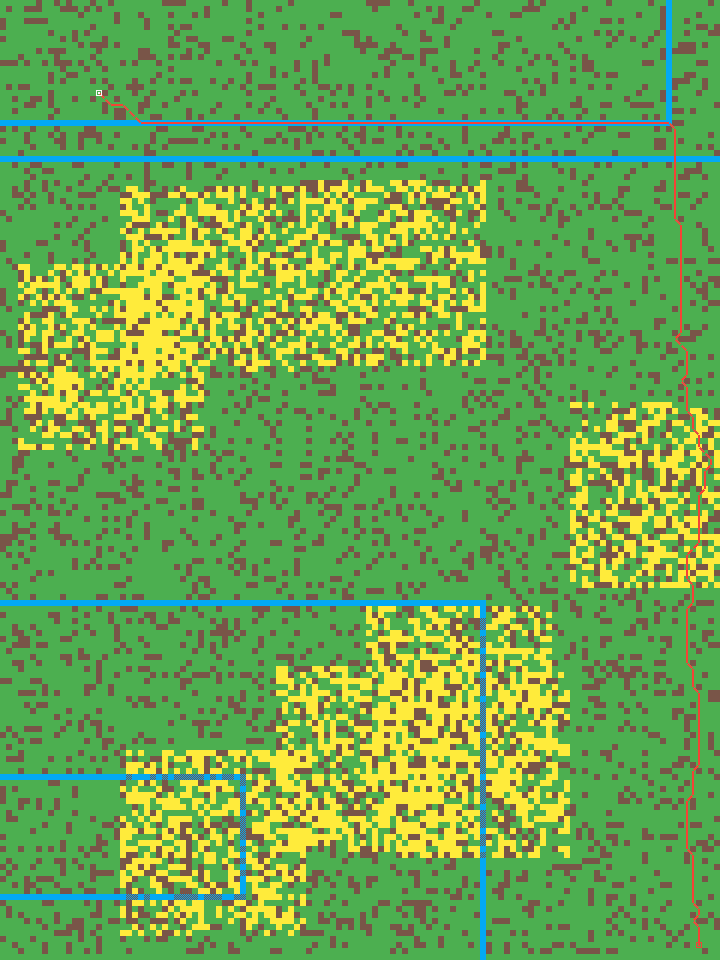
Team member

Yifan Zhang yz745  
Tin Fung tyf3

Yiqian Li yl1165

Xiwen Shen xs143

Basic map information



This is the sample map image for our project. The gird map is 160\*120. Each little square represents a cell. In the coding part, we use a 2-D array to store all the map information. The array uses int 0~4 to present each type of cell.

In the image, the brown dot presents a block cell. In array, we use 0 to present.

In the image, the green dot presents a regular unblock cell. In array, we use 1 to present.

In the image, the yellow dot presents a hard traverser cell. In array, we use 2 to present.

In the image, the blue dot presents a regular unblock cell in a highway. In array, we use 3 to present. Later when we print the map in the terminal, we use 'a' to represent.

In the image, the dark blue dot presents a hard traverser cell with a highway. In array, we use 4 to present. Later when we print the map in the terminal, we use 'b' to represent.

In the image, the red line is the final way we find out. The point with a white outline is the start point and the point with a red outline is the endpoint.

Basic travel distance calculation

The following table is the distance traves at different condition.

|  |  |  |
| --- | --- | --- |
| condition | horizontally | diagonally |
| between two regular unblock cells  (1 to 1) | 1 | sqrt(2) |
| between a regular unblock cell and a hard traverser cell  (1 to 2) | 1.5 | (sqrt(2)+sqrt(8))/2 |
| between a regular unblock cell and a regular unblock cell with highway  (1 to 3)  (This condition only cross by highway) | 1 | sqrt(2) |
| between a regular unblock cell and a hard traverser cell with highway  (1 to 4)  (This condition only cross by high ay) | 1.5 | (sqrt(2)+sqrt(8))/2 |
| between two hard traverser cells  (2 to 2) | 2 | sqrt(8) |
| between a hard traverser cell and a regular unblock cell with highway  (2 to 3)  (This condition only cross by highway) | 1.5 | (sqrt(2)+sqrt(8))/2 |
| between a hard traverser cell and a hard traverser cell with highway  (2 to 4)  (This condition only cross by highway) | 2 | sqrt(8) |
| between two regular unblock cells with high way  (3 to 3) | 0.25 | sqrt(2)/4 |
| between one regular unblock cell with highway and a hard traverser cell with highway  (3 to 4) | 0.375 | (sqrt(2)+sqrt(8))/8 |
| between two hard traverser cells with highway  (4 to 4) | 0.5 | sqrt(8)/4 |

Read file and output

Format for the input of map file: java Astar

Will auto generate a map and run all the algorithms.

