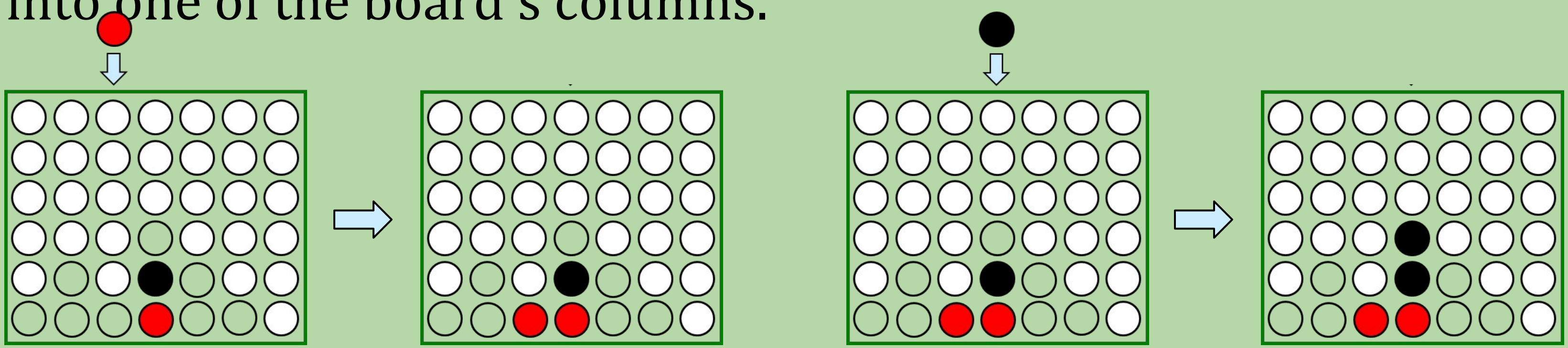
Lecture 13
Intro to Connect Four AI

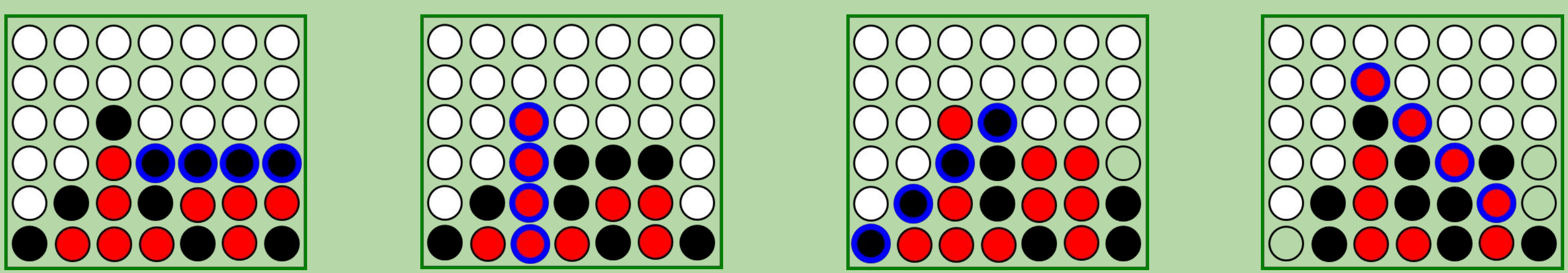
hw07: Connect Four!

- Two players, each with one type of checker
- 6 x 7 board that stands vertically
- Players take turns dropping a checker into one of the board's columns.



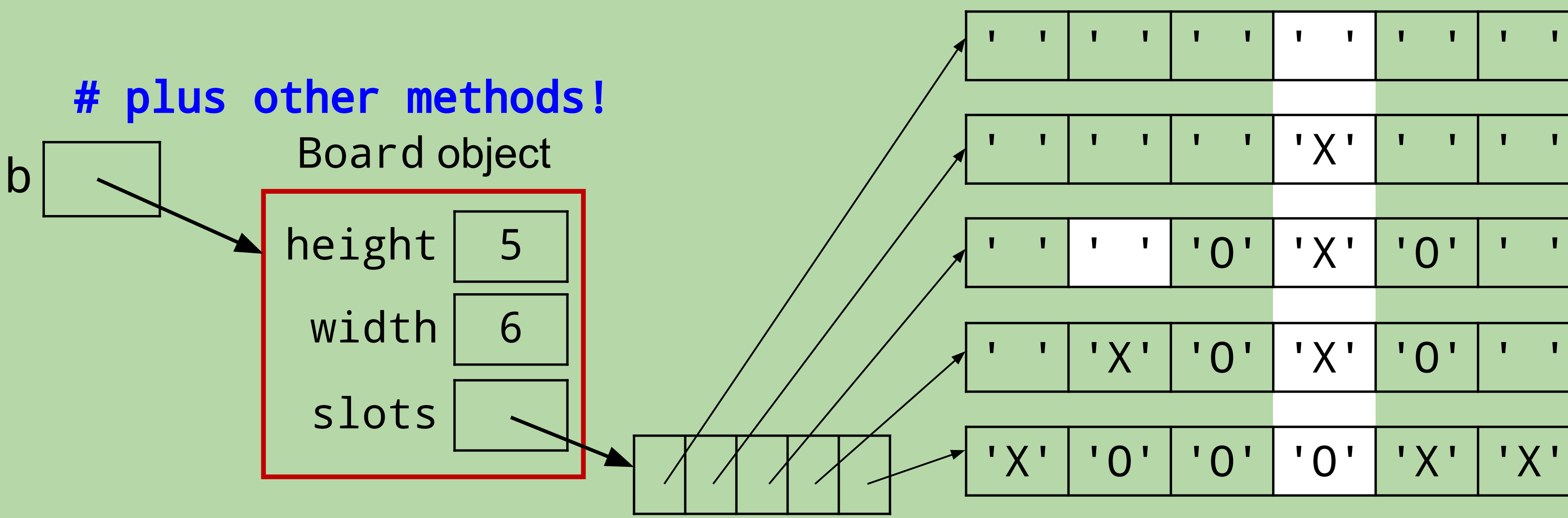
- Win == four adjacent checkers in any direction:

horizontal vertical up diagonal down diagonal



Board Class for Connect Four

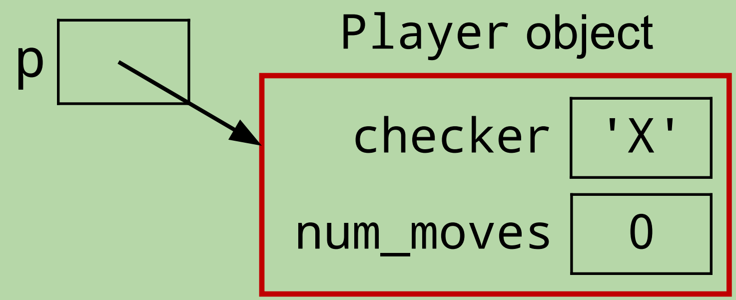
```
class Board:
    def __init__(self, height, width):
        ...

    def __repr__(self):
        ...

    def add_checker(self, checker, col):
        ...

    # plus other methods!
```



Player Class

```
class Player:

    def __init__(self, checker):
        ...

    def __repr__(self):
        ...

    def opponent_checker(self):
        ...

    def next_move(self, board):
        self.num_moves += 1

        while True:
            col = int(input('Enter a column: '))
            # if valid column index, return that integer
            # else, print 'Try again!' and keep looping
```

p = Player('X')



The APIs of Our `Board` and `Player` Classes

```
class Board:
    __init__(self,col)
    __repr__(self)
    add_checker(self,checker,col)
    clear(self)
    add_checkers(self,colnums)
    can_add_to(self,col)
    is_full(self)
    remove_checker(self,col)
    is_win_for(self,checker)
```

(provided)

Make sure to take full advantage of these methods in your work on hw06!

```
class Player:
    __init__(self,col)
    __repr__(self)
    opponent_checker(self)
    next_move(self,board)
```

(for you to implement)

```
def process_move(player, board):
    '''Applies a player object's next move to a board object.
       Returns true if the player wins or a tie occurs,
       False otherwise'''
       pass

def connect_four(player1, player2) # provided in stencil
    '''Plays a connect four game between player1 and player2,
       Returns the final board configuration.'''

     while True:  % Play until a win or tie occurs.
        if process_move(player1, board):
            return board

        if process_move(player2, board):
            return board
```

What are the appropriate method calls?

```
class Board:
    __init__(self,col)
    __repr__(self)
    add_checker(self,checker,col)
    clear(self)
    add_checkers(self,colnums)
    can_add_to(self,col)
    is_full(self)
    remove_checker(self,col)
    is_win_for(self,checker)

class Player:
    __init__(self,col)
    __repr__(self)
    opponent_checker(self)
    next_move(self,board)
```

```
# client code
def process_move(player,board):
    ...

    # get move from player
    col = _____________

    # apply the move
    __________________
    ...
```

