Pathfinding with A* and Greedy Algorithm
Minimax with Alpha-beta Pruning

CISC352 Winter 2020
Assignment 2
Group: 8

Student Name	Student Number
Hanyi Li	20057296
Tong Liu	06236306
Yitong Liu	20028039
Tao Ma	20060593
Yuntian Shan	20057524

Introduction

Pathfinding

In the implementation of finding a path, a mxn input grid is generated. The simulated grid contains a start position labeled ‘S’, goal position labeled ‘G’, obstacles labeled ‘X’, open area labeled, and the path that we find labeled ‘P’. Two algorithms are used, greedy algorithm and A* algorithm separately, where greedy always select next node of path with the lowest heuristic value and A* combines the heuristic value and the cost the path has gone so far. The heuristic function we chose here is Chebyshev, which is max(|q1 - p1|, |q2 - p2|) and it is admissible for both kind input grids. There are two different input grids generated, one input grid can go all direction and another one can go all directions except diagonal. Thus, we implement two input grids with two different algorithms.

Minimax

Minimax is a decision tool used to evaluate deterministic, turn-based, zero-sum two player games. Given a tree or graphical representation of a certain game state, and in combination with some other heuristic to obtain local evaluations at each leaf node, the Minimax algorithm is able to evaluate any position to produce a score for the current player. While such an exhaustive search is feasible for simpler games, it is often not viable for more complex games such as Chess or Go. The large number of possible moves in each game position leads to an “explosion” in the size of the search space as the search depth increases. Alpha-beta pruning is a specialized graph traversal algorithm which can be used to decrease the search space of a Minimax evaluation. This document describes one such implementation of a Minimax evaluation with alpha-beta pruning.

General Algorithms

Pathfinding

The general algorithm used in path finding for any input grid is shown below:

1. Find the start position and the goal position for the input grid
2. Start from the start position
3. Put the start position into notokSpace
4. If the okSpace is empty or the goal is found, stop and return
5. Else if the current node is not goal or the okSpace is not empty
    a. Generate all the possible neighbors of the current node
    b. Choose the node with lowest cost
       b.i Using greedy algorithm: choose the node with lowest heuristic value (return by heuristic function)
       b.ii Using A* algorithm: choose the node with lowest sum of heuristic value and the cost of past path
    c. Put the best node chosen into notokSpace and remove it from okSpace
6. Generate the complete path according to the nodes found

Minimax

The general algorithm used in the alpha-beta pruning of a Minimax evaluation search tree is shown below:

1. Construct a graphical (tree) representation of some game state and determine whether the evaluation is for the maximizing or minimizing player
2. Calls alphabeta(root, alpha = MIN, beta = MAX):
3. If the node is a leaf, return the static evaluation given in the tree
4. Else if the node is maximizing, recursively call alphabeta() on each child node
    a. For each child node, compare the evaluation with the alpha value
    b. If that child has a higher alpha value, then set it as the new alpha

    c. Prune all remaining children if the current alpha value is greater or equal to beta
    d. Return the alpha value
5. Else if the node is minimizing, do Steps 4(a through d) but instead minimize the beta value and by returning the beta value
6. Once all recursive calls are resolved, the final value returned is the evaluation of the root node

Implementation

Pathfinding:

`Class node:`

A data structure used to represent a node in the map. It has an attribute parent to indicate the previous node that the algorithm selected in the path. The attribute x and y indicate the coordinate of this node in the map. The attribute distance indicates the distance of the path so far leading up to this node.

`find_path(maps):`

This function contains the main loop used to find the path. The function starts from the start node and explores the nodes around it each time. It will keep looping until it reaches the goal.

`extend_path(node, maps):`

This function is called in find_path(maps) to explore the nodes around a given node. The function will try all the nodes around the given node and append the useful nodes to a list to store them.

`get_best:`

This function is called by find_path(maps) when the A* algorithm is used. It calculates the fValue (g + h, g is the cost of the path so far. h is an underestimate distance of node to goal) of all the nodes inside the list generated by extend_path, and then selects the node with the smallest f value.

`get_best_greedy:`

This function is called by find_path(maps) when a greedy algorithm is used. It will only consider the fValue as h, which is an underestimate distance of node to goal. The function will select the node with the smallest fValue.

`get_cost(p, new_x, new_y):`

This function defines the cost of moving from a node to another. The cost of moving diagonally will be larger than moving horizontally or vertically.

