Machine Exercise 02 – ADVERSARIAL SEARCH

REPORT | Dana Kathleen Redeña

Implementation of Algorithms

I. **Modules**
   a. Copy – used for duplicating the board configurations
   b. Math – used for infinity values - float("inf") and float("-inf")

II. **class State**
   a. class State represents each state in the Minimax Search
   b. Attributes
      i. alpha – alpha value in alpha beta pruning
      ii. beta – beta value in alpha beta pruning
      iii. parent – parent of current state
      iv. children – array of all the child states of the state
      v. depth – depth of state in minimax tree
      vi. cost – cost of the state (evaluated according to whether state is a minimizer or maximizer)
      vii. s_board – the board of the state
      viii. next – an array of tuples for the possible next moves of the state
      ix. coord – tuple of the taken move of the current state
   c. Set and Get Methods
      i. set_alpha(self, alpha) and get_alpha(self) – for alpha
      ii. set_beta(self, beta) and get_beta(self) – for beta
      iii. set_cost(self, cost) and get_cost(self) – for cost
      iv. get_parent(self) – for parent
      v. get_depth(self) – for depth
      vi. get_board(self) – for board
      vii. get_coord(self) – for coord
   d. Other Methods
      i. min_or_max(self) – returns 0 (max) or 1 (min), evaluated from self.depth%2
      ii. next_op(self, op) – method for handling the next (array of tuples); op == 0 will print the next array; op == 1 will return the array length; otherwise, the method will return and pop the element in the first index

iii. add_child(self, child) – appends the child state on the parent state's children array

iv. get_childsol(self) – gets the move that the AI will move after the minimax algorithm by checking the child that matches the cost of the root node. Returns the board and coordinate of the state.

III. Global Variables

a. board – a 3x3 2D array representing the tic tac toe board

IV. Functions

a. print_board(board)

- i. Parameters
  1. board – 3x3 2D array representing the tic tac toe board
- ii. Functionalities
  1. Outputs the tic tac toe board following the format in the specifications

b. is_win(board)

- i. Parameters
  1. board – 3x3 2D array representing the tic tac toe board
- ii. Functionalities
  1. Checks if the current board state results to a win (by returning the piece that won, 'x' or 'o') or a draw

c. is_filled(board)

- i. Parameters
  1. board – 3x3 2D array representing the tic tac toe board
- ii. Functionalities
  1. Checks if the tic tac toe board is already filled up by returning a True (otherwise, False). This function aids the evaluation of a game ending in a draw.

d. next_moves(board)

- i. Parameters
  1. board – 3x3 2D array representing the tic tac toe board
- ii. Functionalities
  1. Initializes an array that will contain all the tuples with coordinates for the blank cells in a board. The function returns the array.

e. next_move(move_char, board, coord)
   i. Parameters
      1. move_char – takes the character corresponding the move (‘x’ or ‘o’)
      2. board – 3x3 2D array representing the tic tac toe board
      3. coord – the coordinates of target move
   ii. Functionalities
      1. Changes the board according to the move being placed

f. descend(parent)
   i. Parameters
      1. parent – current state that will serve as the parent of the state it will descend to
   ii. Functionalities
      1. Duplicates the board of state (deepcopy) to provide as the board configuration of the child state. The function checks certain conditions before a descent (instantiation of a child state) and returns a False when a condition is meant (which will eventually trigger an ascent).
      2. Checks if the current state (parent) is a leaf node already (by checking is_win(board). If the state is a terminal node, the current state’s cost is changed according to whether it’s a win, lose or draw (then the function returns a False). The winning cost is evaluated by 10 – depth (depth of state, get_state()). The losing cost is evaluated by depth – 10. The cost of a draw is 0.
      3. Checks if the state’s alpha value is greater than or equal to its beta value, and returns a False if satisfied.
      4. Checks if the state’s next array has a length of 0 that means all possible children are checked already, and returns a False if satisfied. Otherwise, the function proceeds by popping the next array to get the next move of child.
      5. For the descent: If the state is a maximizer, a minimizer state is instantiated with a modified board simulating the ‘o’ move being made. The said new state will have the alpha value of the parent (beta is set to default, float(“inf”)) and its cost will be its beta value. Then

if the state is a minimizer, a maximizer state is instantiated with a modified board simulating the ‘x’ move being made. The said new state will have the beta value of the parent (alpha is set to default, float(“-inf”)) and its cost will be its alpha value.

6. The function then returns the child state.

g. ascend(child)
   i. Parameters
      1. child – current state that will serve as the child of the state it will ascend to
   ii. Functionalities
      1. Gets the parent of the current state (child.get_parent()). If the parent is None, this means the child is the root state already, with this the function returns.
      2. For the ascent: If the parent is a maximizer, the beta value of the child is checked to whether it is equal to float(“-inf”) and to whether the child’s beta value is less than or equal to the parent’s beta value. If it is either, the function simply returns without modification of alpha and beta values. Otherwise, the parent’s alpha value is set to be the child’s beta value and the parent’s cost is set to be its own alpha value.
      3. For the ascent: If the parent is a minimizer, the alpha value of the child is checked to whether it is equal to float(“inf”) and to whether the child’s alpha value is less than or equal to the parent’s alpha value. If it is either, the function simply returns without modification of alpha and beta values. Otherwise, the parent’s beta value is set to be the child’s alpha value and the parent’s cost is set to be its own beta value.

h. ai_move()
   i. Functionalities
      1. Duplicates the board for the instantiated root state. The function descends until an ascension is triggered by descend(parent) returning a False value. This continues until we reach back to the root state.

        2. The best board configuration to take based on the minimax algorithm along with the coordinates of the move it took is retrieved from root.get_childsol(). The board is then printed and the function returns the board.

i. human_move()
    i. Functionalities
        1. Takes the coordinates of the move that the human will take. The function loops until the inputted coordinates are valid. The valid move will be updated on the board.