Assignment 2: Sokoban Game Trees

Created By: Faiz Chaudhry

Due on: DUE-DATE

Introduction:

For this assignment we revisit our friend Sokoban, but instead on focusing on purely state searching we will be discussing game trees. As you have already seen in lecture game tree searching is yet another way for AI to work at solving puzzles. This time when you tackle Sokoban you will see that the player now has enemies he will need to avoid while completing the level. Due to the nature of game tree algorithms the goal to complete levels in this assignment has been changed to simply reaching the switch tile on the map. This is because solving box based puzzles via game tree algorithms would be much more difficult to do.

Overall this assignment gives an introduction to agents within Sokoban and how they can influence the game. By using game trees and agents we can develop AI which pay attention to the state of the game and move accordingly. Now let's get solving!

Installation:

In order to solve the assignments you will need to install Python 3 along with the Tkinter library (GUI) and TkThread library (for TKinter multithreading). You can install Python 3 through your terminal if you're on unix or through an installer found here `https://www.python.org/downloads/release/python-374/`. To install the Tkinter and TkThread libraries we recommend downloading and using python pip:

- pip install tkinter
- pip install tkthread

Files to Edit:

- game_tree_searching.py

- evaluation_functions.py

Both files can be found in the game_trees directory.

Question 1:

First up you will be implementing the *Minimax* algorithm as seen in lecture. Recall that this algorithm is implemented via recursion and alternates picking between the largest value and the smallest value. Note that in lecture the algorithm is presented with one opposing agent in mind however there can be multiple opposing agents here so keep that in mind. Write your implementation under the *minimax_search* method found in *game_tree_searching.py*. Similar to assignment 1, you will find the *GameStateHandler* helpful for this question and the next two.

Question 2:

For this question you will write the *Alpha Beta Pruning* algorithm. This algorithm is similar to the *Minimax* algorithm above except now we prune parts of the game tree which we know that the max/min agents will not consider. Once again recall that more than one opposing agent can be present on the map at a given time and they should all be accounted for in your solution. Implement your solution under *alpha_beta_search* found in *game_tree_searching.py*.

Question 3:

Next you will implement the last of the three game tree searching algorithms, *Expectimax* search. As you know, expectimax uses a slight amount of probability to determine which move an opposing (min) agent will take. However, to make things simple you will be assuming a uniform distribution for all opposing agents. In other words, all actions of an opposing agent have the same probability of being chosen. Implement you solution under *expectimax_search* in *game_tree_searching.py*.

Question 4:

In the next two questions you will be writing two different evaluation functions. To start, you will write the box only evaluation under *box_evaluation* found in *evaluation_functions.py*. This evaluation function will consider scenarios in which there are only boxes to help you complete the level along with opposing agents to prevent you from doing so. You will want to use inverses in these evaluation functions to ensure that measurements such as shortest distance give a positive effect on the evaluation. Another thing to consider is weighing the parts you are evaluating. For example, you may want to avoid an opposing agent less often than moving towards a box and so you would weigh the two components

accordingly. **Note: There will be no armory points on the map when dealing with this scenario**.

Question 5:

Lastly, you will write an evaluation function for scenarios in which only **armory points** exist on the map (and consequently only one switch) as well as opposing agents. All the mentioned considerations above remain for this evaluation function. Write your implementation under *points_evaluation* found in *evaluation_functions.py*. **Note: There will be no boxes on the map when dealing with this evaluation function**.

Testing:

To test your solutions we provide a partial autograder. This autograder **should not** be treated as your final mark as there will be more tests involved during actual marking. To run the autograder simply type 'python autograder.py' or 'python3 autograder.py' (depending on how your environment variables are setup).

Inspiration and References:

The structure of this assignment and some questions are inspired from those found in the assignments provided to students in CSC384 during 2018 (last year). This assignment follows the course material (specifically the lecture slides) of CSC384 during 2018. As the CSC384 course and assignments from 2018 reference concepts taught to students in a similar AI course found at Berkeley, I find it important to include Berkeley as a reference for these assignments as well. To see more information on Berkeley's AI course visit `http://ai.berkeley.edu`.