What are the appropriate method calls?

```
class Board:
   __init__(self,col)
   __repr__(self)
   add_checker(self,checker,col)
   clear(self)
   add_checkers(self,colnums)
   can_add_to(self,col)
   is_full(self)
   remove_checker(self,col)
   is_win_for(self,checker)

class Player:
   __init__(self,col)
   __repr__(self)
   opponent_checker(self)
   next_move(self,board)
```

```
# client code
def process_move(player,board):
   ...

   # get move from player
   col = player.next_move(board)

   # apply the move
   board.add_checker(..., col)
   ...
```

Inheritance in Connect Four

- `Player` – the superclass
  - includes fields and methods needed by all Connect 4 players
  - in particular, a `next_move` method
  - use this class for human players

Inheritance in Connect Four

- `Player` – the superclass
  - includes fields and methods needed by all C4 players
  - in particular, a `next_move` method
  - use this class for human players

- `RandomPlayer` – a subclass for an *un*intelligent computer player
  - no new fields
  - overrides `next_move` with a version that chooses at random from the non-full columns

Inheritance in Connect Four

- `Player` – the superclass
  - includes fields and methods needed by all C4 players
  - in particular, a `next_move` method
  - use this class for human players

- `RandomPlayer` – a subclass for an *un*intelligent computer player
  - no new fields
  - overrides `next_move` with a version that chooses at random from the non-full columns

- `AIPlayer` – a subclass for an "intelligent" computer player
  - uses AI techniques
  - new fields for details of its strategy
  - overrides `next_move` with a version that tries to determine the best move!

Using the Player Classes

- Example 1: two human players

```
>>> connect_four(Player('X'), Player('O'))
```

- Example 2: human player vs. AI computer player:

```
>>> connect_four(Player('X'), AIPlayer('O', 'LEFT', 3))
```

- `connect_four()` repeatedly calls `process_move()`:

```
def connect_four(player1, player2):
    print('Welcome to Connect Four!')
    print()
    board = Board(6, 7)
    print(board)

    while True:
        if process_move(player1, board):
            return board
        if process_move(player2, board):
            return board
```

OOP == Object-Oriented Power!

```
def process_move(player, board):
    ...
    col = player.next_move(board)

    ...
```

- Which version of `next_move` gets called?

OOP == Object-Oriented Power!

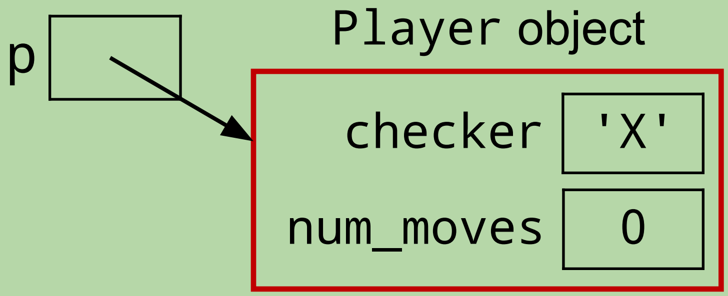
```
def process_move(player, board):
    ...
    col = player.next_move(board)

    ...
```

- Which version of `next_move` gets called?
- It depends!
  - if `player` is a `Player` object, call `next_move` from that class
  - if `player` is a `RandomPlayer`, call that version of `next_move`
  - if `player` is an `AIPlayer`, call that version of `next_move`
- The appropriate version is automatically called, depending on which object `player` was defined as!

RandomPlayer, AIPlayer Class

```
class Player:

    def __init__(self, checker):
        ...

    def __repr__(self):
        ...

    def opponent_checker(self):
        ...

    def next_move(self, board):
        self.num_moves += 1


        ???
```

<u>p = Player('X')</u>



Why AI Is Challenging

Make no mistake about it:
computers process numbers – not symbols.

Computers can only help us to the extent that
we can ***arithmetize*** an activity.

- paraphrasing Alan Perlis

"Arithmetizing" Connect Four

- Our `AIPlayer` assigns a score to each possible move
  - i.e., to each column

- It *looks ahead* some number of moves into the future to determine the score.
  - *lookahead* = # of future moves that the player considers

"Arithmetizing" Connect Four

- Our `AIPlayer` assigns a score to each possible move
  - i.e., to each column

- It *looks ahead* some number of moves into the future
  to determine the score.
  - *lookahead* = # of future moves that the player considers

- Scoring columns:

  –1: an already *full column*

"Arithmetizing" Connect Four

- Our `AIPlayer` assigns a score to each possible move
  - i.e., to each column

- It *looks ahead* some number of moves into the future to determine the score.
  - *lookahead* = # of future moves that the player considers

- Scoring columns:

  –1: an already *full column*

  0: if we choose this column, it will result in a *loss* at some point during the player's lookahead

"Arithmetizing" Connect Four

- Our `AIPlayer` assigns a score to each possible move
  - i.e., to each column

- It *looks ahead* some number of moves into the future
  to determine the score.
  - *lookahead* = # of future moves that the player considers

- Scoring columns:

  –1: an already *full column*

  0: if we choose this column, it will result in a *loss* at some point during the player's lookahead

  100: if we choose this column, it will result in a *win* at some point during the player's lookahead

"Arithmetizing" Connect Four

- Our `AIPlayer` assigns a score to each possible move
  - i.e., to each column

- It *looks ahead* some number of moves into the future to determine the score.
  - *lookahead* = # of future moves that the player considers

- Scoring columns:

–1: an already *full column*

0: if we choose this column, it will result in a *loss* at some point during the player's lookahead

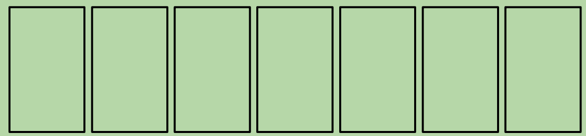
100: if we choose this column, it will result in a *win* at some point during the player's lookahead

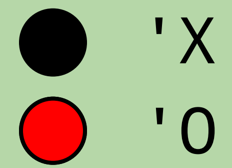
50: if we choose this column, it will result in *neither a win nor a loss* during the player's lookahead

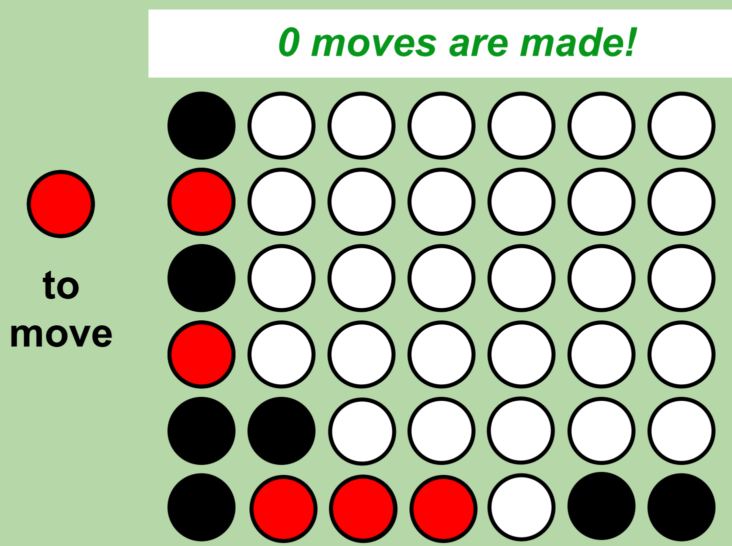
A Lookahead of 0

- A lookahead-0 player only assesses the current board (0 moves!).

LA-0 scores for







A Lookahead of 0

- A lookahead-0 player only assesses the current board (0 moves!).

LA-0 scores for

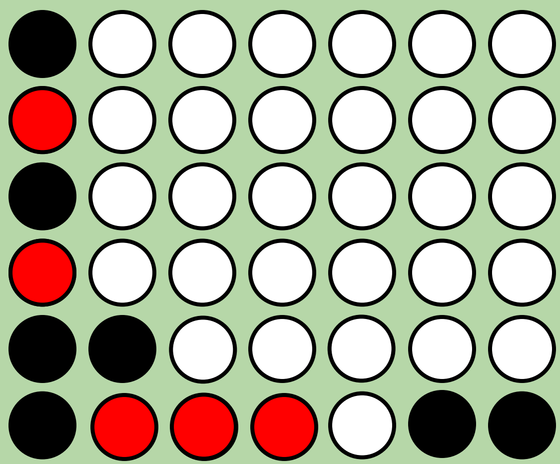
-1						

'X'

'O'

0 moves are made!

to move



A Lookahead of 0

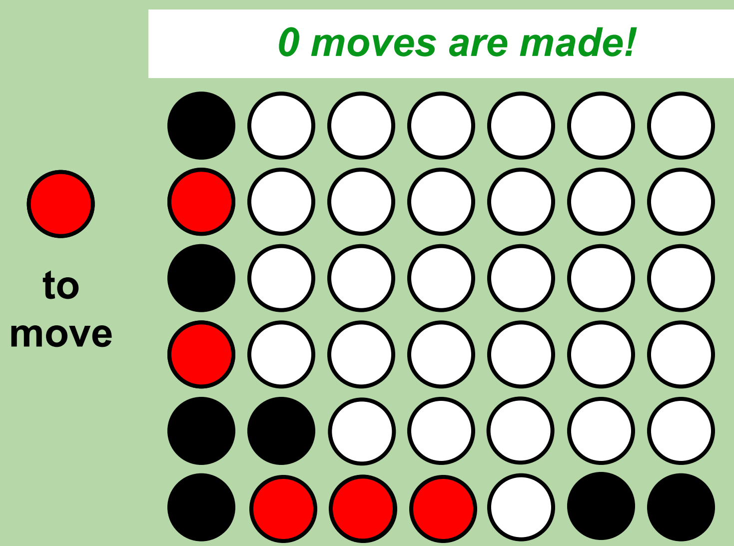
- A lookahead-0 player only assesses the current board (0 moves!).

