Kennesaw State University

2D Path Finding Algorithm

CS6045 Advanced Algorithm Project Report - Team 4

Member:
Po Chun Lu : (Idea, A* algorithm, Exhaustion method, paper work)
Date:
December 3, 2020

Abstract

Find an appropriate algorithm is the most important step for solving problem. If the problem is not that clear to defied, then I should find the similar one. For this problem (Bead moving mobile game), I considered it as a kind of Travelling Salesman Problem (TSP) problem[1]. This problem does not have a big input size, so I think there must be at least one solution for this problem. I have considered A* algorithm, Dijkstar algorithm, D* algorithm. I finally choose to compare the performance between A* algorithm and Exhaustion method. If only comparing the performance metrics we set, Exhaustion method has the better result, but it takes more times than A* algorithm.

Introduction

In the class, we just discussed the TSP problem, and I just found some features are same as the mobile game I was playing, so I decided to take a look and analysis this mobile game. In the beginning, I thought this problem was pretty simple, and might not have too much content to become a project. However, it turned harder than I thought when I paid more attention on this. For instance, it doesn't like the original path finding problem, there is not fixed start and end point. As the result, we want to check if any existing path finding algorithm is also suitable for solving this problem.

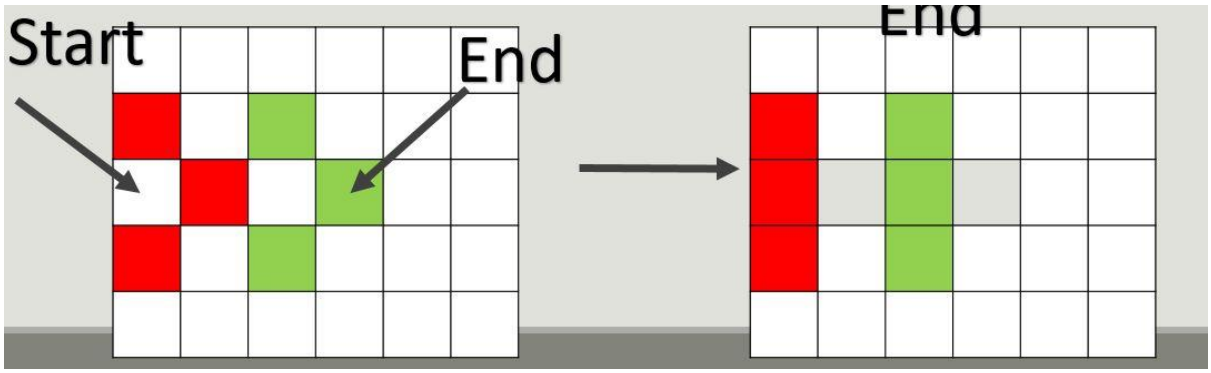
Problem Definition

As I mention, this is a mobile game, and I don't have the skill to develop a tool for phone, so I create the similar situation on computer. 6 times 5 matrix with random distribution color character is the input (fig.1), and I hope to find the maximum number of set. Set's confirm when there are more than three same color character stage together in a line. For example, there is one set in fig. 1, and it is the first three yellow "Y" on the left top corner.

To get the result, I have to choose a start point (character), and then move it to the end point. When the character move, it will change the position with those one the path, after that, I can get the final distribution, path and the number of set (fig.2) . Start and Destination points are not fixed, so I need to analysis the situation every time.

To evaluate the performance, in the beginning, I use the set/cost metrics for comparison. To find which algorithm can get larger set per cost. However, this metrics has some problem, I will explain more in Solution – End point decide part. Therefore, I change the metrics to compare max set number with 9 costs (movements) between two methods.

(fig. 1, input)



(fig. 2, moving & swapping)

Solution

A* algorithm[3]

A* algorithm is the most common path finding algorithm, and most of the NPC in games are using this algorithm as well. (J. Hu, W. gen Wan and X. Yu, 2012)[2].