Minimax with Alpha-beta Pruning:

`Class node:`

A data structure used to represent a node in the tree. A node has two boolean attributes indicating whether the node is max or min, and whether it is a leaf. Leaf nodes must also be initialized with a numeric value indicating the static evaluation of that node. Max/min nodes must be initialized with a list of child nodes.

`parse_line(string):`

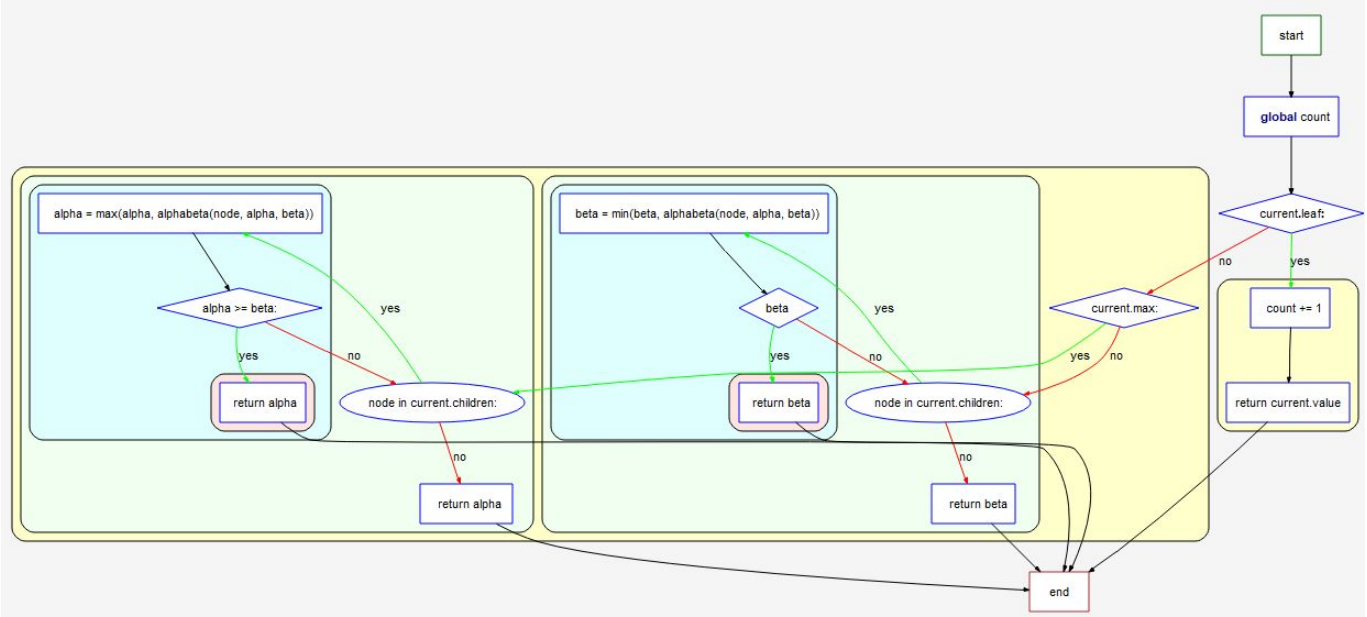
This helper function is used to convert one line from the input file into a tree data structure. The python “regular expression” module is used to parse the pre-formatted text input. The tree is stored as a dictionary of {Node name: node instance} pairs.

Input: string

Output: root node of tree

`alphabeta(node, alpha, beta):`
The main function used to traverse a tree representing the search space of Minimax. The generalized algorithm alpha-beta pruning algorithm is summarized above in the General Algorithms section. The control flow of this function is summarized in Figure 2 below.
Input: current node, alpha, beta values
Output: float indiecating the evaluation of the inputted node

Figure 2: control flow for alphabeta() function



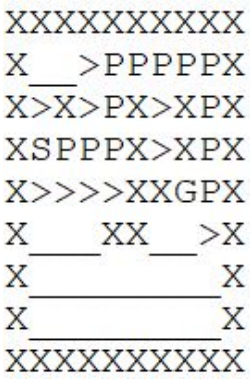
Results

PathFinding:

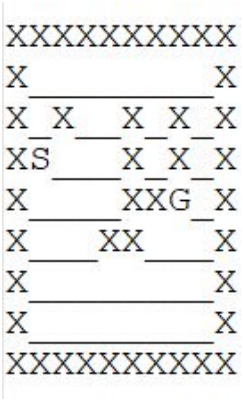
An input grid that can go all directions except diagonal:

```
XXXXXXXXXX
X________X
X_X__X_X_X
XS___X_X_X
X____XXG_X
X___XX___X
X________X
X________X
X________X
XXXXXXXXXX
```