LA-0 scores for

-1	50	50	50	50	50	50

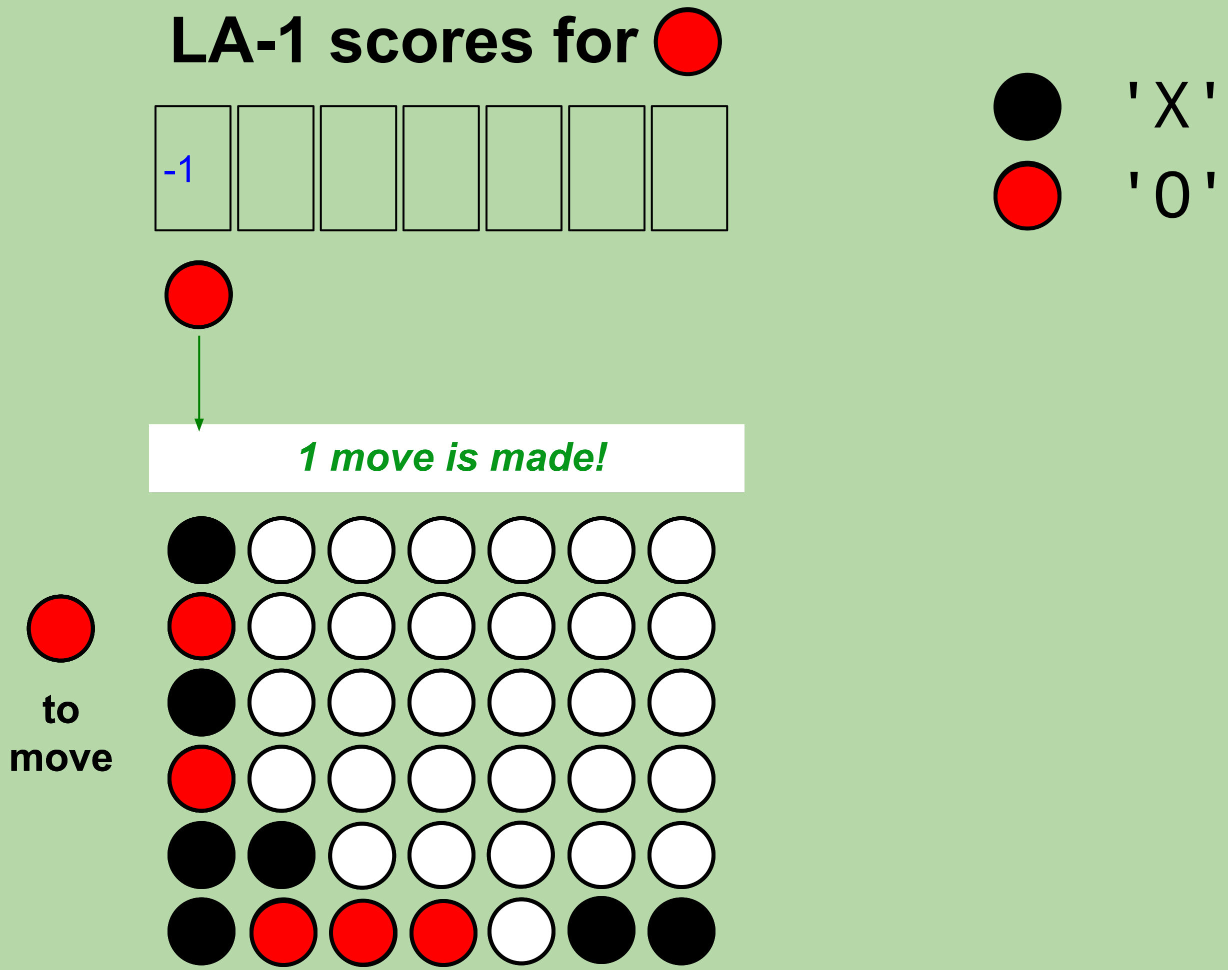
'X'

'O'



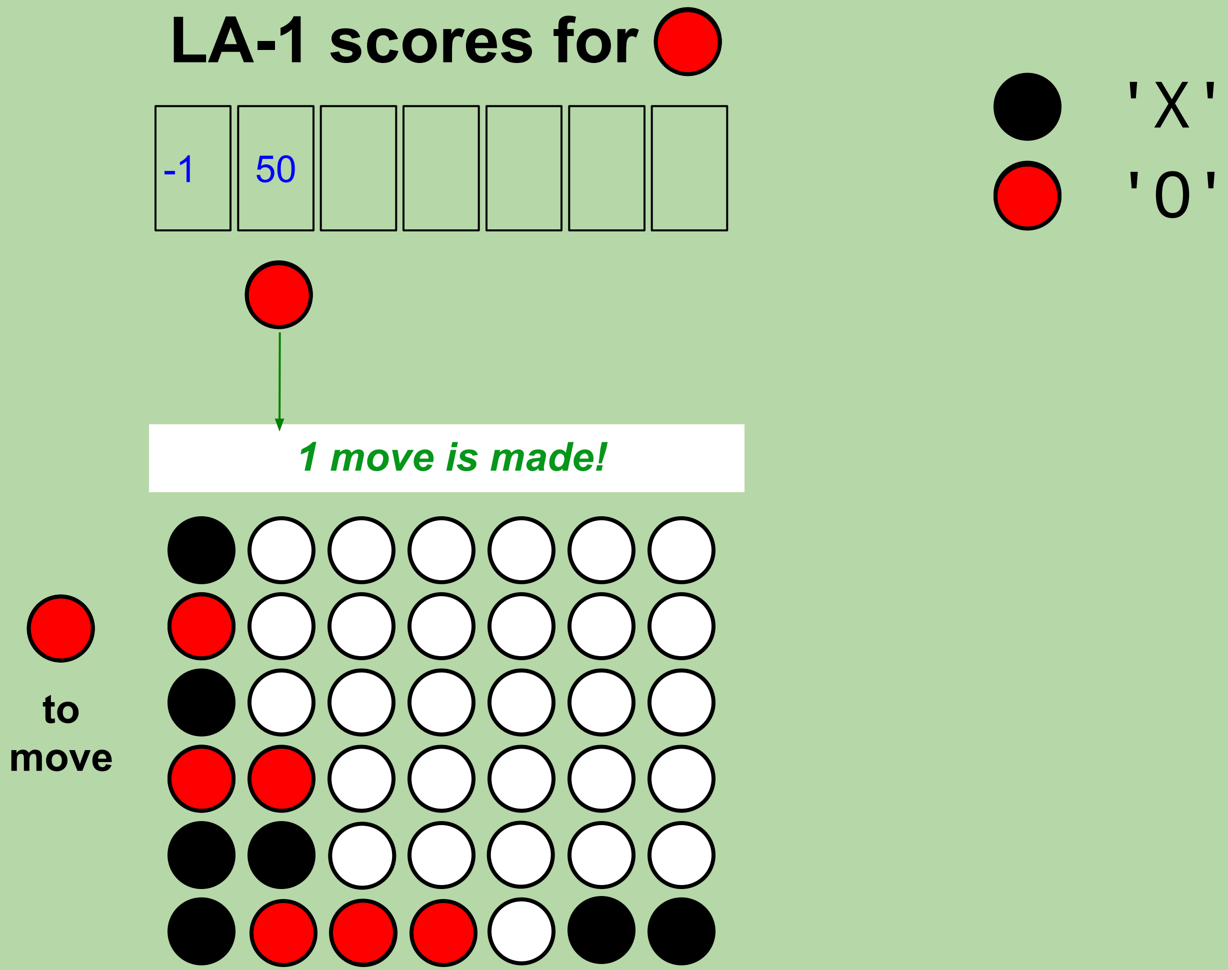
A Lookahead of 1

- A lookahead-1 player assesses the outcome of *only* the considered move.