Start Point choice

With random distribution of the input, I have to pick an appropriate start point for getting optimal result. My strategy is to pick up the point whose element is scattered, so I need to calculate the weight of every point. If there are same color nearby, add the weight of that point. (fig.3)

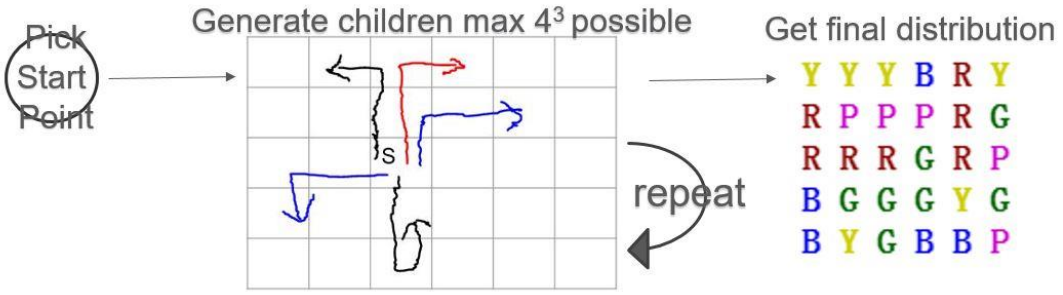
Y	Y	G	Y	Y	B	+1	+2	+1	2	2	4	2	2	0
P	G	G	G	G	G	+2	C	+2	0	5	7	6	6	3
Y	G	Y	P	G	P	+1	+2	+1	0	5	0	3	4	0
R	G	P	P	Y	B				1	2	4	6	0	0
Y	R	R	P	G	P				0	3	2	3	0	0

(Fig.3)

After compute the weight, points with the zero weight will be chose. If there are more than one points have zero weight, program will random choose the start point from all the zero location.

Path finding

From the start point, I can move up, down, left, and right. In the original A* algorithm, one step at once is considered, but it's not suitable for our problem. The reason is that one movement means one swap, only two sets will be created at most. Therefore, in this situation, I decide to have tree movements at a time, so the children number will come to 4^3 (every move have 4 direction). This children number is really big, and it is the reason than I don't choose more number of movements at a time. After generate children, compute which has best result, and storage the distribution, path and set number, and then do next run until get total 9 movements. (fig.4)



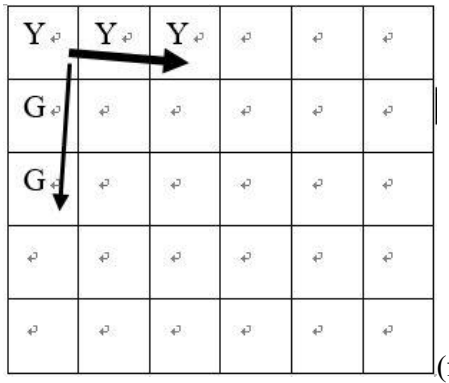
(fig. 4)

End point decide

When this program will stop? In the beginning, I use Set/Cost rate as the performance metrics. However, due to the input is generate randomly, sometime will get 1 or more set without any movement. As the result, I change the method to evaluate the performance. Algorithm have to spend 9 costs (9 movement) to get the result, and I will compare the maximum number of the set between two methods.

Set compute

Start from left top (first) point, scanning two point on the right and two one the bot, if they are the same set number plus one. (fig. 5) To avoid double count, also handle an exist array for those point which already count. If one of the three point is already in that array, others will append into the exist array, and the set number won't add up.



(fig. 5)

Exhaustion method

In Exhaustion method, because I think the input scale (6*5 matrix) isn't too larger, this method is to find the all possible results. Instead of random start point chose, it's start point is from 0 ~ 29 (every points), and so no need to compute the weight of every point.

Path finding

It is quite simple, just use 9 for loop to generate the children, and only pick up the one

which has the maximum set number.

Set number compute function is same as A* algorithm

Dijkstar algorithm

Original algorithm we are going to use, but one of team member left our team, and I don't have enough time to finish it.

