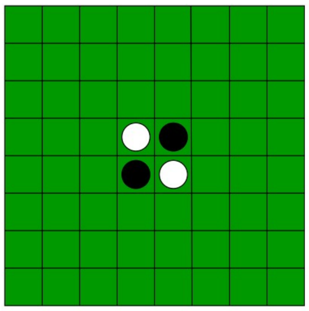
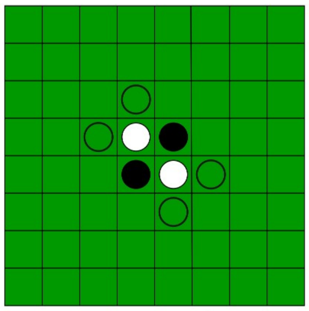
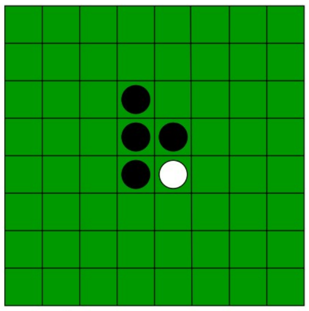
2 Intelligent Strategy for Othello

Although not very well-known, Othello is a very interesting strategy board game for two players. It is played on a 64-tile 8×8 board, starting with 2 white and 2 black pieces arranged in a diagonal manner as shown in the figure. The player with black pieces moves first. The rules are relatively simple, compared to the complexity of the game.

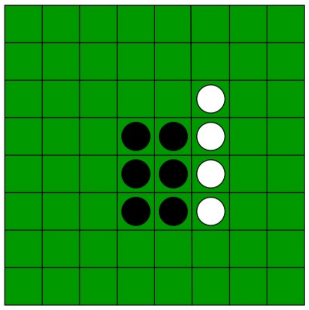
- In any turn, the player to move (for simplicity, say it is the one with black pieces) must place a black piece in such a way that there is at least one contiguous straight line (horizontal, vertical or diagonal) of white pieces between a black piece on the board and the new black piece being placed.

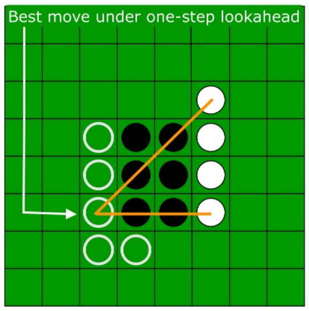

Initial board


Possible moves (black)

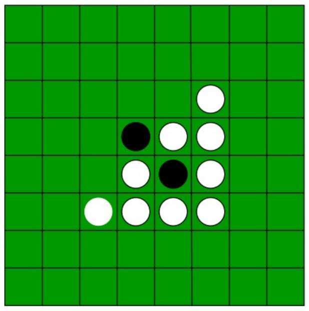

After move white flips

- After a black piece is placed, all the contiguous straight lines of white pieces between the newly placed black piece and any other black piece on the board will be flipped to black.


Board



Possible moves (white)


After move black flips

- After the move, the turn switches to the other player, who plays under the same rules but with colours reversed.
- If a player has no valid move, the turn is passed to the other player.
- The game proceeds until either the board is full or there are no valid moves on the board for both players. At this point, the player with higher number of pieces on the board wins.

One of the simplest and most intuitive approaches of building a good strategy for such turn-based two player games is look-ahead. In one-step look-ahead, from the current board position, the player looks at what the board will look like after all possible sequences of one move (your own), and based on some scoring function and tie-breaking rules chooses the best move. A simple scoring function for the player with black pieces is *number of black pieces minus number of white pieces on the board.* So, the aim of player with black pieces is to maximize this score and the opponent's aim is to minimize the score.

A generalisation to one-step lookahead is k-step lookhead strategy. To also model an intelligent opponent's moves, MiniMax algorithm was introduced. This can be viewed using a decision tree comprising of alternating layers of Max and Min nodes, thus the name. A demonstration of the MiniMax algorithm applied to 4-step look-ahead is given in the figure below.

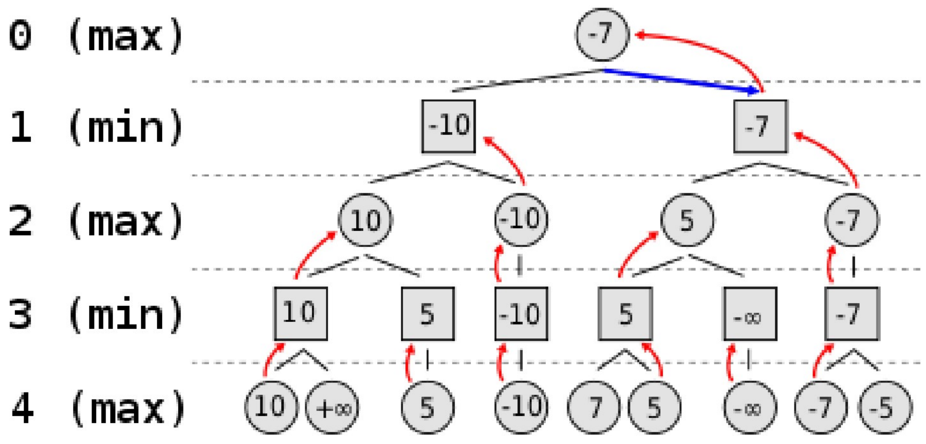


Figure 4: A depiction of 4-step look-ahead in a strategy game where each player has at most two strategies: left and right. Each of the circular nodes in the the decision tree is a Max node and the square nodes are Min nodes. After evaluating the scores upto a depth 4, the scores are propagated upwards based on the function in each node (min or max). At layer-0, i.e. current state, the action is taken which gives the maximum score, and thus the action leading to the right subtree is taken.

