

Snake game using A* search and Q learning

Abstract: snake games normally have been created with just inputs from players to play the games and for basic entertainment. I intended to create a snake game that just plays by itself and can choose the shortest path to its next food (apple). After learning many search algorithms in AI class, I have an inspiration that I would like to experiment the snake game with a couple of search algorithms. At the end I would like to see which algorithms perform better for the snake.

Key words: A* search and Q learning

I. Introduction

When I searched online to find a snake game to play, I mostly found snake games written in C++ or PYTHON. What I noticed is that it is just a game written that takes users input to play the game. What if I can create a game that intuitively plays by itself and is smart in making decision on its own for shortest path to reach its destination (apple). For these experiments, I would like to use two algorithms in particular: A* search and Q learning.

II. Methods

To enable me to do the experiments, I would need to build the program that would use the two algorithms and my preferred programming language is C++.

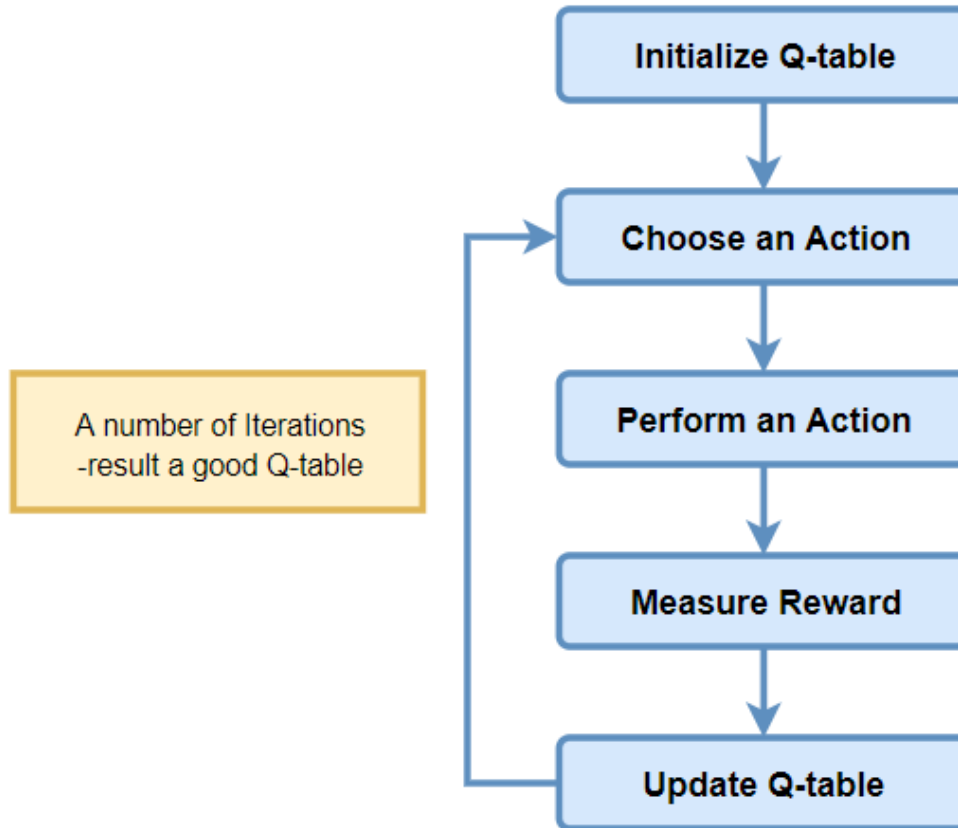
There are two algorithms will be used in the game. The first algorithm is A* search. I would like recap what A* search is and how it works. A* search algorithms is one of the most popular algorithms and it can be used to find optimal path between two nodes. It chooses its next move based on the lowest value of the function called 'f'. $f = g + h$, where g is the cost from initial state to the current state and h is the cost from the current state to the goal state [1]. A* search is also known to be optimal and complete.

The last algorithm is Q learning, which is also known as reinforcement learning. Q learning is a way of learning using experiences. We use our past experiences to make better decision for the future. For example, training dogs to perform our desired actions, it could be challenging. However, if we give them treats, the dogs will learn to do what we want them to do better. This is all what reinforcement learning about, learning from experiences and act better next time. To simplify, Q learning involves the following process [2]:

1. Monitoring the environment
2. Making a decision and applying a strategy
3. acting appropriately
4. Getting a reward or a punishment

5. Using the lessons learned to improve our approach
6. Iterate until you find the best course of action is found

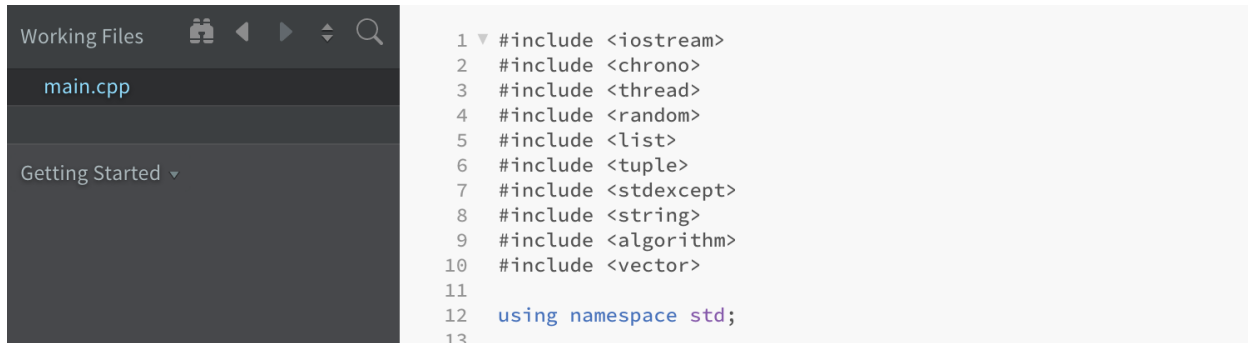
Q-learning Algorithm Process



(Image source: Towards data science [2])

The algorithm keep iterating till the best strategies are found.

