star search being the best and fastest search algorithm. [Kap20] Another review paper was published in 2020, which also describes A star, Dijkstra's, genetic, and ant colony algorithms. The genetic and ant colony algorithms are used to find the lowest cost path, which is the largest drawback of the two algorithms. This review also gives a basic description of the different optimizations used. With A star, the biggest drawback is how much space it uses. [AR20]

In 2021, research using systematic literature review was done. Xiao Cui and Hao Shi's paper was cited as well as 39 other papers. The main takeaway from the review is that a regular A star algorithm is not good enough for video game pathfinding and some modifications to the algorithm is needed. Similar to the other papers, hierarchical pathfinding was found to speed up the run time, but may not always find the best path. Today's variations of the A star algorithm have better speed and efficiency than the old A star algorithm as video games now are a lot more complex than a simple game like Pac-Man. By combining algorithms or using hash-tables, a better heuristic is able to be generated and modifications that target the weaknesses of the algorithm are more common today. [FGK+21]

Through each paper, the different search algorithms that were used in the review were thoroughly explained, along with the different means of making the A star algorithm more efficient, mainly through heuristics, NavMesh, and minimizing space. Each different algorithm was compared and it was found in each that a modified A star search is the best way to solve video game path finding as today's video games are much more complex that the video games made in the 2000's.

3 Approach

There are many different ways to approach this problem. As in the literature review, most simply compared how fast an algorithm could go from one area to another with obstacles in the way, such as a maze. Using a maze game or a snake game seemed to be the two most commmon approaches in comparing video game pathfinding algorithms.

Using the approach of a game of snake, the results are easier to see and understand as multiple decisions are made during the gameplay. The first step was finding research previously done on video game pathfinding algorithms, which is a thoroughly researched topic. There were many example codes online regarding a snake game and using search

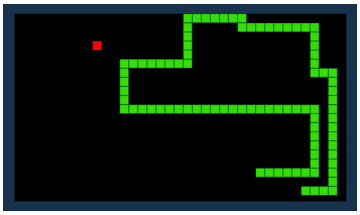

Figure 1: An example of a snake game

algorithms to play a game of snake 1, from freeCodeCamp.org and Python Engineer on YouTube. Through these example codes, this experiments code was found and slightly edited. Using research from these two example codes, a search algorithm comparison was created, there are a few online example projects very similar to this one and most of the code came from those. [Tec21]

This example code utilizes three different search algorithms, so Dijsktra's algorithm code was added through example code available utilizing only Dijsktra's algorithm. Each algorithm was implemented on the snake code individually and then scored. The code outputs a .txt file that contains each algorithms number of actions taken and how many fruits eaten. This txt file can then be analyzed to find the highest scoring algorithm average.

Below is an example of the depth first search code used in this snake game search algorithm comparison. Breadth first is very similar to depth first in this fashion.

```
while 1:
if dfs_stack.isEmpty():
    break
current, directions = dfs_stack.pop()
# print("Current pos:", current)
if current not in visited:
    visited.add(current)
    if s.isGoalState(current):
        s.score += 1
        s.addCube()
        performActions(directions, slow)
        # print("DFS number of actions:", len(directions))
        actionsList[0].append(len(directions))
        # print("DFS score:", len(s.body))
        scoreList[0] = len(s.body)
        # scoreList[0] = s.score
    for childNode, direction, cost in s.getSuccessors(current):
        if childNode not in dfs_stack.list:
            if childNode in visited:
                continue
            dfs_stack.push((childNode, directions + [direction]))
```

A snake game will start as an open area where the snake is two blocks long. As the snake eats more fruits on the map, it will grow one block and gain one point. Eventually, the snake will grow large enough it could block itself off. When the snake

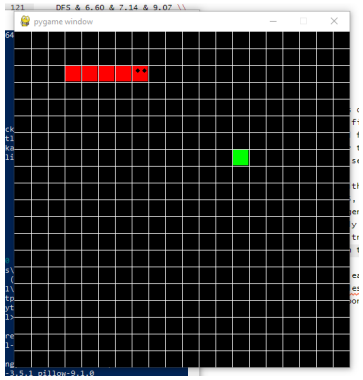

Figure 2: The game created when running showexample() which plays through each algorithm once and shows where how the snake is moving based on the algorithm's choices

hits itself or the edge border, the game is over. Because there are many movements needed to control the snake in order to prevent it from hitting walls or itself all while collecting fruits, there is a lot of decisions the AI has to make.

