Today I will introduce a popular game called Pac-Man. Pac-Man is a real-time arcade game. It is my favorite game when I was young and I think it's a good environment for me to apply some algorithms to create an automatic Pac-man. So I choose it in this internship. Before we start, it is better to tell some rules of this game.

The player controls the main character through a maze, eating pills and avoiding the ghosts chasing him. The maze also contains some power-pills. When Pac-Man eats the power-pills, it leads to the ghosts’ scared time. During the ghosts’ scared time, Pac-man can eat them to obtain higher scores.

At each state, in case there are no walls in front of him, he has 5 possible moves: North, South, East, West and Stop. After his move, the ghosts will choose a random move in their possible moves. The game ends when there are no pills on the maze or the Pac-man dies.

The rewards after each move are:

Make move: -1

Eat pill: +10

Eat ghost: +200

Win: +500

Lose: -500

So let's start.

The basic step we need to do is that we have to teach the Pac-man to explore the maze without the ghosts. In this case, I choose depth first search, breadth first search and A\*.

Depth first search is an algorithm for traversing or searching tree or graph data structures. One starts at the root and explores as far as possible along each branch before backtracking

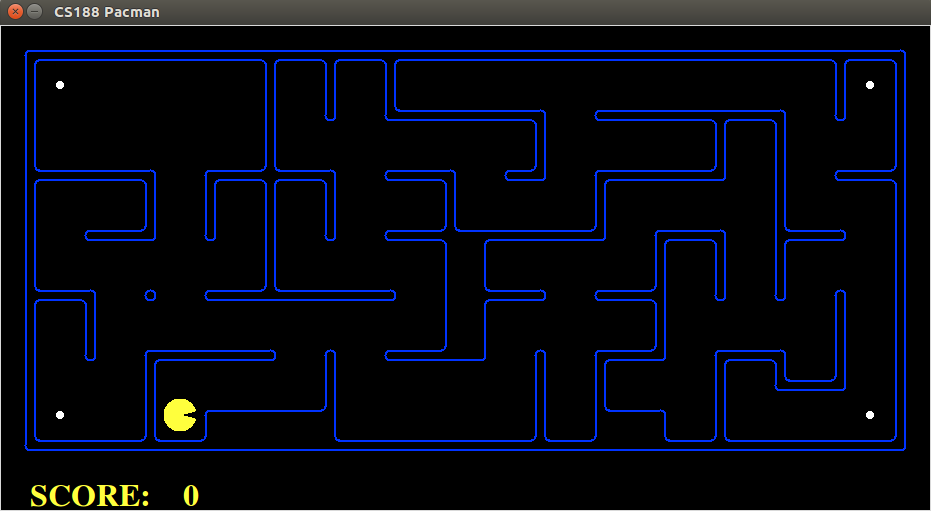
Here is a demo about dfs.

With breadth-first search, it explores the neighbor (same level) nodes first, before moving to the next level neighbors.

So you can see this video to understand about bfs.

In both cases, the goal state is there are no pills on the maze. The only difference is that the way Pac-man explores the maze. In dfs, he take more steps to reach the goal because he searched as far as possible. Although he explored less nodes than bfs algorithm, his path is not optimal. So if I want to get higher score, I should choose bfs for this game.

Then I try to apply them to a more complex maze. With the same goal, our maze now has 4 pills



In this case, the nodes expanded are about 2400. It is acceptable for a simple maze. But I scare that it will take a long time for complex maze, so I try to improve it by using A\*.

A\* is an algorithm which search among all possible paths to the solution, and among these paths it considers the one that has lowest cost to expand first. And the cost will be evaluated by the following function:

f(n) = g(n) + h(n).

where n is the last node in the path, g(n) is the cost from the start node to n, and f(n) is a heuristic that estimates the cost of the cheapest path from n to the goal.

