Meltem Ceylantekin 28089

Ayşenur Güller 27796

CS 404 / Spring 2024
Assignment 3 - Hashi Game

1. Modelling the game

States:

- Grid that represents the current board:
  - Integers: {1,2,3,4}: islands
  - "0": unlabeled island
  - ".": empty cells, potential bridge
  - "-", "=", "|", "X": horizontal, vertical, double bridges
- Player turn (1 or 2)
- Scores for both players: calculated based on the number of completed bridges connected to each label.

Initial state:

- starting game board configuration as defined by the game. Typically has empty cells and labels placed on specific cells. There is no bridge at initial board.

Terminal state:

- All cells on the board are filled (i.e., no empty cells remaining)
- No legal moves are available for either player (according to the "get_legal_moves" function)

State transition function:

Takes a current state (S), a player (P), and an action (A) as input and returns a new state (S')

- Action(A): valid move for the current player (either placing a label or bridge)
- The function updates the grid in the state(S) based on chosen A, updates the player turn (P' = 3-P), and calculates the new scores for both players.
- The new state(S') will have the updated grid, player turn, and scores.

Payoff function:

Takes a terminal state(S) as input and returns a numerical value representing the utility for Player 1. In this case, the payoff function is the difference between Player 1's score and Player 2's score at the terminal state(S).

- Positive value: Player1 wins.
- Negative value: Player2 wins.
- Zero: tie.

2. **Implementation in Phyton using alpha-beta pruning**

alpha_beta_search Function:

This function implements the Alpha-Beta Pruning algorithm to search through possible moves and find the optimal move for the current player.
It takes three parameters: depth (remaining depth of the search tree), alpha, and beta.
If the depth is 0 or the game is over, it returns the evaluation of the current game state using the evaluate_state function.

If it's Player 1's turn (the maximizer):
It initializes max_eval to negative infinity. It iterates over all legal moves and evaluates each move recursively using the same function but with reduced depth. It updates max_eval with the maximum evaluation found so far. It updates alpha with the maximum evaluation found so far. If beta is less than or equal to alpha, it prunes the remaining moves in the loop. It returns the maximum evaluation found.

If it's Player 2's turn (the minimizer):
It initializes min_eval to positive infinity. It iterates over all legal moves and evaluates each move recursively using the same function but with reduced depth. It updates min_eval with the minimum evaluation found so far. It updates beta with the minimum evaluation found so far.

If beta is less than or equal to alpha, it prunes the remaining moves in the loop.
It returns the minimum evaluation found.

Pruning occurs when the algorithm realizes that further exploration of a particular branch will not lead to a better outcome, allowing it to skip unnecessary evaluations and improve efficiency. This happens when the minimizer finds a move that is worse than a previously explored move by the maximizer or vice versa.

get_computer_move Method:

This method is responsible for selecting the best move for the computer player (Player 2) using the Alpha-Beta Pruning algorithm.
It initializes best_move to None and best_score to positive infinity. It retrieves all legal moves. It iterates over each legal move: It makes the move on a copy of the game grid.
It calculates the score using the alpha_beta_search function with a depth of 3. It undoes the move. If the score is less than the best_score, it updates best_score and best_move. It returns the best move found.