HW 2 Informed Search Agent

Needs:

Functions –

1. Examine a GameState and give it a score [-1.0, 1.0]
   1. Additional criteria for getting score
   2. Always return…
      1. -1.0 = enemy won
      2. 1.0 = AI won
   3. Look up win condition states? In AIPlayerUtils…
      1. getWinner(currentState) determines who won…
         1. But the function does not evaluate/print -1.0 or 1.0 respectively given
   4. Number of ants
      1. (our ants – their ants)/(sum of all ants)
      2. Do comparison on worker, soldier and overall total
   5. Soldier moving/targeting
      1. If solider movement is closer to queen, give positive points
   6. Queen moving/targeting
      1. If queen movement is further from solider/drone/ranged ant, give positive points
   7. If worker does not have food
      1. If worker movement is closer to food source, give positive points
   8. If worker does have food
      1. If worker movement is closer to anthill/tunnel, give positive points
   9. Food Stored
      1. Start… (our food – their food)/(sum of all food)
      2. Modify… exponential where x => [0, 11] and y => [0,1]
      3. Favors collecting food over building additional ants
2. Helper method to evaluate a list of nodes and determine their overall evaluation score
   1. Average, highest, etc.
      1. Highest
         1. Add the evaluation score to the score of the parent node
            1. Easy to implement
            2. Wont be range [1.0, -1.0]
            3. Scores may be inflated based on how deep the current node is (more nodes to be added)
      2. Average
         1. Average the evaluation scores to be the (sum of the evaluation scores of the current node and its parents) divided by (total number of states evaluated [current depth+1])
            1. Will range [1.0, -1.0]
            2. Can compare nodes based on score no matter the depth
            3. Need to implement two variables to dictionary

Sum of evaluation scores to that point

Allows for child node to add its evaluation score to this total and then easily divide by (current depth+1)

Average

Number that will be used in determining which node to choose

1. Recursive method
   1. Inputs: GameState and Current Depth
   2. Steps:
      1. Create list of all possible moves from the given GameState
      2. Generate list of GameState objects that result from the possible moves
      3. If depth limit not reached… make recursive call for each state in the list (new GameState and increment depth limit +1)
      4. Use helper method to assess the score of the GameState
      5. If depth > 0, return the overall score… else return the Move object from the node with the highest overall evaluation score
   3. Use getMove() method to make the new move (outside of the recursive method)

Structure –

1. Design a data structure for a “node”
2. Member (self.) variable for depth limit
   1. Initially set to 1

Unit Test & Debugging –

Add code to the bottom of the file, outside of the class (no indentation) to create a unit test for each function as stated below…

1. Call each of the new methods at least once with a meaningful input
   1. Verify that the correct output is returned
2. If the test fails, print an informative error message to the console
   1. Else, do not print anything