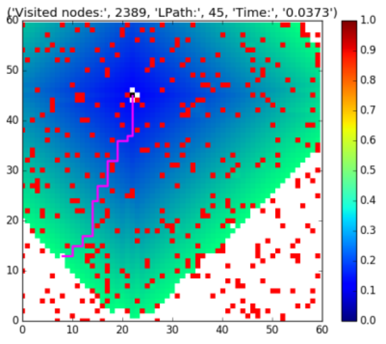
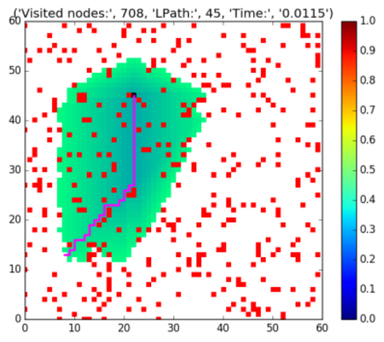
Task1:

In this task, some special lib of python such as heappush and heappop has been used and in order to simplicity an object has been designed that is called Node. It takes some variables such as Position of node, its cost and priority (f = c + h) and heuristic function type as inputs and initialize each node with these parameters. This class has some methods: nextMove, updatePriority, estimate and comparison method for priority queue. To implement, Start point has been considered as first node and has been expanded its neighbors using defined two auxiliary vectors [1,0,-1,0] and [0,1,0,-1]. Therefore, for each node there are maximum 4 neighbors. First of all is generated neighbors for start point and are calculated their priority according their cost and their distance to goal using given-heuristic function then is pushed them to heap after that in the next sequence is necessary to open a new node among pushed nodes according to the minimum value of each opened nodes that has been pushed in heap. Heappop command is used for reading from heap according to low priority it helps us to implement A* algorithm easily. To find the shortest path, a matrix has been considered for opened nodes that saved the position each node parent it helps us to find the path when the goal is found. To evaluate each heuristic function has been implemented a counter which count the number of opened node and a counter that count the number of steps found path from start to finish point. Also with help of time lib, calculation time to obtain the path has been measured.

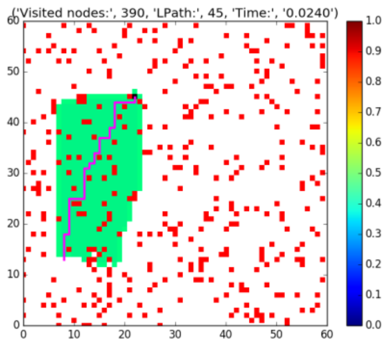
This algorithm has been test for 3 different heuristics function: Chebyshev, Manhattan and Euclidean and has been compared with the state which there isn't any heuristic function.



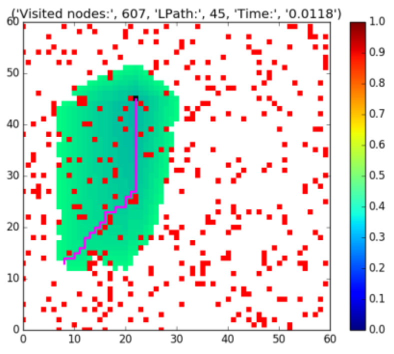
a) None, Visited Nodes:2389 , Time:0.0373



b) Chebyshev , Visited Nodes: 708, Time:0.0115



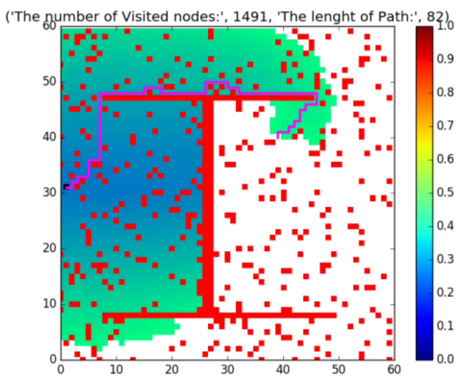
c) Manhattan, Visited Nodes:390 , Time:0.024