Format for the input of map file: java Astar map.txt

Users can type the name of the input file in the terminal.

The map file uses number 0~4 to present the whole map.

(the same int number as data stored in the array)

If the user also wants to input the start point and the goal at the terminal:

java Astar map.txt start.x start.y goal.x goal.y

the first input is the file name

the second one is the x coordinates for the start point

the third one is the y coordinates for the start point

the fourth one is the x coordinates for the goal point

the fifth one is the y coordinates for the goal point

weightedAstar20.generatePic("generatedPicW20.png");

This method will output is the similar to the sample diagram showed above.

Every time, the result is found, an image will print the final way and the number of steps will show in the terminal.

Therefore, as the final result, our project will create 50 different images for each map. The file No. 1~10 is the ten different start-goal parts of the map, and inside the file have the 5 different heuristics which is A\*, weighted A\* with value 1.5, 2.0, 2.5, and USH(Uniform Cost Search)

weightedAstar20.printPath();

This method will print the result in terminal.

It looks like : the \* points out the road.

图片包含 图形用户界面

描述已自动生成

If you run the file Astart\_withInPut.java, you will have the accuses to check the g-value, h-value and f-value after the algorithms find the route. It works on terminal. The first line asks weather user want to check the point. If true, it will ask user to input the x and y coordinate for the points they want to check. If the point is not reachable, it will return “the point did not be visited”, otherwise, it will return the g-value, h-value and f-value. If user type false, it will go to next algorithms.

文本

描述已自动生成

Q3. Optimizations:

Before, we want to use a 2-D array to store the information of the map. However, later, we realize it is difficult to keep all the information on the diagram we need. So, we created a new data structure called point to gain all the information we need.

Also, for the output image. Before, we use different numbers to present the route. However, it is a really difficult way to read it. Therefore, we choose to use a different color to present the diagram.

To calculate the final result, we use a for loop to get the average data for the result.

Also, we divided the function of check the g-value, h-value and f-value and the get the average results of 50 maps into 2 different java programs.

Because the function of check value will ask user to input the value after the find the way to the goal, so it does not work well with the for loops. So, for easy testing, we just choose two different programs to run it.

Q4

The admissible is A\*. It keeps a good balance on the run time and cost for the path. It has an h-value that keeps tracking of the distance between the current point and the goal. This way will make sure it goes in the right direction toward the goal. Also, compare to weighted A\*, its weight is small, therefore, it also has a chance to explore the area around, so it has a higher chance to find a highway to decrease the cost.

Other:

BFD: will check all the possible ways on the map. Therefore, it may go in the totally opposite direction, which is a waste of time. For example, the start point is at the upper right connector and the goal is at the bottom left part of the map. Most of the time the algorithm shouldn't visit the way to the upper right connector because it is the opposite direction of the map. However, the BFS does not have the h-value which will show whether the way toward the goal. Also, the BFS uses a stack to store app the possible points on the map. Therefore, the stack will become super large.

DFS: most of the time DFS will use the recursion. Therefore, lots of information will store in the system of the stack. If the map is too large, maybe there will be a problem with the stack of flow. Also, the same as BFS, DFS also needs to check all the possible ways in order to find the shortest path. It will visit the path to opposite directions.

Uniform-cost search: This requires a large form to record the shortest path to achieve passing points on the map. Therefore, it will record lots of unnecessary information such as the point in opposite directions. Because it treats all the points that have the same past cost equality, it will check all the points with distance less than the final step will be visited. Although it stores less information than DFS and BFS, it still will waste lots of time to update the direction information for the useless points. So, it will have a large memory usage and slower runtime. However, because it checks all the possible results. It should be the one with smallest cost.

Waited for A\*: compare to the A\*, it usually increases the cost for not go in the target direction. Therefore, the program will try in fewer possible ways. This decreases the memory need for the program. The path usually will be the direct line to the goal, so it has fewer steps. However, because it tries less possible ways. It is difficult for it to find the highway. Therefore, the cost may be higher.