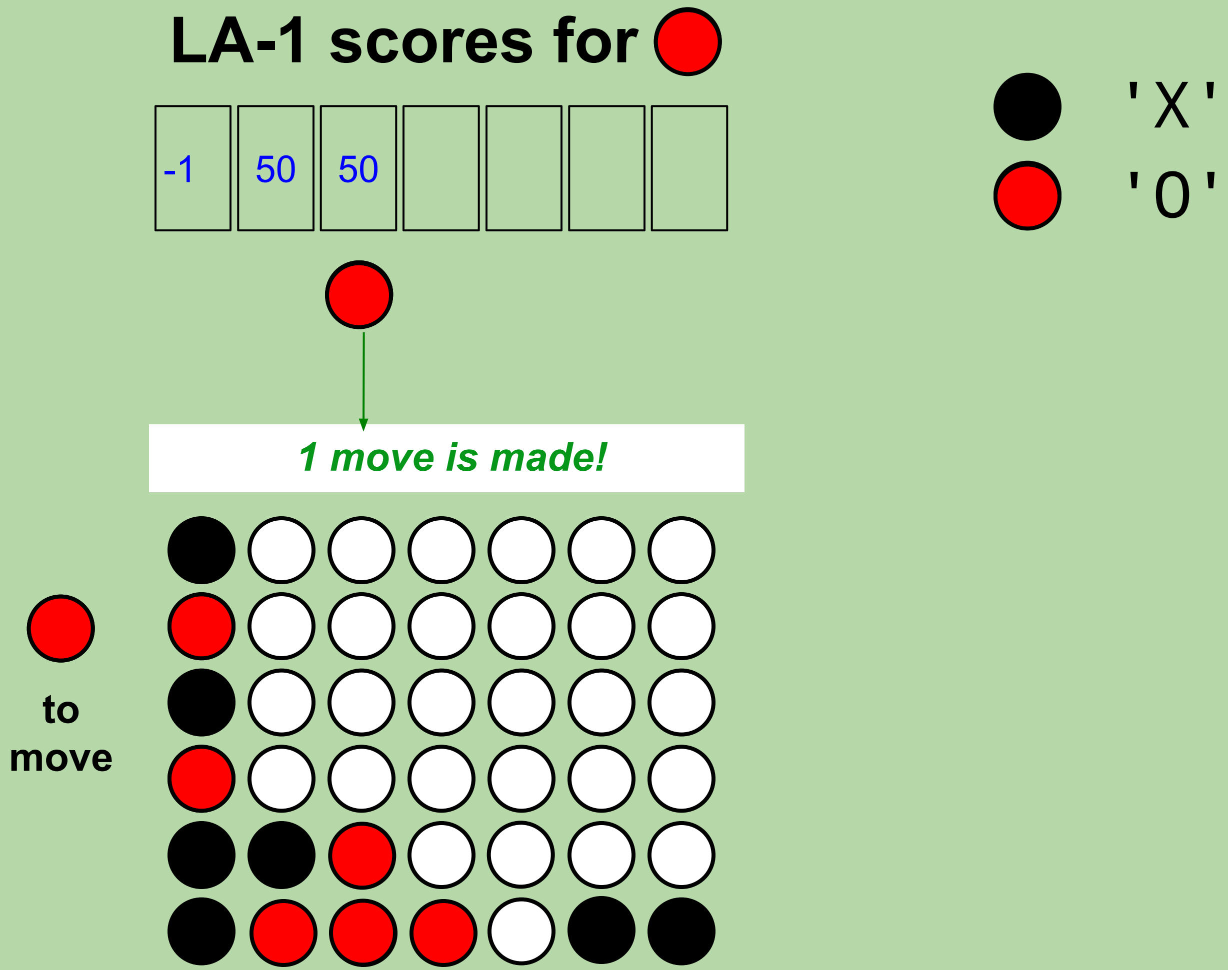
A Lookahead of 1

- A lookahead-1 player assesses the outcome of *only* the considered move.



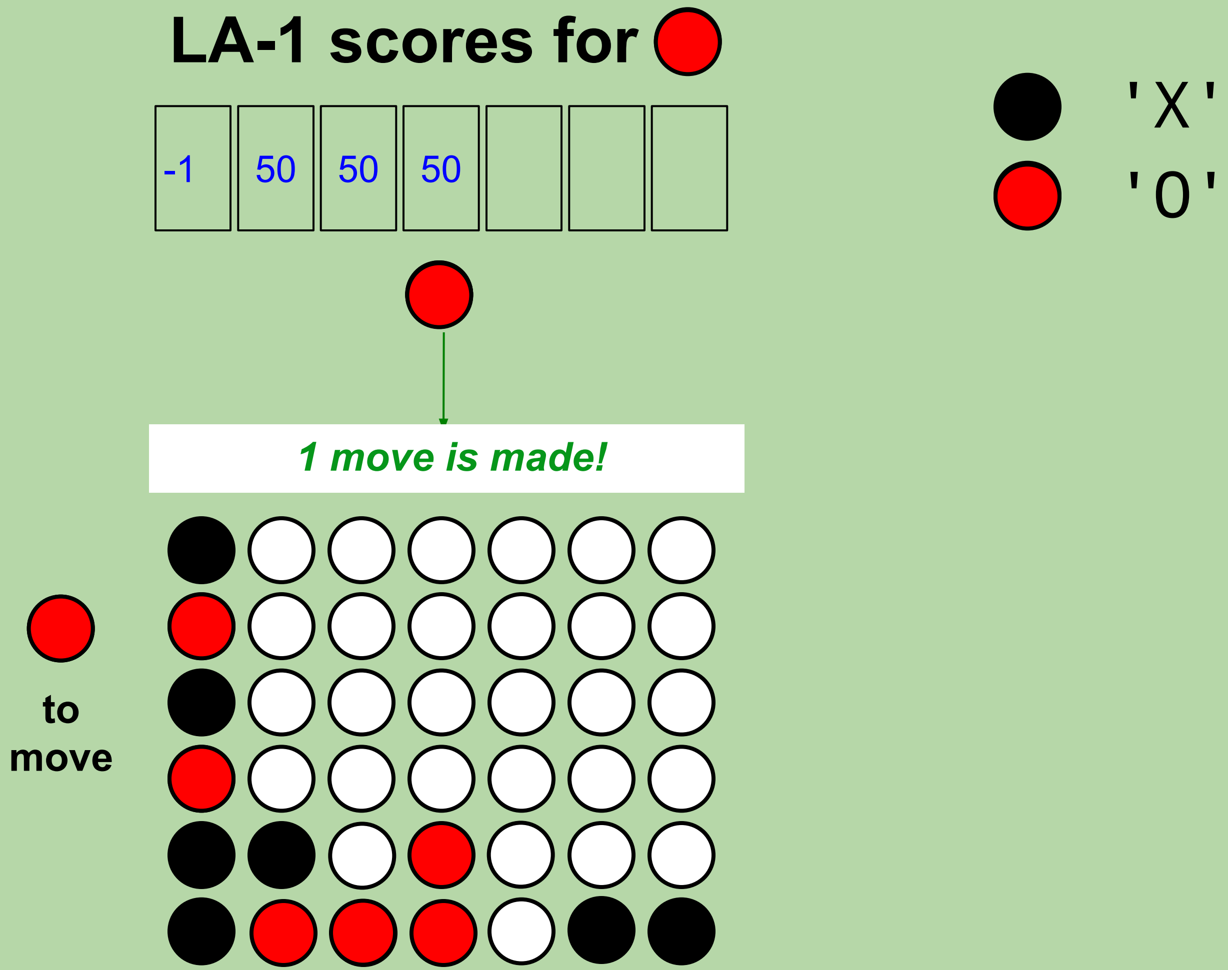
A Lookahead of 1

- A lookahead-1 player assesses the outcome of *only* the considered move.



