Artificial Intelligence-Fall 2022

Project 02-Game

Ehsan Hosseini-Mohammad Afshari

Mohammad Afshari

Overview

Othello is a strategy board game for two players (Black and White), played on an 8 by 8 board. The game traditionally begins with four discs placed in the middle of the board as shown below. Black moves first.

Our duty in this project is to implement the game for machine play vs machine using minimax algorithm.

Mohammad Afshari

Phase one

In this phase, we will model the problem and implement the functions and general structure of the program that is supposed to be used in the algorithms.

We will explain about the modules and packages which are used in implementation.

- **Othello**
  This module contains the Othello class which could be said is the main part of the program in which the main logic of the program is implemented.

  This class has the following properties:
  - Board: saves the current state of the game
  - rows & cols: saves the number rows and columns of board
  - current_player: players turn

    - num_black & num_white:saved the number of black and white tiles

  This class has the following methods:

    - **actions:** this method is used for finding all possible moves with respect to game rules and uses two other functions called get_captured_tiles & get_captured_tiles_in_direction which counts the number outflanked tiles done by a taking a certain action and checking every direction of of the newly placed tile to check for any outflanking possible.

    - **is_out_of_bound:** this method checks if a certain position is possible or not with respect to board size.

    - **make_move:** this method is makes move and update number of tiles for each player and flip tiles

    - **successor:** takes in a move and returns a new Othello object representing the state of the game after making that move.

    - **is_terminal:** returns true if the game is over, and false otherwise

    - **utility:** this function is used for getting the game result which is done by comparing the number of tiles in equal case 0 is returned if the first player wins 1 is returned and otherwise -1 is returned.

    - **is_cut_off:** returns True if the search should be terminated at the current depth, False otherwise by using the is_terminal function.

    - **heuristic:** this method is replacement of utility method and used to estimates a state's utility

- **GameManager**

This module contains the GameManager class which is used to manage the game and run algorithms.

This class has the following property:

- **initial_state:** it is initialized with starting board game

This class has the following methods:

  - **start_game:** this method start the game and continues until reach the terminal state

  - **get_winner:** it is use to specify the winner of the game

  - **get_game_information:** this method displays some information about the number of tiles on the board and action of the current player.

- **Main**
  The flow of the program starts from here and all modules will be used here

ehosseini8001@gmail.com

Phase two

In this phase we will implement a minimax algorithm which we can see below.

Minimax:**to implement minimax algorithm we defined a function called minimax_decision as argument we take an Othello state & depth and returns a move.Minimax algorithm includes two other functions called max_value and min_value both functions return a best_value and best_move as value and move,these variables are defined within each function with respect to its purpose meaning min_value function searches for the lowest utility and therefore returns the lowest possible value and max_value function searches for the**

highest utility and therefore returns the highest possible value and each time max_value or min_value is called depth is decreased by one.

max_value:as arguments we take an Othello state & depth.first we check if the state is terminal or not which is done by the isTerminal() function of depth is equal to zero returns a tuple consisting of utility which is calculated by the utility() function of state and None.secondly we define two variables called best_move and best_value and set them to them minus infinity so as to find the highest utility.We call the Actions() function on the the current state and loop through every action possible and let the min_value return the lowest values of childs(possible actions or moves) and choose the highest among them and set the best_move and best_action variables accordingly.

min_value:just like max_value first we check for terminality and then we take the same steps only this time the best_move and best_action variables are set to infinity and choose the lowest utility possible among all the values return by the max_value called on all possible actions(childs) and set the best_move and best_action accordingly.

ehosseini8001@gmail.com

Phase three

Minimax algorithm with alpha-beta pruning

Minimax algorithm implemented with alpha-beta pruning is just like the base minimax algorithm with some minor differences to enhance time and space complexity ,for this algorithm we defined a function called minimax_decision_alpha_beta we take the same arguments as base minimax_decision function.This algorithm includes two other algorithms just like the base model called max_value_alpha_beta and min_value_alpha_beta,both take the same arguments only this time they take two other arguments called alpha and beta.

Alpha:alpha is the highest possible utility achievable by choosing all possible actions of each state, in other words highest utility among children of each state, which is used for maxPlayer.Alpha is set to minus infinity as initial value.

Beta:beta is the lowest possible utility achievable by choosing all possible actions of each state, in other words lowest utility among children of each state, which is used for minPlayer.Beta is set to infinity as initial value.

max_value_alpha_beta:it's the same as max_value function the only difference is that each time we want check an action we set the alpha value which is the maximum amount between current alpha and current best_value and check if the current best_value is equal or higher than the beta so as to avoid checking other branches.

min_value_alpha_beta:it's the same as min_value function the only difference is that each time we want check an action we set the beta value which is the minimum amount between current beta and current best_value and check if the current best_value is equal or less than the alpha so as to avoid checking other branches.

Mohammad Afshari

Phase four

heuristic_minimax_search():just like the **minimax_decision_alpha_beta**
We start with the maxPlayer but this time since we use the heuristic function we need to call the heuristic_max_value function.

heuristic_max_value():this function is just like the simple **max_value_alpha_beta**
Instead of checking for terminality we check for the depth limit and if we haven't reached the limit we stop the algorithm and return the heuristic function called on current state(to predict utilities).

heuristic_min_value():this function is just like the simple **min_value_alpha_beta**
Instead of checking for terminality we check for the depth limit and if we haven't reached the limit we stop the algorithm and return the heuristic function called on current state(to predict utilities).