D* algorithm[4][5]

It is one of the method I have considered. The reason I didn't choose is because, in my understanding, this algorithm suitable on dynamic obstacle. That is, obstacle will change over time, but in our problem, I don't have the obstacle. In the beginning I have, but I found that if I set the point as an obstacle, sometime those obstacles will block the optimal solution.

Performance Evaluation

A* algorithm result

The (fig.6) below shows the initial set number and distribution, and also the final set number and distribution. The point is that some time if the program chose perfect start point only three movements (costs) can get a lot of set number than original.

```
initial set number: 1
#===========first===========#
set number after 123 step (stage 1): 4
star_point:25
#===========second==========#
Set number after 456 step(stage 2): 4
start point of(stage 2): 7
#===========third===========#
Set number after 789 step(stage 3): 5
Start point of stage 3: 3
#===========final===========#
[25, [19, 13, 7], [8, 9, 3], [4, 10, 9]]
initial:
Y Y Y R B Y
R R P P R G
R G R G R P
B Y G G Y G
B P G B B P
Final:
Y Y Y B R Y
R P P P R G
R R R G R P
B G G G Y G
B Y G B B P
```

(fig.6)

Exhaustion method result

The (fig.7) below shows partial of onetime simulation of exhaustion method. It will show the best result and the path from every point. It also shows the set number

```
initial array:
P B R R Y Y
G R Y P Y B
R G P B P Y
P Y R Y Y B
R Y R R Y G
initial set number: 0

Start Point: 0
Set number: 3
B R R Y Y Y
G R Y P P P
R G P B Y B
P Y R Y Y B
R Y R R Y G
[1, 2, 3, 4, 10, 16, 17, 11, 10]

Start Point: 1
Set number: 4
P R R R Y Y
G Y P P Y B
R G R B Y B
P Y Y Y P B
R Y R R Y G
[7, 8, 14, 20, 21, 22, 16, 17, 11]

Start Point: 2
```

(fig.7)

A* algorithm V.S. Exhaustion method (100 time simulation)

To compare these two method, I let the program run 100 times and get the average set number of both methods. Fig.8 is the result for A* algorithm. Fig.9 is the result of Exhaustion. In my original thought, A* algorithm might powerful than Exhaustion method. If I only compare the metrics I give (set number with 9 movements), Exhaustion method is better than A* clearly. However, if I consider about input scale and execute time, A* algorithm definitely will destroy Exhaustion method. Exhaustion method is going to find all the possible solution and pick the best one. In this scale of the input it will take almost five minute to finish onetime simulation, meanwhile A* only need few second.

```
90 Times path: [3, [4, 10, 9], [15, 14, 20], [21, 22, 28]] Set number: 4
91 Times path: [10, [4, 3, 2], [8, 14, 13], [7, 8, 2]] Set number: 3
92 Times path: [14, [13, 7, 1], [2, 3, 9], [8, 7, 1]] Set number: 5
93 Times path: [7, [13, 14, 20], [21, 27, 26], [25, 24, 18]] Set number: 3
94 Times path: [5, [11, 10, 9], [10, 16, 17], [11, 10, 9]] Set number: 2
95 Times path: [24, [25, 19, 13], [14, 20, 21], [15, 14, 8]] Set number: 4
96 Times path: [0, [1, 2, 3], [4, 5, 11], [10, 4, 3]] Set number: 4
97 Times path: [15, [16, 22, 23], [17, 11, 10], [9, 3, 4]] Set number: 4
98 Times path: [11, [10, 4, 3], [9, 15, 14], [8, 9, 3]] Set number: 4
99 Times path: [3, [9, 8, 14], [8, 9, 10], [16, 22, 21]] Set number: 4
100 Times path: [27, [21, 20, 14], [20, 19, 18], [12, 13, 14]] Set number: 3
Largest set number: 6
Smallest set number: 2
Average set number: 3.46
```