Now that we understand the two algorithms, the game is ready to be built. The game was implemented in C++. Below are all the C++ libraries needed for building the game.



```
1 #include <iostream>
2 #include <chrono>
3 #include <thread>
4 #include <random>
5 #include <list>
6 #include <tuple>
7 #include <stdexcept>
8 #include <string>
9 #include <algorithm>
10 #include <vector>
11
12 using namespace std;
13
```

This is the code for A* search algorithm.

```
392 void Snake::calcAStar()
393 {
394     list<tuple<int, int> > open_list = {};
395     list<tuple<int, int> > close_list = {};
396
397     tuple<int, int> temp = make_tuple (snake_x, snake_y);
398
399     open_list.push_back(temp);
400
401     while (open_list.size() != 0)
402     {
403         int lowest_val = h_board[get<0>(getTuple(open_list, 0))][get<1>(getTuple(open_list, 0))];
404         tuple<int, int> lowest_tup = getTuple(open_list, 0);
405
406         // get the lowest value state on the list
407         for (int i = 0; i < open_list.size(); i++)
408         {
409             if(lowest_val >= h_board[get<0>(getTuple(open_list, i))][get<1>(getTuple(open_list, i))])
410             {
411                 lowest_val = h_board[get<0>(getTuple(open_list, i))][get<1>(getTuple(open_list, i))];
412                 lowest_tup = getTuple(open_list, i);
413             }
414         }
415
416         open_list.remove_if([&lowest_tup](tuple<int, int> elem){
417             return get<0>(elem) == get<0>(lowest_tup) && get<1>(elem) == get<1>(lowest_tup); });
418
419
420         tuple<int, int> down_child = make_tuple (get<0>(lowest_tup)+1, get<1>(lowest_tup));
421         checkChild(down_child, lowest_tup, open_list, close_list, 's');
422         tuple<int, int> up_child = make_tuple (get<0>(lowest_tup)-1, get<1>(lowest_tup));
423         checkChild(up_child, lowest_tup, open_list, close_list, 'w');
424         tuple<int, int> right_child = make_tuple (get<0>(lowest_tup), get<1>(lowest_tup)+1);
425         checkChild(right_child, lowest_tup, open_list, close_list, 'd');
426         tuple<int, int> left_child = make_tuple (get<0>(lowest_tup), get<1>(lowest_tup)-1);
427         checkChild(left_child, lowest_tup, open_list, close_list, 'a');
428         // add current state to closed list
429         close_list.push_back(lowest_tup);
430         if(board[get<0>(lowest_tup)][get<1>(lowest_tup)] == -10){
431             break;
432         }
433     }
434 }
435
436 }
```

This is the code Q learning. First Q table is declared. Then it is initialized to 0. Next it is filled with best Q values after a good number of iterations. This means it is ready to be used by the snake to get to its goal (apple).

```
21 int board[12][12];
22 //int h_board[12][12];
23 float q_table[12][12][4], mx, epsilon, discount_factor;
24 char path_board[12][12];
25 bool up, left, down, right, exit, eaten;
```

```
65 for (int i = 0; i < board_height; i++)
66 {
67     for (int j = 0; j < board_length; j++)
68     {
69         for (int k = 0; k < 4; k++)
70         {
71             q_table[i][j][k] = 0;
72         }
73     }
74 }
75
76 }
```

```
174 void Snake::updateQTable(void){
175     int reward = 0;
176     if(eaten) reward = 100;
177     if(exit) reward = -500;
178     if(board[snake_x][snake_y] > 0) reward = -300; //We ate ourselves
179
180     //Add q equation
181     q_table[psnake_x][psnake_y][action] += epsilon * ((reward + (discount_factor * mx)) -
182     q_table[psnake_x][psnake_y][action]); //Need to add Q equation with epsilon and reward
183     values
184 }
```

Now the snake is set to play the game.

III. Experiment and Result

I allow the snake to start from any position in the board but eventually it will start from every position. From a given starting position, I will keep track of how many steps the snake takes to get the apple. In A* search algorithm (based on AStar_Result.txt file), it shows that, it does not matter how many steps or how far the snake is from the goal, it always gets 1 point (reaching the apple from any position). However, in Q learning algorithm (based on Qlearning_Result.txt file), it does not matter how many steps the snake takes, it gets 0 point. It will take a while for the snake to earn and gradually get a point. From the results, A* search overtakes Q learning.

IV. Conclusion

At the beginning, I assumed that Q learning algorithm will blow A* search algorithm. However, after the testing, it showed that A* search outperform Q learning. A* search consumes less steps comparing to Q learning, yet it always gets a point while Q learning get 0 point even it takes max steps (10). In short, A* search is better than Q learning when implemented in snake game.

V. References

- [1] Geeksforgeeks <https://www.geeksforgeeks.org/a-search-algorithm/>
- [2] Towards data science <https://towardsdatascience.com/a-beginners-guide-to-q-learning-c3e2a30a653c>