**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
   1. For each *row*, a number between 0 and 7 inclusive,
      1. For each *col*, a number between 0 and 7 inclusive,
         1. *new\_board* 🡨 a copy of *board*
         2. If *boardrow, col* is equal to -1 and *boardrow-1, col+1* is equal to 0,
            1. *new\_boardrow, col*🡨 0
            2. *new\_boardrow-1, col+1* 🡨 -1
            3. Append *new\_board* to the end of *possible\_boards*
         3. *new\_board* 🡨 a copy of *board*
         4. If *boardrow, col* is equal to -1 and *boardrow-1, col-1* is equal to 0,
            1. *new\_boardrow, col*🡨 0
            2. *new\_boardrow-1, col-1* 🡨 -1
            3. Append *new\_board* to the end of *possible\_boards*
4. If *player* is 1, then
   1. For each *row*, a number between 0 and 7 inclusive,
      1. For each *col*, a number between 0 and 7 inclusive,
         1. *new\_board* 🡨 a copy of *board*
         2. If *boardrow, col* is equal to 1 and *boardrow+1, col+1* is equal to 0,
            1. *new\_boardrow, col*🡨 0
            2. *new\_boardrow+1, col+1* 🡨 1
            3. Append *new\_board* to the end of *possible\_boards*
         3. *new\_board* 🡨 a copy of *board*
         4. If *boardrow, col* is equal to 1 and *boardrow+1, col-1* is equal to 0,
            1. *new\_boardrow, col*🡨 0
            2. *new\_boardrow+1, col-1* 🡨 1
            3. 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
   1. *winner* 🡨 0
3. Otherwise, if *player* is equal to -1 and the length of *next\_board*(*board*, *player)* is 0, then
   1. *winner* 🡨 1
4. Otherwise, if *player* is equal to 1 and the length of *next\_board*(*board*, *player)* is 0, then
   1. *winner* 🡨 -1
5. Otherwise, if *moves\_ahead* > 0 and *player* is equal to 1, then
   1. For each *new\_board* in *next\_board*(*board*, 1), do
      1. *winner* 🡨 *moves\_ahead*(*new\_board*, -1, *moves\_ahead* – 1)
      2. If *winner* is not equal to 0, then
         1. Return *winner*
6. Otherwise, if *moves\_ahead* > 0 and *player* is equal to -1, then
   1. For each *new\_board* in *next\_board*(*board*, -1), do
      1. *winner* 🡨 *moves\_ahead*(*new\_board*, 1, *moves\_ahead* – 1)
      2. 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.