(fig.8)

```
90 Times path: [3, 9, 15, 14, 20, 26, 25, 19, 18] Set number: 4
91 Times path: [1, 2, 8, 9, 10, 16, 22, 21, 20] Set number: 4
92 Times path: [20, 21, 22, 16, 15, 9, 10, 4, 3] Set number: 6
93 Times path: [2, 8, 14, 20, 26, 27, 21, 15, 16] Set number: 4
94 Times path: [11, 10, 9, 3, 4, 10, 9, 8, 2] Set number: 5
95 Times path: [1, 2, 3, 2, 8, 7, 13, 12, 18] Set number: 4
96 Times path: [2, 3, 9, 8, 14, 15, 16, 22, 21] Set number: 5
97 Times path: [1, 2, 1, 7, 13, 12, 18, 19, 13] Set number: 6
98 Times path: [23, 29, 28, 22, 21, 20, 14, 8, 9] Set number: 5
99 Times path: [17, 16, 15, 14, 8, 9, 15, 21, 22] Set number: 5
100 Times path: [15, 14, 8, 7, 13, 14, 20, 19, 25] Set number: 4
Largest set number: 7
Smallest set number: 4
Average set number: 4.96
```

(fig.9)

Prognosis

Exhaustion method

When I implement Exhaustion method, I use very straight forward method (9 for loo). I am not sure if this is the problem that make this method have very long execution time.

Set evaluation function

About set evaluation function, I have write two versions. One version can check the set number fast but there are some special cases it won't work. The other version can check all the possible problem correctly but it will consume extra large of time, due to a lot of comparison. The special case doesn't happen every time, so I chose the one has lower execution time.

Below is the example (fig.10) first is correct but take a long time. Second one is not correct on special cases, but it works on other cases.

```
G Y Y Y P B
P P B P P P
B P B G P Y
G P B B P R
Y B P P P P
4
```

correct set compute

```
G Y Y Y P B
P P B P P P
B P B G P Y
G P B B P R
Y B P P P P
5
```

wrong set compute

(fig.10)

Conclusion and Future Work

To choose an appropriate solution for a problem is really important. Although this problem is simple, I still spend a lot of time for finding a solution and implementing. The most difficult time is the clarify the problem and find the similar problem and its solution which is already existed. Like our project, I found the puzzle game at the previous day of second presentation. At that time, I have already started implementing. In this project, due to the metrics I set, it doesn't show that A* algorithm is better than straight method. If I can consider more, it definitely can show how better the A* algorithm is.

I don’t have enough time to implement the other algorithm, like Dijkstar and D*. About Dijkstar, due to my team member’s left. About D* I think I ought to understand more, there might have some methods can override the problem of none obstacle problems. Moreover, larger scale of the input also doesn’t measure this time. More implement definitely can show that choosing appreciated algorithm is really important.

Code – Google Codelabs

A* algorithm 100 times:
https://colab.research.google.com/drive/1niL0ja6Q8gTPt4femZRrS_o7zJiisAhE?usp=sharing
Exhaustion method 100 times:
https://colab.research.google.com/drive/1gQus2BNrsxI7xn3QzjnXRWjUelVsyk93?usp=sharing

Reference

[1]. Wikipedia contributors. "Travelling salesman problem." Wikipedia, The Free Encyclopedia. Wikipedia, The Free Encyclopedia, 2 Dec. 2020. Web. 3 Dec. 2020.

[2]. J. Hu, W. gen Wan and X. Yu, "A pathfinding algorithm in real-time strategy game based on Unity3D," 2012 International Conference on Audio, Language and Image Processing, Shanghai, 2012, pp. 1159-1162, doi: 10.1109/ICALIP.2012.6376792.

[3]. A* algorithm(original): https://www.geeksforgeeks.org/a-search-algorithm/

[4]. Stentz, Anthony. "The focussed d^* algorithm for real-time replanning." IJCAI. Vol. 95. 1995.

[5]. Stentz, Anthony. "Optimal and efficient path planning for partially known environments." Intelligent unmanned ground vehicles. Springer, Boston, MA, 1997. 203-220.