Coming back to Othello, your task is to implement an intelligent game-playing agent for Othello which uses k-step look-ahead and MiniMax algorithm to select its moves. For this task, you'll be given a class `Othello` with the following class variables:

- `int turn`: 0 if it is Black's turn to move and 1 otherwise.
- `int board[8][8]`: A 8×8 array to represent the current board state, using integers -1 for empty tiles, 0 for black pieces, and 1 for white pieces.
- `int winner`: -1 initially, 0 if black wins and 1 if white wins. In case of a draw (both players have equal pieces on the board), keep it as -1.

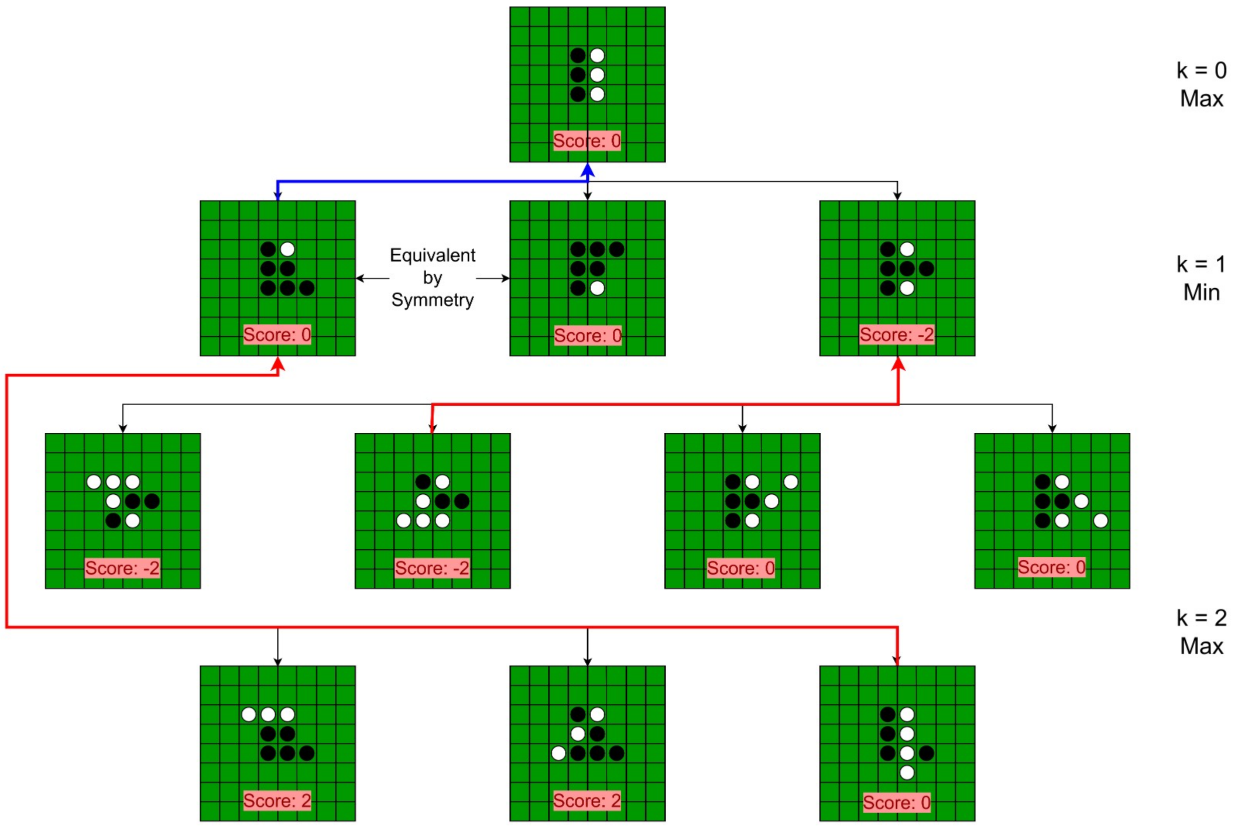
You will have to implement the following four class functions:

- `public Othello()`: Initializes the board and other class variables based on input given in file `input.txt`. The input will contain 9 lines, first line would contain 1 integer representing the turn. The next 8 lines will contain 8 space separated integers from $\{-1, 0, 1\}$ representing the board starting from `board[0][0]` to `board[7][7]`, row-wise.

- `int boardScore()`: The scoring function for evaluation of boards at the maximum depth of the decision tree. This will be a simple difference of the number of pieces on the board of the player whose turn it is and the opponent's pieces. To clarify, when `turn = 0`, this will be `num_black_pieces - num_white_pieces`. Similarly, when `turn = 1`, this will be `num_white_pieces - num_black_pieces`.

- `int bestMove(int` k`)`: Using k-step look-ahead from the current board, return the best move for the player whose turn it is as an integer. If the best move is to place the piece at `board[`i`][`j`]`, return $(i * 8 + j)$. To break ties, return the smallest $(i * 8 + j)$ value with the maximum score.

- `ArrayList<Integer> fullGame(int` k`)`: Assuming that both players are using k-step look-ahead starting from the current board, return a list of moves encoded as integers of how the game proceeds. Update the board accordingly and set the winner at the end.



A demonstration of 2-step lookahead for Othello with difference scoring function