Assignment 1

Name: Zhaoqing Teng NetID: zt414

In this assignment I implemented Bread First Search, Depth First Search, Deep Iterating Search, A* Search, Climbing Hill Search, Simulating Annealing, and two evolution search, one with mutation and the other one with crossover.
First to say I want to illustrate my evaluation function. There are two evaluation function and the first of them use the last game state and the current game state to evaluate the game state. The second one used only current evaluation function to evaluate. The first one use the shortest distance between the ghost and paceman, the shortest distance between the available pill and the paceman and the score to evaluate the state. The second function uses only the score and the number available pills to eat to evaluate the state.
For the first one, DFS, I use the thought which the Note supplies. It is use the function to calculate the possible state in the future and to see which one is the best. The weak point of this is that when there are few pill left, it is difficult to predict the path because all the path have no effect, as the score does not increase. Thus only the depth of the search is enough to know the location of the pills can we give a better performance. In my experiment, the best score of 20 times is 3860 and the average score is 2200.
For the second one, BFS, I use the Map to score the path to find the best state in the final level. It is a waste of time and space of course because I need the extra space to store the path. This method also use possible moves to predict the next state. And of course we need a depth limit to determine when to stop or it is a endless cycle. In my experiment, The best score of 20 times is 8020 and the average score is 3635.
For the deep iterating search, like the Depth First Search, uses the next possible move to predict the next state. The difference is that the depth is dynamic. I use a random function to determine the depth and I do not know if it will be better if I use an exact function to determine the depth. In my experiment, the best score of 20 times is 5010 and the average score is 3100.
For the A* Search, I use the priority queue to store the node which contains the state, last move and the layer. As I search the Internet, I choose the final state when all the pills are eaten or it enters the next level. The remaining pills is the estimated element to decide which state is better. And the priority queue stores the node according to the remaining pill. In my experiment, the best score of 20 times is 31980 and the average score is 14477.
For the Climbing Hill Search, I use the score to determine whether the state is good or bad. Using the class Random to generate the sequence and make little change in there to find the best state. In my experiment, the best score of 20 times is 5340 and the average score is 2180.
For the Simulating Annealing, like the climbing hill search, when the possibility larger than the temperature, I do not choose the change and still use the old value. And in my experiment, the best score of 20 times is 2700 and the average score is 1350.
For the evolution search, the first one I implement with mutation, and the second one with crossover. Use the random function to generate a series of sequence in the length of u + lambda and finally get the mutated sequence to see if it is better or not. For the first one, the best score of 20 times is 2690 and the average score is 1794.5. For the second one, the best score of 20 times is 2480 and the average score is 1900.
We can see from the result that the A* is the best because it is also choosing the state which is the mostly closest to the final state, which means that all of the pills are eaten. The weakness of the tree search is the goal state. Because I do not have a better evaluation function, so the only way to get the search better is the make the depth larger. But it also use more time and space. And the climbing hill function and the evolution search is also based a lot on the evaluation function which I do not find a better one. So in my implements, A* is the best search.