Q5

Result for the average of 50

图形用户界面, 文本, 应用程序

描述已自动生成

Because we use a for loop to get the average of 50 results.

The Uniform cost search have the largest runtime which is 23ms. Then is the A\* which is 7ms. The weighted A\* have the same runtime which is 2ms.

The Uniform cost search have the smallest cost which is 173. Then is the A\* which is 199. The weighted A\* with w=1.5 have the cost of 210 and the rest have the same runtime which is 2ms.

The Uniform cost search have the largest step which is 257. Then is the A\* which is 172. The weighted A\* with w=1.5 have the step of 159 and the rest have the same runtime which is 156.

There is a problem of the result of memory: it has the negative number. Because the java will auto free the memory at the time do not need, so the memory use does not in sequence. If we use the ending memory address to minus the benign address, it might have some problem: the address is not used in sequence. Therefore, we need to get the memory before the free happens which means we cannot use for loop. Therefore, following is the data run by each individual time.

表格

描述已自动生成

Q6

As the diagram shows above, Uniform Cost Search has the smallest cost, because it checks all the possible ways. However, it takes most steps, because it searches for a large area in order to walk on the highway. Also, because of its search larger area and need to recode the g-value, h-value, and f-value for all the possible points. It requires the largest memory.

The A\* with a large weight will spend less time on run time. However, the cost will increase. Because when the weight increase, the program will check a less possible way solution, so it saves time and loses accuracy. Because it goes directly toward the goal, the step will also decrease. Since it checks fewer paths, fewer points will be visited and the memory requirement will decrease.

Q7

Before the implement, we assume the running time and the running memory usage of Sequential Heuristic A\* is much large than other algorithm since this algorithm is based on the weighted A\* and runs N+1 algorithm. However, the data shows that the usage of memory and the running time of Sequential Heuristic A\* does not have an obvious adjustment compared to our weighted A\*.

Also, we can find a variation w1 and w2 for the admissible and inadmissible situation, and this algorithm is more compatible than the original A\* since it concerns about the inadmissible problems.

Here is demo of a comparison between each algorithm

文本

描述已自动生成

Q8

For the implementations,

We reuse some code from the original A\* search, and analysis the optimal paths formed by weighted A\* search to get the different gi for (i=1, 2, …, n). For example, the sequential heuristics will implement from Astar class and overload some method from the original one.

To compare the weighted A\* and the sequential A\*, the procedure of Weighted A\* algorithm is much simpler than the sequential A, but the memory usage and running time of weighted A\* is similar to the memory usage and running time of sequential A\*. We conclude that for each time of using weighted A\*, Java will optimize the memory usage and release the useless memory, since the sequential A\* is based the weighted search to do N+1 searching.

Q9

Statement: For any u in OPEN0, the statement Key(s,0) <= Key(u,0) is valid when Key(s,0) <= w1\*g\*(Sgoal).

Prove:

From the A\* algorithm, we can prove that for any s, Key(s,0) <= Key(u,0) for all u in OPEN0 when go(s)<=w1\*c\*s where c\*s is the optimum cost to state, s. By the assumption of Key(s,0) <= Key(u,0) for any s, s is the state that get the smallest Key for all state from Sstart to Sgoal which is similar to the weighted A\* algorithm, so the optimal cost for state Sgoal is g which is g\*Sgoal. Therefore Key(s,0) <= w1\*g\*(Sgoal).

Then prove that when the Sequential Heuristic A\* terminates in the ith search, gi(Sgoal) <= w1\*w2\*c\*(Sgoal).

Where w1\*w2 sub-optimality factor

Since the sequential Heuristic A\* terminates in the ith search, there is no solution at this moment. gi(Sgoal) < OPEN.minkey() < infinity. Since there is no solution for Sgoal, OPEN.minkey() will be infinity so that the optimal cost to Sgoal is infinity. And OPEN.minkey() <= w2\* OPEN.minkey(). Since OPEN.minkey() = Key(s,0) and since we proved that Key(s,0)<=Key(u,0) for all states, OPEN.minkey() <= w1\* c\*s <= w1\*g\*(Sgoal). Hence, the current Open.minkey()<= w1\*w2\*g\*g(Sgoal) => Open.minkey()<= w1\*w2\*c\*g(Sgoal). Then we get the gi(Sgoal)<= w1\*w2\*c\*g(Sgoal). ­