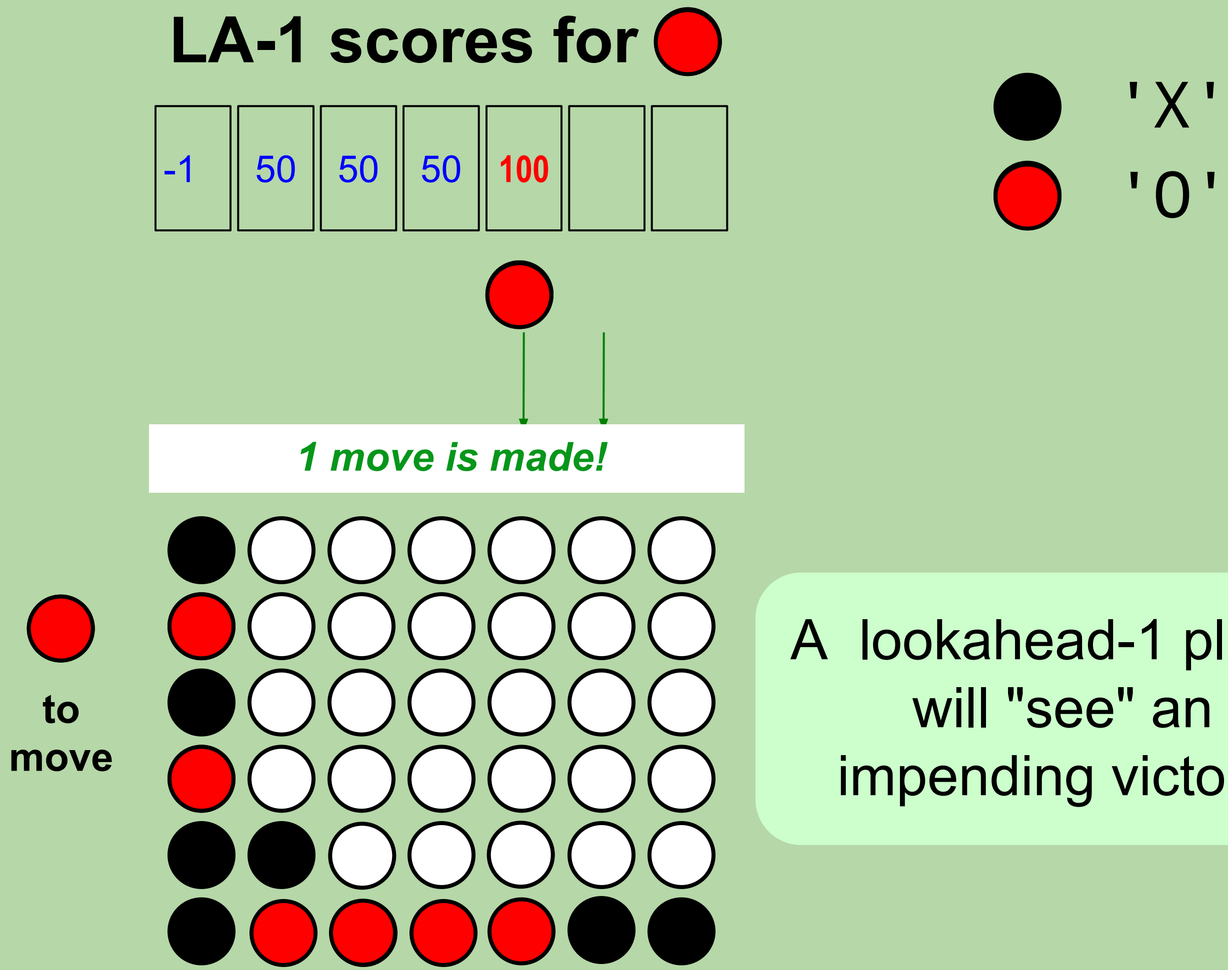
A Lookahead of 1

- A lookahead-1 player assesses the outcome of *only* the considered move.



A Lookahead of 1

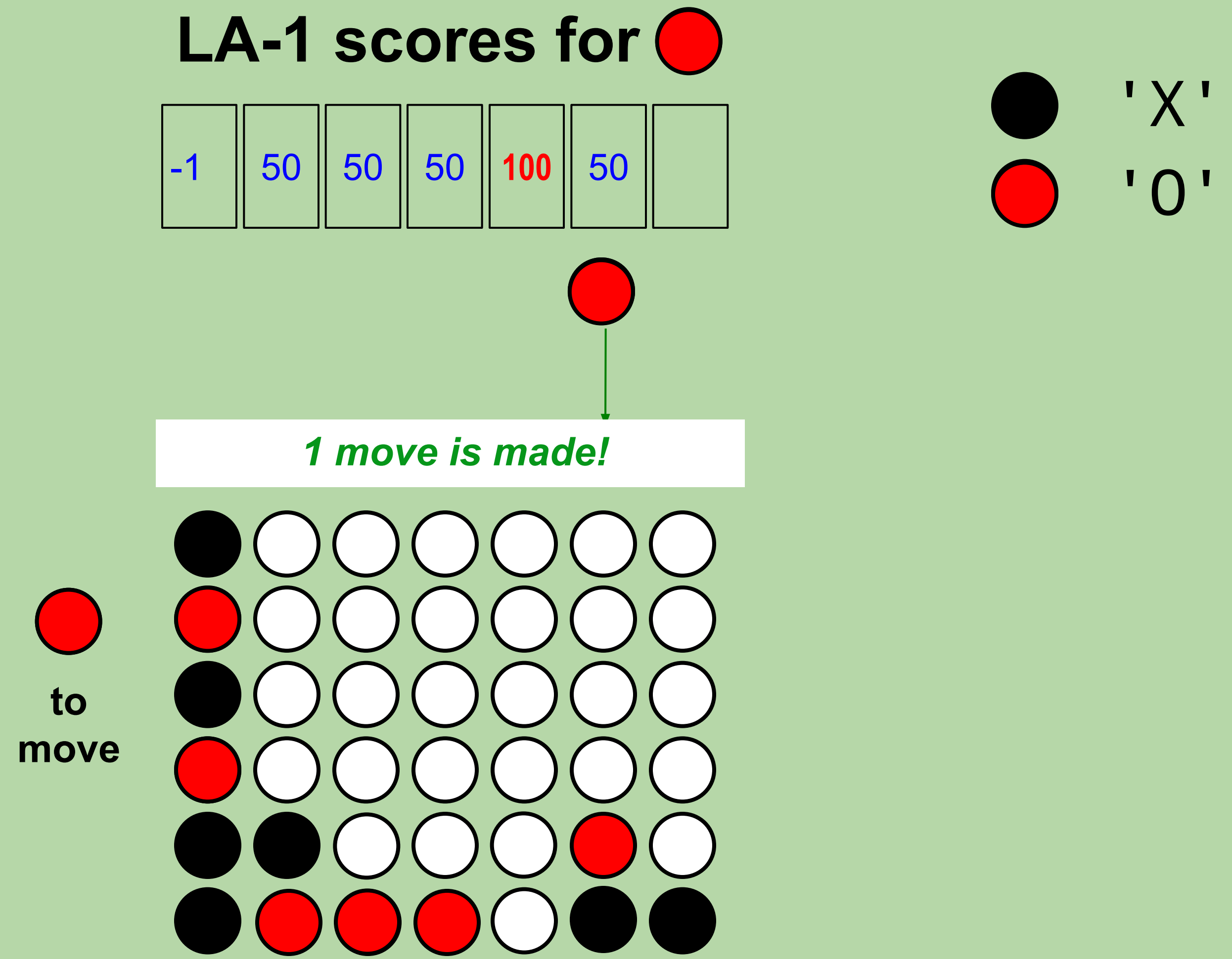
- A lookahead-1 player assesses the outcome of *only* the considered move.



A lookahead-1 player will "see" an impending victory.