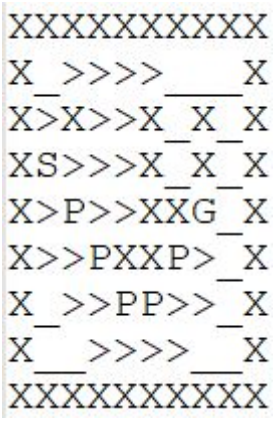
A* algorithm: Greedy algorithm:



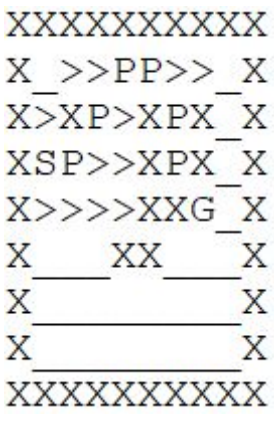
An input grid that can go all directions:



A* algorithm:



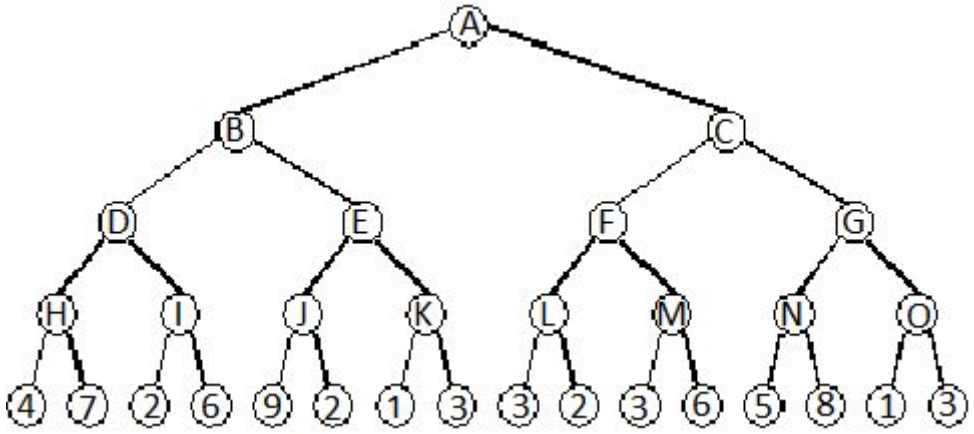
Greedy algorithm:



Minimax:

In the case of the example input for a tree with a total of 8 leaf nodes, 2 nodes (25%) were pruned when the alpha-beta pruning algorithm was applied. In another example with a total of 16 leaf nodes, 4 nodes were pruned when Minimax was executed with alpha-beta pruning.

{(A,MAX),(B,MIN),(C,MIN),(D,MAX),(E,MAX),(F,MAX),(G,MAX),(H,MIN),(I,MIN),(J,MIN),(K,MIN),(L,MIN),(M,MIN),(N,MIN),(O,MIN)}
{(A,B),(A,C),(B,D),(B,E),(C,F),(C,G),(D,H),(D,I),(E,J),(E,K),(F,L),(F,M),(G,N),(G,O),(H,4),(H,7),(I,2),(I,6),(J,9),(J,2),(K,1),(K,3),(L,3),(L,2),(M,3),(M,6),(N,5),(N,8),(O,1),(O,3)}



Conclusion:

Pathfinding:

In our results of the path we found, we notice that the paths find by using A* and Greedy are the same, no matter what kind of input grid it is. Although the path they found is the same from the results we can see that the attempts tried to find the path to goal are different for those two algorithms, which means their search space is different. Therefore, the paths found by these two different algorithms might be different if the input grid changes.

Minimax with alpha-beta pruning

From the evaluations using alpha-beta pruning, we noted a significant reduction in the size of the search space traversed by Minimax. Using balanced binary trees , we found that roughly 25% of the nodes were pruned. However, this is likely an under-representation of the actual improvements in efficiency over a brute-force search of the whole problem space. In more complex games, there will often be more than 2 legal moves in any given game state. With a greater degree of branching in the tree, the relative number of nodes which become pruned will also be greater (compared to the binary trees used in this assignment).