In this experiment, multiple trials are ran of each different algorithm and then given a score based on how many movements it made compared to how many fruits were collected. A series of trials is then ran since the fruits will not always spawn in the same areas. The scores from each trial is then averaged to compare the four different algorithms. Through the number of moves made and how many fruits are collected, the efficiency of each algorithm is able to be calculated.

4 Experiment and Results

The code itself uses two files. Snake.py contains functions that moves the snake and keeps track of how long it is, where it is, and if it is alive. It also contains each function that tests the different algorithms. Util.py contains helper functions for implementing the search algorithms such as queues. This code is from FreeCodeCamp.org and Lazy Tech on YouTube. [Tec21] [Car22] The .txt file contains all the scores and number of moves each search algorithm makes. In order to run the code, pandas and pygame must be installed.

This experiment utilizes four different algorithms. The code has four different ways to run it. Running main() will run the game of snake by itself with the player playing it. Running showExample() will run all four searches, one at a time starting with depth first, breadth first, A star and then Dijsktra's all while printing each move the algorithm makes. Running runmultiple(times) runs each search an x amount of times for five fruits and will place the results into a .txt file for analysing. Running runsearch(times) will run each search algorithm and for 400 moves or until the snake dies and then write the results into a .txt file. Below is an example of the .txt file after running runsearch(1).

```
BFS ACTIONS:            1318
DFS ACTIONS:            4127
ASTAR ACTIONS:          1463
```

```
UCS ACTIONS:            1391

RAW BFS SCORE:          42
RAW DFS SCORE:          87
RAW ASTAR SCORE:        88
RAW UCS SCORE:          87

CALC BFS SCORE:         1.0176883935061787
CALC DFS SCORE:         6.600910470409711
CALC ASTAR SCORE:       6.015037593984962
CALC UCS SCORE:         6.2544931703810205
```

Showexample() 2 is run first in order to visualize each search algorithm and understand the speed of the algorithm and how it makes it decisions. A decision is come across moving each block: whether to go forward, turn left or right. This means that many moves are made. Then, runmultiple(3) is ran, in order to run each algorithm multiple times for a basis of data to compare. Runsearch(3) is then run to get the final data components such as the averages of each score.

Below, the table is from running runmultiple(3)

Search	Moves1	Score1	Moves2	Score2	Moves3	Score3
DFS	4127	87	3721	34	3894	63
BFS	1318	42	375	37	820	40
A star	1463	88	395	36	822	62
DIJ	1391	87	393	38	822	62

From calculating the score of each by (score / actions) * 100, the calcscores() function in snake gets the average and prints it to the .txt file as well. This occurs during runsearch(times). Below is a table of the scores gotten from each algorithm by using calcscore() on each algorithm's scores.

Search	Trial1	Trial2	Trial3
BFS	1.02	0.79	0.99
DFS	6.60	7.14	9.07
A star	6.02	6.16	9.11
DIJ	6.25	6.42	9.67

5 Analysis

In figure two, 3 it shows a graph of the average scores of the four algorithms. Overall, breadth first search performs the best by a large amount. By running showexample(), a graphical image of the game play is produced. From this, depth first search runs pretty slow with a lot of actions. Breadth first search runs quickly and linearly with a lower amount of actions taken. A star search runs quickly and in a diagonal fashion but takes more actions than breadth first, but usually has close to the same amount of actions as breadth first. Dijsktra's runs quickly and diagonally and is very similar to the A star search, even having the same amount of moves and score in trial 3. Dijsktra's and A

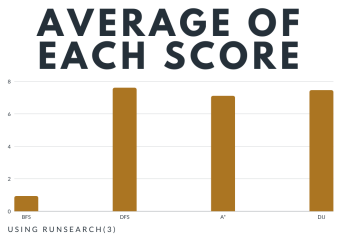

Figure 3:

star often have a good score in the game with generally more actions than breadth first search.

In terms of scores, depth first search is the best scoring algorithm. With an average score of 7.6 being the highest out of all four algorithms. Breadth first search had a average score of 0.93, A star had a 7.1 and Dijsktra's had an average of 7.45, which is very close to depth first search's score. Because A star and Dijsktra's use a variation of depth first search, it was expected that all three would be fairly close. With a larger scope, perhaps about one hundred trials, the scores would most likely be very close. It is interesting how low breadth first search's score was. It seemed to perform very poorly in a snake game. This is most likely because of the randomization of where the foods show up on the map for the snake to travel to, as it seemed to have a better score in other trials. Although breadth first search consistently had the highest amount of moves made. This is because the snake traverses to the top of the map then to the next line, then to the bottom of the map, searching every box up and down. When the A* and Dijsktra's algorithms were run, they were much faster and had a smaller amount of moves than breadth first as they moved somewhat diagonally, moving one over and one down to get to the fruit in the fastest way that was calculated. Both algorithms use a diagonal distance formula in calculating where to move.