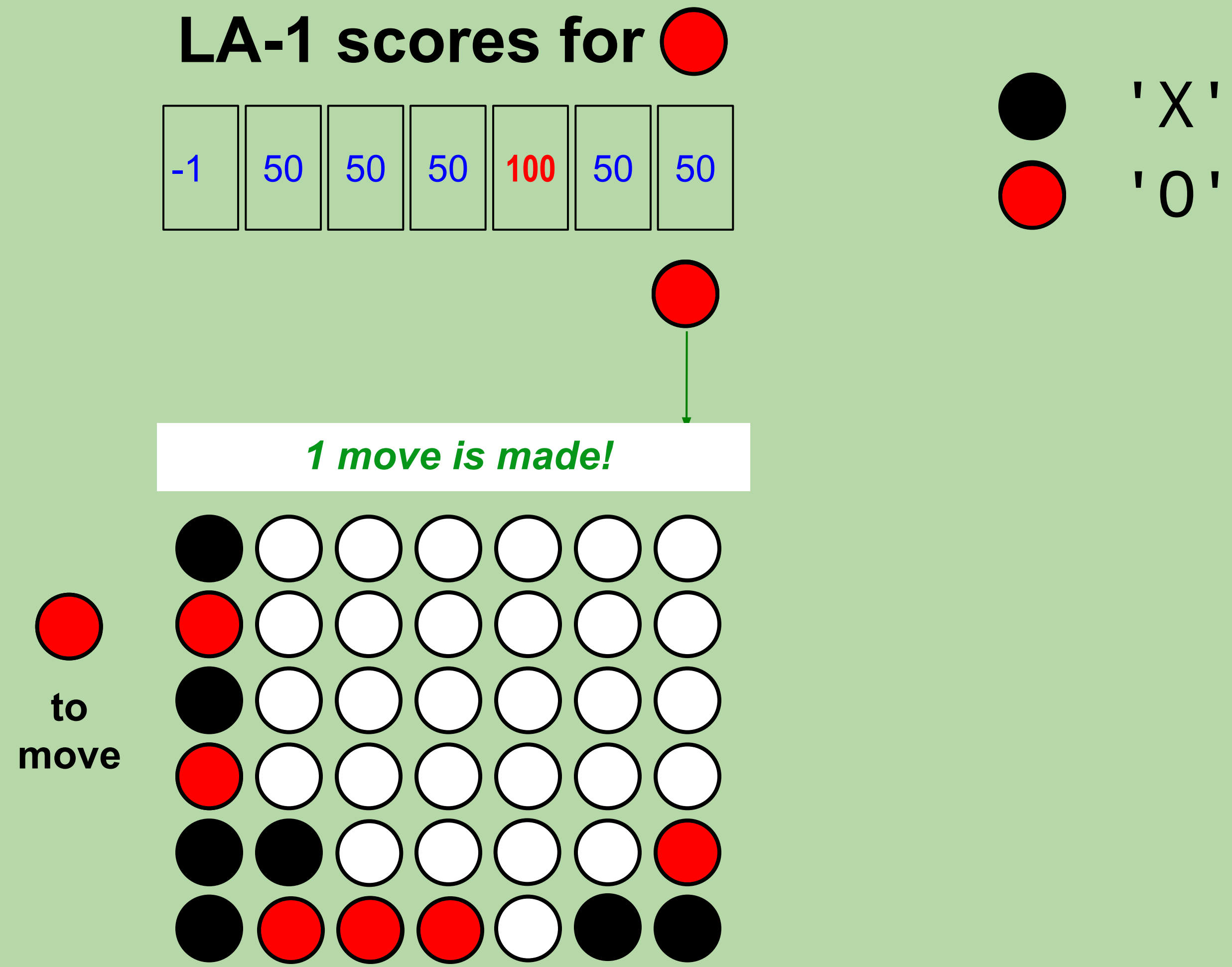
A Lookahead of 1

- A lookahead-1 player assesses the outcome of *only* the considered move.



A Lookahead of 1

- A lookahead-1 player assesses the outcome of *only* the considered move.



A Lookahead of 1

- A lookahead-1 player assesses the outcome of *only* the considered move.

LA-1 scores for

-1	50	50	50	**100**	50	50

A lookahead-1 player will "see" an impending victory.

`next_move` will return 4 for `AIPlayer`!

A Lookahead of 1

- A lookahead-1 player assesses the outcome of *only* the considered move

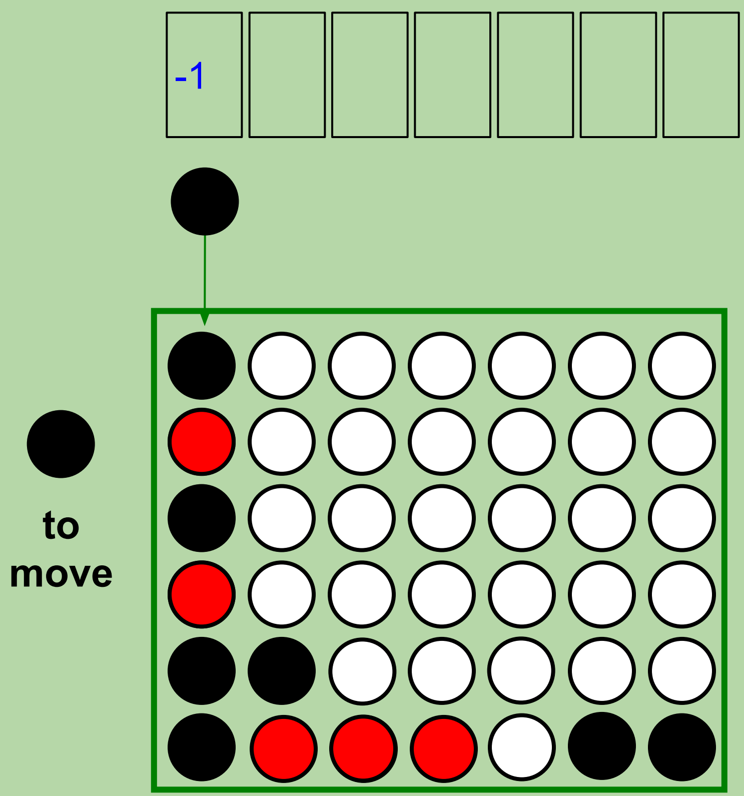
How do these scores change if it is ●`s turn instead of ●`s?

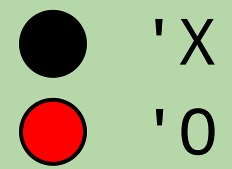
Let’s look at the lookahead-2 scores for the ● player.

A Lookahead of 2

- A lookahead-2 player looks 2 moves ahead.
  - what if I (●) make this move, and then my opponent (●) makes *its best move*?
  - ***note:*** we assume the opponent looks ahead 2 – 1 = 1 move