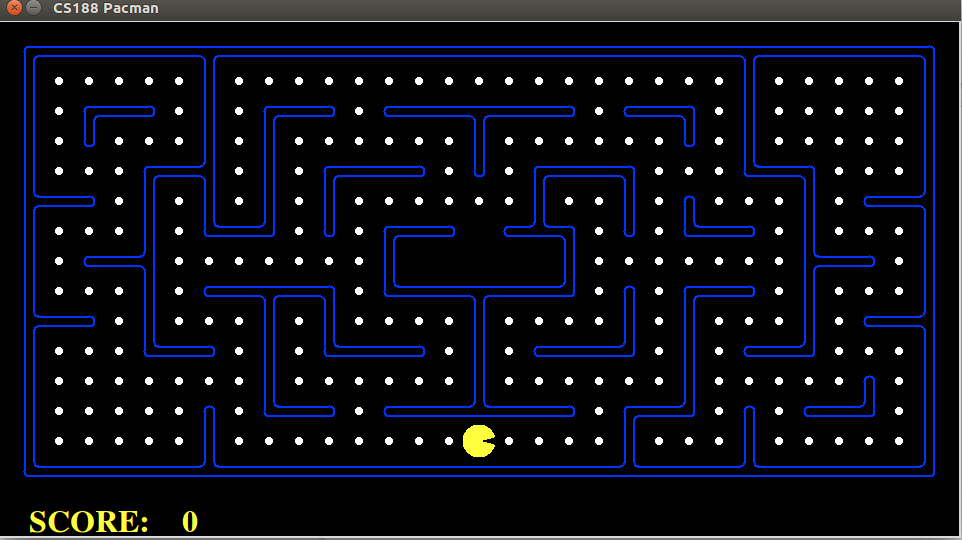
[

In this case, my heuristic function is calculated the manhattan distance from current position to the closest pill plus the distance from that pill to another closest pill. Repeat it until there is no pill left on the maze.

]

An this is the result of A\*, it find the optimal path as breadth first search but it expands just about 900 nodes compare to 2400 nodes of bfs.

After finding a good algorithm for search path, I try to apply A\* to a normal maze.

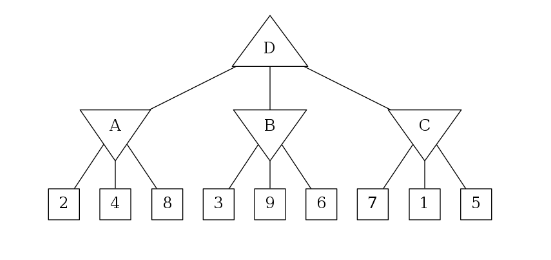


Although I have changed many heuristic functions, it still takes lots of time for Pac-Man to find an optimal path. So I decided to find a sub-optimal path with less time to calculate.

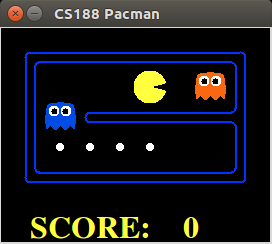
My strategy is a greedy search that the Pac-man will always eat the closest pill.

This is the result with that strategy (video).

After having some basic knowledge, I try to deal with ghost in maze. And with normal search, I realized that I do not have enough information about ghosts’ move. So I use minimax algorithm to handle this problem.



Minimax is an algorithm which you always choose the best action based on opponent’s actions. It means sometimes you have pessimistic thinking that the opponent will choose the move which lead you to the worst situation. Here is an example of this problem.



–-- (video)

With depth 3, because you are pessimistic, you will think these ghost will go toward and eat you. So the best case in this situation is that you will go to suicide. It’s reasonable because after each move, you will got -1 score. So the faster you die, the higher score you get. ^^

I also try to get out of this situation by implementing expectimax algorithm. It mean instead of thinking pessimistically, you hope for a bright future. And it works in this case. The Pac-man will assume that the ghosts will not go toward him, so he can follow the ghosts to get some pills and win the game.

--- (video)

Because the ghost move random, so you will have 50% to win. It’s the difference between minimax and expectimax algorithm.

Then I apply minimax with depth 3 to a mini maze so see whether this algorithm can solve the maze. And here is the result.

--- (video)

Sometimes it got stuck like this, maybe it’s because my evaluation function is not good enough.

My current evaluation function is:

….

So it is reasonable for him to stop in front of the pill, because in that case, the manhattan distance is 1 compare to 10 after he eats the pill. And he has to wait the ghost comes to him so that evaluation affects his behavior. I will investigate to find a better evaluation function for this algorithm later.

Now I will introduce a new algorithm to improve my Pac-man called Q-Learning.

Q-Learning can be used to find an optimal action-selection for any given states.

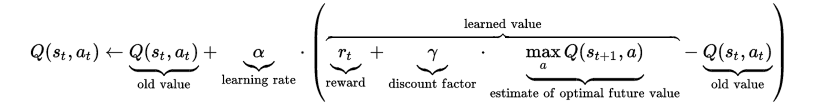
Assume in each state S, there are 3 things to be considered:

- Set of actions (per state) A.

- Reward (R(s,a,s’)) after executing an action a in A.

- New state (s’) after executing that action.

Before learning has started, Q(s,a) has an initial fixed value chosen by the designer. And after an action is executed, Q(s,a) is updated as the following:



And how an action is chosen among set of actions?

So we have to choose base on the following equation:

a = maxarg( Q(s,A) )

The action that is optimal for each state is the action that has highest long-term reward.

After having basic knowledge, I try to apply to some basic problems.

First is the Copy game on gym.openai.

This game has only 20 states (I normalize it to number from 0-19) and 6 actions (A-F be 0-5, respectively) for each state.

So I only need 10.000 training episodes to solve this problem.

Now I change to a more complex game. Mountain-Car.

Why