PA1 Reporter

1. Main Purpose

The game is a classic Snake and the dimension is 400 by 300. We are supposed to implement a search algorithm to control the snake to get as many scores as possible.

2. Environment Analysis

The environment is a 400 by 300 grid and the snake can only move up, down, left, and right. So we can use the Manhattan distance as the actual cost between two points. Besides, the body is made up of some vertical and horizontal lines, which means we can do collision detection by some simple judgments instead of using mathematical methods to compute the distance between a point and the body.

As a general solution, we can simply let the snake move like a scanner by columns or rows. In this way, the snake will get all scores and has little possibility to have a collision with itself. However, this strategy is meaningless and boring. Besides, obviously, a teleportation feature can reduce steps in the path when snakehead and food are far apart, which I implemented in my algorithm.

3. Approaches

Basically, we can use a normal search algorithm, like the Dijkstra algorithm, Local-greedy search, Breadth-First search, Greedy Best-First search, and A-Star search to find a path for the snake. In the beginning, I used a Local-greedy algorithm. It ran pretty well when the body was not too long to bother its navigation. But it would surround himself when the body getting longer which would cause a collision eventually. After that, I was trying to use the Dijkstra algorithm. But I found it is not efficient in a grid game.

At last, I decided to use A-Star search, since it is not as primitive as Local-greedy search is and search way fewer points than Dijkstra. I used the Manhattan distance as the base cost, the cost between the current point and the head, and Euclid distance as heuristic distance, the estimated distance between the current point and the target.

But it didn’t run well. Though it took tens of seconds to find the path, could not even get high scores than using a Local-greedy search. To improve the search speed, I not only optimized some time-consuming functions but also implement a little feature that the search would stop if the base cost of the current point if greater than the length of the snake tail, the last line in the body, and make movements to that point, then start a new search from the new position. In this way, we can reduce the number of traversing nodes and take advantage of updated information, since the map will look different after the snake making some movement. But there was no noticeable improvement in terms of getting scores.

Then I noticed in the observation of snake behavior that, as the path is shortest, the snake always puts it in a dangerous position. Then a question came into my mind, is the optimal path necessary? Because in a snake game, all we care about is the scores. It doesn’t matter the snake moves around a little to the destination. So I turned the algorithm from A-Star search to Greedy Best-First search by setting all points’ base cost to zero. The scores were much higher than before.

In order to improve the ability on avoiding having a collision with itself, I thought it would be helpful if the point gets bigger cost if it is close to the body. Like two poles of the same magnetic field near each other. The idea is pretty good that the snake would prefer to choose the path away from the body, which means less possibility of collisions, but the snake is not absolutely can't move by its body. However, I failed on implementing this feature. It did prefer the points away from the body but moved strangely when turning.

4. Feedback

During doing this assignment, I got much involved in several search algorithms and have learned about them in detail. The game is a kind of challenge since unlike the example in course, the map is half-dynamic as it would change with each step taken by the snake. This kind of feature causes a few problems for us to design the algorithm. Besides, I felt quite confused when I could not get a good result. I used to have no direction on how to improve my algorithm, which is so frustrating.