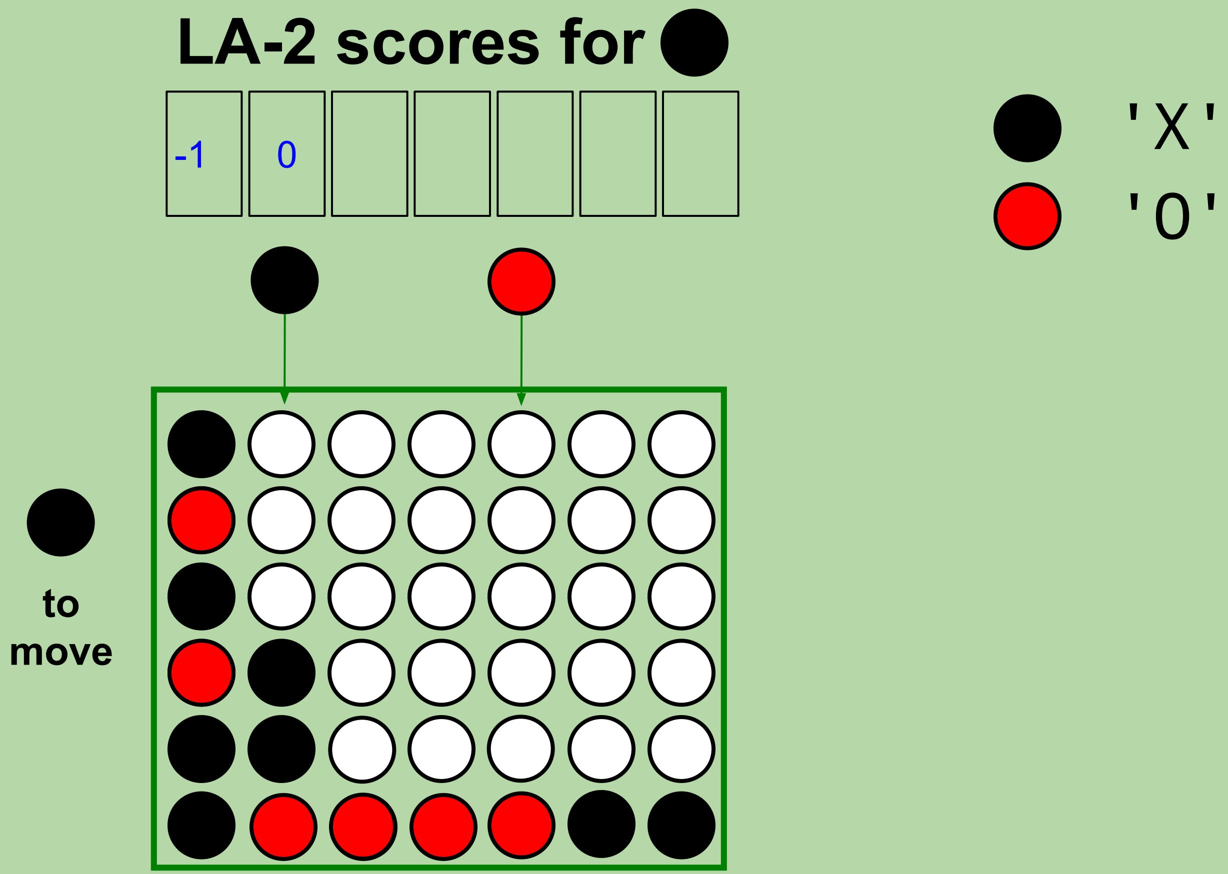
LA-2 scores for ●





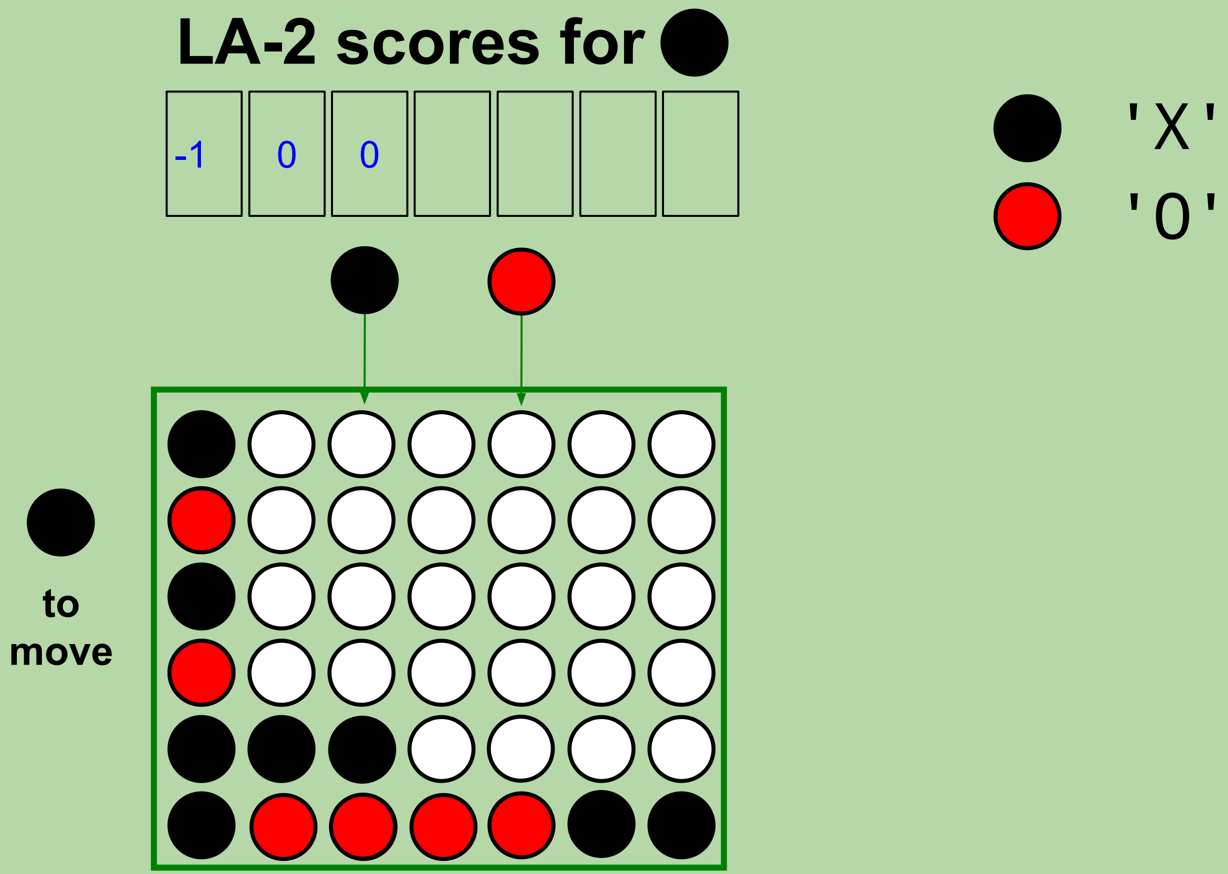
A Lookahead of 2

- A lookahead-2 player looks 2 moves ahead.
  - what if I (⚫) make this move, and then my opponent (🔴) makes *its best move*?
  - ***note:*** we assume the opponent looks ahead 2 – 1 = 1 move



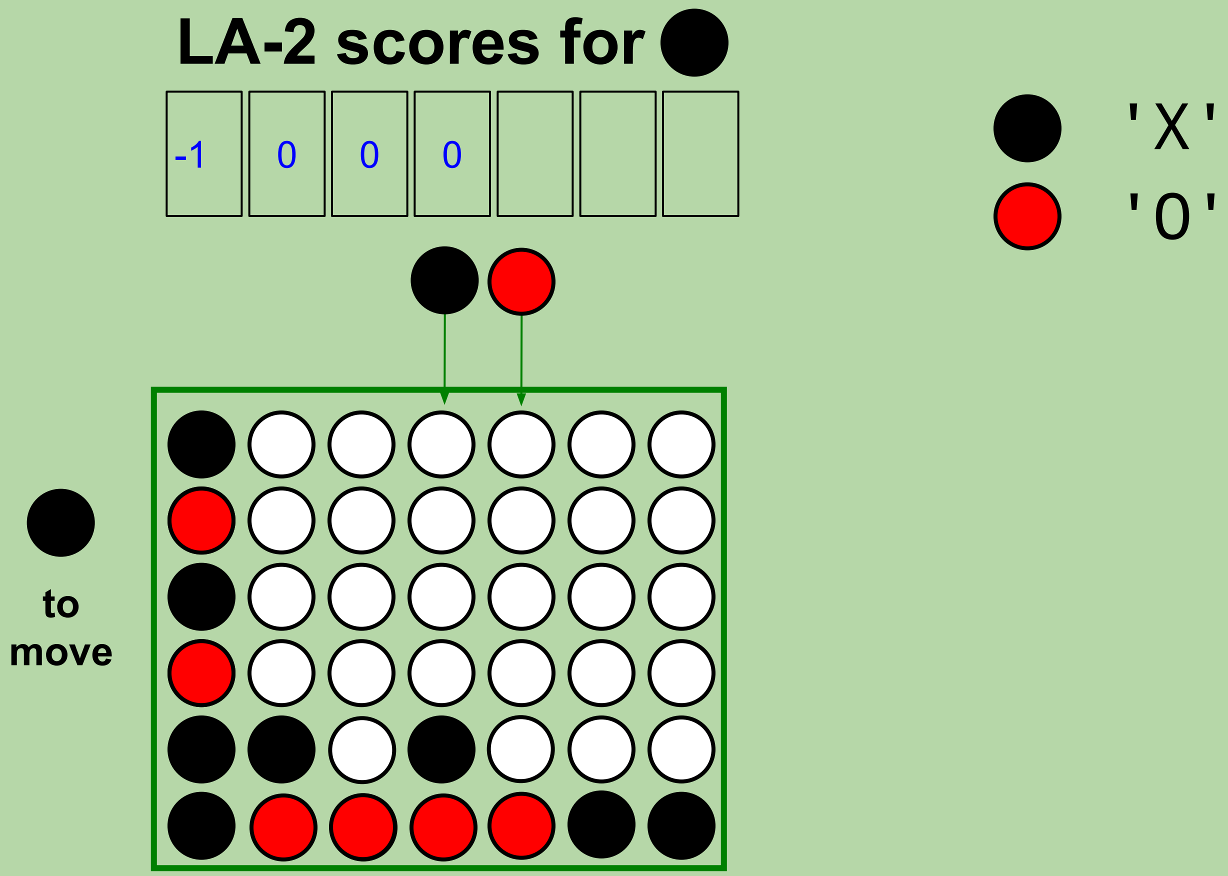
A Lookahead of 2

- A lookahead-2 player looks 2 moves ahead.
  - what if I make this move, and then my opponent makes *its best move*?
  - ***note:*** we assume the opponent looks ahead 2 – 1 = 1 move



A Lookahead of 2

- A lookahead-2 player looks 2 moves ahead.
  - what if I make this move, and then my opponent makes *its best move*?
  - ***note:*** we assume the opponent looks ahead 2 – 1 = 1 move



A Lookahead of 2

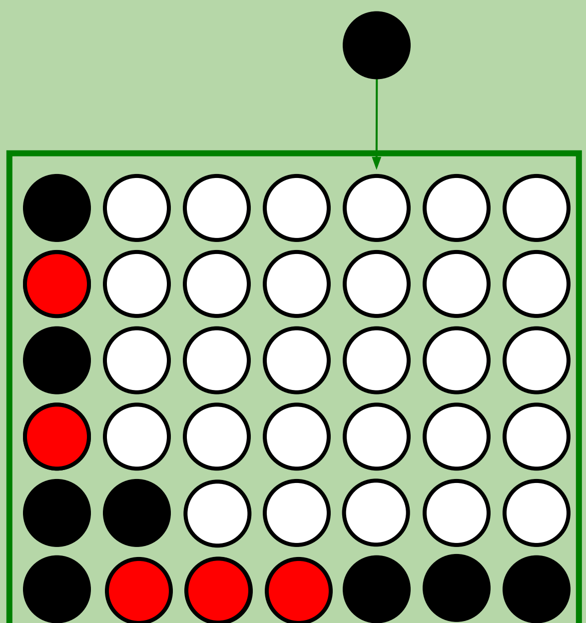
- A lookahead-2 player looks 2 moves ahead.
  - what if I make this move, and then my opponent makes *its best move*?
  - ***note:*** we assume the opponent looks ahead 2 – 1 = 1 move

LA-2 scores for ●

-1	0	0	0	50		

● 'X'
🔴 'O'

● **to move**

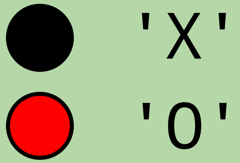


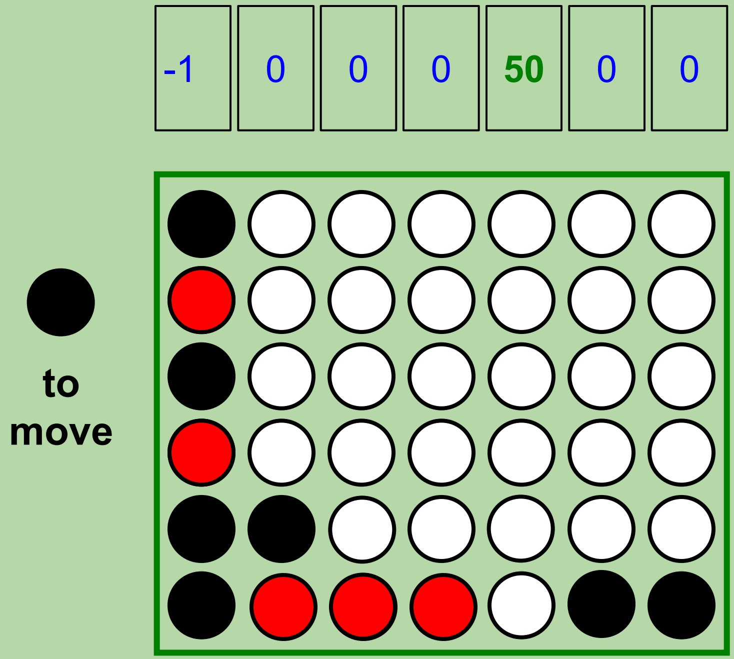
If ● moves into column index 4, then all of 🔴's directly subsequent moves will result in neither a win nor a loss for ●.

