Helper Recipe Name: next_board

Inputs:

board, the current board configuration

player, either -1 or 1, with -1 representing the red player's turn and 1 representing the black player's turn

Outputs:

possible_boards, a sequence of possible board states from the current *board*

Steps:

```
1. Initialize possible_boards as an empty sequence
2. Initialize new_board as a copy of board
3. If player is -1, then
    a. For each row, a number between 0 and 7 inclusive,
        i. For each col, a number between 0 and 7 inclusive,
            1. new_board ← a copy of board
            2. If $board_{row, col}$ is equal to -1 and $board_{row-1, col+1}$ is equal to 0,
                a. $new\_board_{row, col} \leftarrow 0$
                b. $new\_board_{row-1, col+1} \leftarrow -1$
                c. Append new_board to the end of possible_boards
            3. new_board ← a copy of board
            4. If $board_{row, col}$ is equal to -1 and $board_{row-1, col-1}$ is equal to 0,
                a. $new\_board_{row, col} \leftarrow 0$
                b. $new\_board_{row-1, col-1} \leftarrow -1$
                c. Append new_board to the end of possible_boards
4. If player is 1, then
    a. For each row, a number between 0 and 7 inclusive,
        i. For each col, a number between 0 and 7 inclusive,
            1. new_board ← a copy of board
            2. If $board_{row, col}$ is equal to 1 and $board_{row+1, col+1}$ is equal to 0,
                a. $new\_board_{row, col} \leftarrow 0$
                b. $new\_board_{row+1, col+1} \leftarrow 1$
                c. Append new_board to the end of possible_boards
            3. new_board ← a copy of board
            4. If $board_{row, col}$ is equal to 1 and $board_{row+1, col-1}$ is equal to 0,
                a. $new\_board_{row, col} \leftarrow 0$
                b. $new\_board_{row+1, col-1} \leftarrow 1$
                c. Append new_board to the end of possible_boards
5. Return possible_boards
```

Assuming that given 0 moves ahead and the current player is blocked, it counts as a win for the opposing player.

Recipe Name: simple_checkers

Inputs:

board, the current board configuration

player, either -1 or 1, with -1 representing the red player's turn and 1 representing the black player's turn

moves_ahead, the number of moves to check for a winning board state

Outputs:

winner, -1 if Red wins, 1 if Black wins, 0 if nobody can win

Steps:

```
1. winner ← 0
2. If moves_ahead is equal to 0 and the length of next_board(board, player) is not 0, then
    a. winner ← 0
3. Otherwise, if player is equal to -1 and the length of next_board(board, player) is 0, then
    a. winner ← 1
4. Otherwise, if player is equal to 1 and the length of next_board(board, player) is 0, then
    a. winner ← -1
5. Otherwise, if moves_ahead > 0 and player is equal to 1, then
    a. For each new_board in next_board(board, 1), do
        i. winner ← moves_ahead(new_board, -1,  moves_ahead – 1)
        ii. If winner is not equal to 0, then
            1. Return winner
6. Otherwise, if moves_ahead > 0 and player is equal to -1, then
    a. For each new_board in next_board(board, -1), do
        i. winner ← moves_ahead(new_board, 1,  moves_ahead – 1)
        ii. If winner is not equal to 0, then
            1. Return winner
7. Return winner
```

Base case 1:

When *moves_ahead* is 0 and the current player can move, return 0.

Base case 2:

When the black player cannot move and it is black’s turn, return -1

Base case 3:

When the red player cannot move and it is red’s turn, return 1

Recursive case 1:

When *moves_ahead* is greater than 0, it is black’s move, and black can move, run simple_checkers on all possible moves and return the first one that is non-zero.

Recursive case 2:

When *moves_ahead* is greater than 0, it is red’s move, and red can move, run simple_checkers on all possible moves and return the first one that is non-zero.