Compared to other experiments such as Clement Mihailescu's maze, each algorithm was fairly close in results. Perhaps with a larger scope breadth first would perform better, or it simply does not perform well in a snake game. In comparing Mihailescu's maze algorithm comparison to this experiment, depth first search seems to be the fastest and most accurate search algorithm out of the four. Breadth first also performed poorly, but was not too far behind in terms of score, which is very different from the snake game scores. Reiterating that with a larger scope of trials, breadth first most likely will perform better.

In regards to the A star search algorithm, there are many ways that it could be improved by its heuristic function. This experiment uses the Manhattan distance to calculate the distances of the current location of the snake and the fruit. In using a better heuristic, A star would score better than the depth first search algorithm. Because the map of the game is done by blocks, moving diagonally needs two moves, one over and one down. Meaning for calculating distance, a diagonal distance cannot be used as it takes longer than simply moving over all of the way and then down, which is what the depth first search algorithm does. The Euclidean Distance heuristic also moves diagonally so it is not a wise choice for this type of game. Because the snake cannot move diagonally, the A star search cannot use a better distance formula and must stick with moving one over and one down to move diagonally.

6 Conclusion

Video game pathfinding is a complex problem as there are so many different variables and problems. Movement is the most basic element in any video game and is successful movement is necessary for a video game. The literature review down looks at the different search algorithms. Each piece included in the literature review looks thoroughly at how each algorithm works and what each does. Many of the pieces also include details regarding how to maximize the A star algorithm's efficiency

Through each paper, the different search algorithms that were used in the review were thoroughly explained, along with the different means of making the A star algorithm more efficient, mainly through heuristics, NavMesh, and minimizing space. Each different algorithm was compared and it was found in each that a modified A star search is the best way to solve video game path finding as today's video games are much more complex that the video games made in the 2000's.

This experiment looks at different algorithms that can be used in a snake game, one of the first video games programmed in 1977 for Nokia. Depth first, breadth first, A star and Dijsktra's algorithms are the four searches used in this comparison. Depth first was found to be the best search algorithm as it continuously had a good amount of points and did not have a lot of moves. Because the snake game map is made up of cells the snake can only move left, right, up or down, the A star search algorithm did not perform as well since the better distance heuristics cannot be used. The same problem would arise if the game was a maze as well.

With future research, an A star search that can find the shortest possible path and adapt to a grid type would help solve many different problems in older video games that use grid types for its map. However, in today's video game industry, most games no longer use a grid and have coordinates and planes that make up the map that must be taken into account when moving in those games. Video game technology now highly surpasses what was used just even five years ago. With future research, the problem of space complexity in A star can be solved, as it seems to be the largest issue today as it stores all nodes in its memory.

References

[AR20] Abdul Kadir Siti Normaziah Ihsan Abdul Rafiq, Tuty Asmawaty. Pathfinding algorithms in game development. *IOP Conf. Series Materials Science and Engineering*, pages 769–780, 2020.

[Car22] Beau Carnes. Train an ai to play a snake game using python, website, 2022.

[CS10] Xiao Cui and Hao Shi. A*-based pathfinding in modern computer games. *IJCSNS International Journal of Computer Science and Network Security*, 11:125–130, 11 2010.

[FGK+21] Daniel Foead, Alifio Ghifari, Marchel Budi Kusuma, Novita Hanafiah, and Eric Gunawan. A systematic literature review of a* pathfinding. *Procedia Computer Science*, 179:507–514, 2021. 5th International Conference on Computer Science and Computational Intelligence 2020.

[Kap20] Azyan Kapi. A review on informed search algorithms for video games pathfinding. *International Journal of Advanced Trends in Computer Science and Engineering*, 9:2756–2764, 06 2020.

[KL05] Sven Koenig and Maxim Likhachev. Adaptive a*, 2005.

[Mih] Clement Mihailescu. Pathfinder visualization.

[MSSV15] Parth Mehta, Hetasha Shah, Soumya Shukla, and Saurav Verma. A review on algorithms for pathfinding in computer games. 03 2015.

[Tec21] Lazy Tech. Comparing search algorithms on snake, youtube, 2021.

[you19] *The Projects That Got Me Into Google (tips for software engineering projects).* YouTube, Aug 2019.