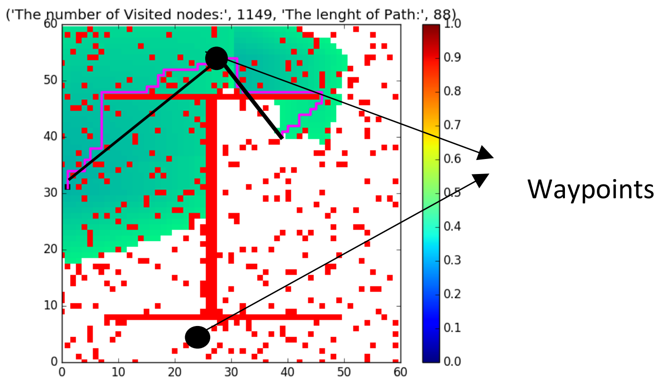


d) Euclidean, Visited Nodes:607, Time: 0.0118

As it has been shown, with Manhattan heuristic obtains minimum visited nodes but Chebyshev takes minimum time to find the path.

For H obstacles problems, two waypoints have been considered in the map. One is the top-middle of map a little bit above of the given-top position and the other is the given-bottom position.

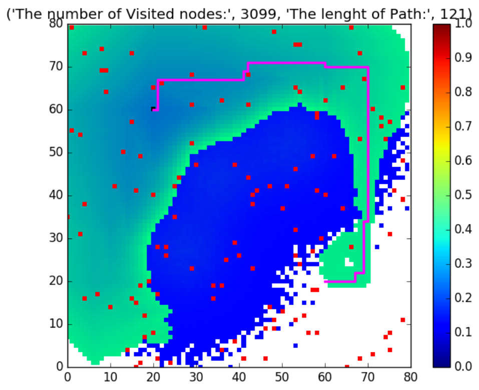


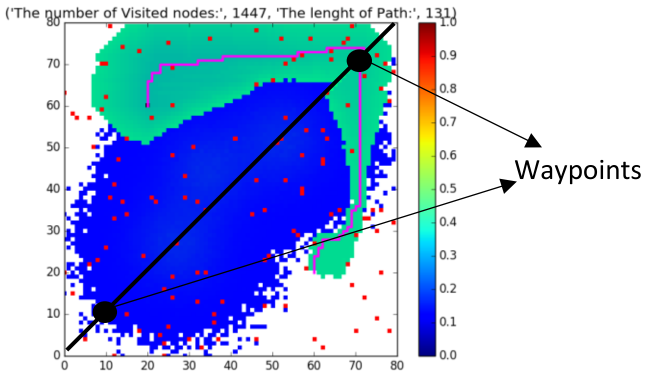


a) Manhattan heuristic b) my heuristic function with using waypoint

To solve this problem, first of all, the distance of start to finish point through two waypoints are computed then is made decision that top path to be selected or the bottom path.

To solve swamps problem, the same strategy has been used so that two waypoints has been considered a little above and bottom of the center of given-holes positions and the similar of H problem, the distance of between start and finish point has been measured and is made decision that which path is shorter.





a) Manhattan heuristic b) My heuristic using waypoints

If the point is above of the virtual line between two waypoints the distance between the point and finish point is computed through the way point d = dis(p,w)+dis(w,F) otherwise the distance is measured directly.

Task2:

To implement this part, two objects have been designed agent and node. The agent is an object that is used for defining each player. It saves current hand, total money, strength of hand and action of each player. A node class has been designed for implementing A* algorithm like task1. The node records index of each node and parent of it and calculates cost of each action. In this case cost is the bet of each player or cost of each call action.

An analyzer hand has been designed for this game in order to determining the strength of hand according to the given-strength table. At first, two collection variables have been defined numeral_dict and suit_dict. With using these two collections, the number and suit of hand are separated and counted then with regards to the number of each collection, the analyzer assigns a strength to each hand according to the given-table. Moreover, a rank hand is reported that shows rank of that hand. For example if your hand is one-pair it shows the number of pair (king or 10 or …). Rank has variable length with regards to your card strength for instance if your hand is two-pair so we have 3 ranks, one for each pair and one for the last card. It helps us to implement Show down function and identifies the winner.

A game class has been defined to implement the game and handle some important variable such as the number of play, the value of pot and so on.

For part B, a simple fixed agent has been implemented so that my agent always bet 5 and waits for the response of player2 if player2 rises bet then my agent calls and is shown down their hands. With this strategy, my agent loses the game. (Pokerstrategy_PartAB.py)

For part C, A* algorithm has been implemented like task1 with some changes in node definition and generating of children of each node. (Pokerstrategy_PartC.py)

In this part, heuristic function has been assumed as zero so this algorithm has been converted to breadth first search.