A Lookahead of 2

- A lookahead-2 player looks 2 moves ahead.
  - what if I make this move, and then my opponent makes *its best move*?
  - ***note:*** we assume the opponent looks ahead 2 – 1 = 1 move

LA-2 scores for ●





A lookahead-2 player will "see" a way to win *or* a way to block the opponent's win.

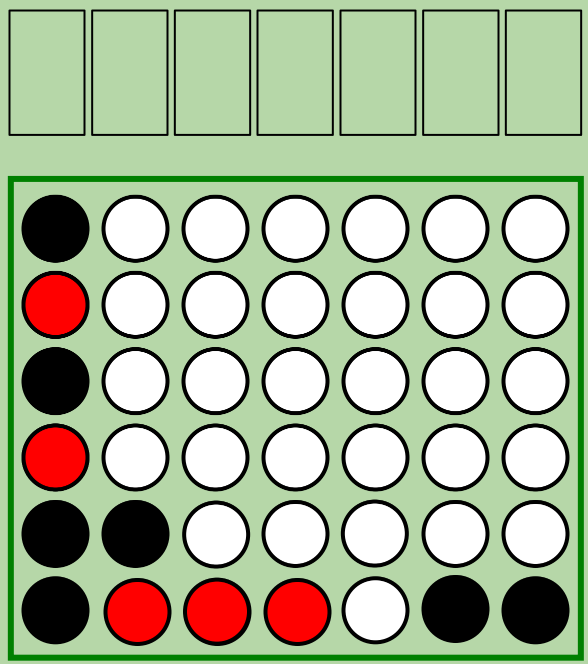
A Lookahead of 2

- A lookahead-2 player looks 2 moves ahead.
  - what if I make this move, and then my opponent makes *its best move*?
  - ***note:*** we assume the opponent looks ahead 2 – 1 = 1 move

LA-2 scores for ●

● 'X'
● 'O'

Try it!



to move

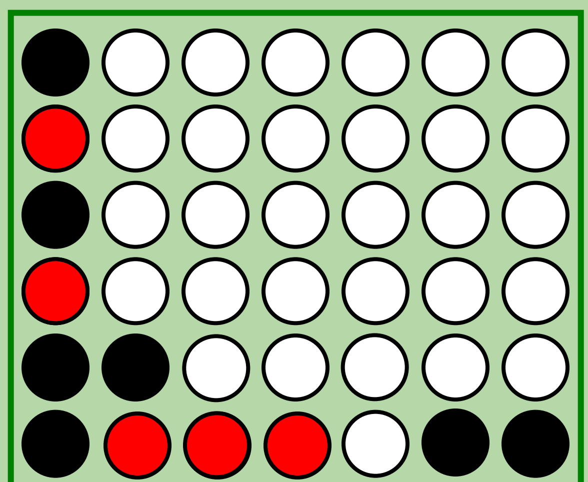
A Lookahead of 2

- A lookahead-2 player looks 2 moves ahead.
  - what if I make this move, and then my opponent makes *its best move*?
  - ***note:*** we assume the opponent looks ahead 2 – 1 = 1 move

LA-2 scores for

Try it!

-1	50	50	50	100	50	50



to move

'X'

'O'

AI for Connect Four (cont.)

based in part on notes from the CS-for-All curriculum developed at Harvey Mudd College

Recall: "Arithmetizing" Connect Four

- Our `AIPlayer` assigns a score to each possible move
  - i.e., to each column

- It *looks ahead* some number of moves into the future to determine the score.
  - *lookahead* = # of future moves that the player considers

- Scoring columns:

  –1: an already *full column*

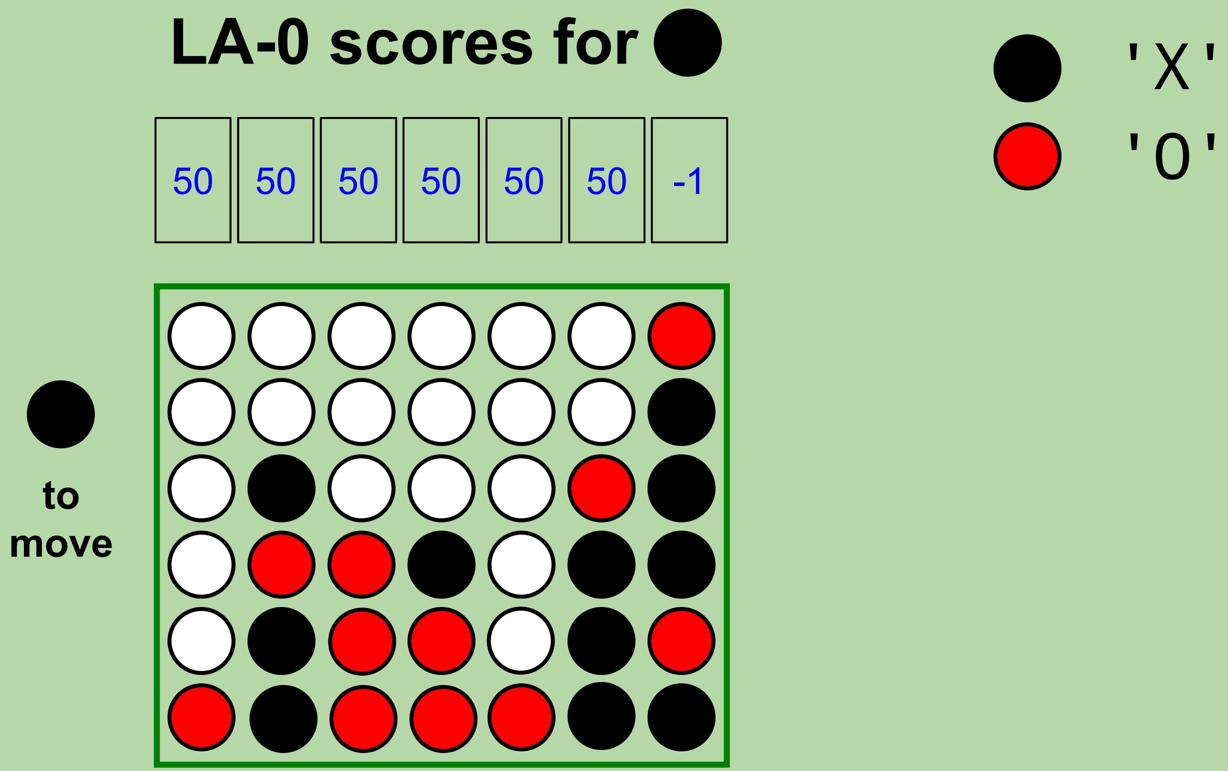
  0: if we choose this column, it will result in a *loss* at some point during the player's lookahead

  100: if we choose this column, it will result in a *win* at some point during the player's lookahead

  50: if we choose this column, it will result in *neither a win nor a loss* during the player's lookahead

Example 2: LA-0

- A lookahead-0 player only assesses the current board (0 moves!).



Example 2: LA-1

- A lookahead-1 player assesses the outcome of *only* the considered move.

LA-0 scores for

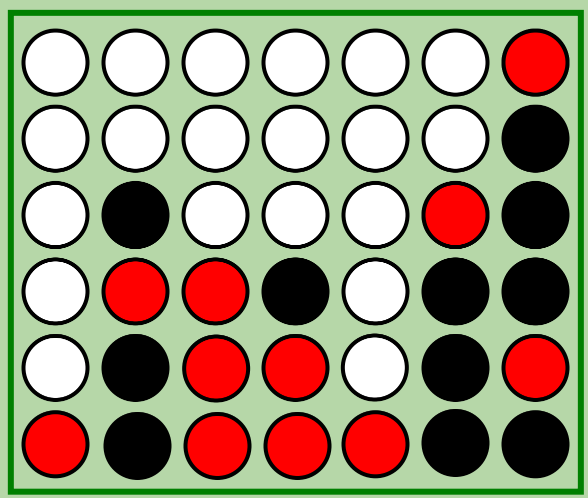
'X'

'O'

What scores change with the increased LA?

50	50	50	50	50	50	-1

to move



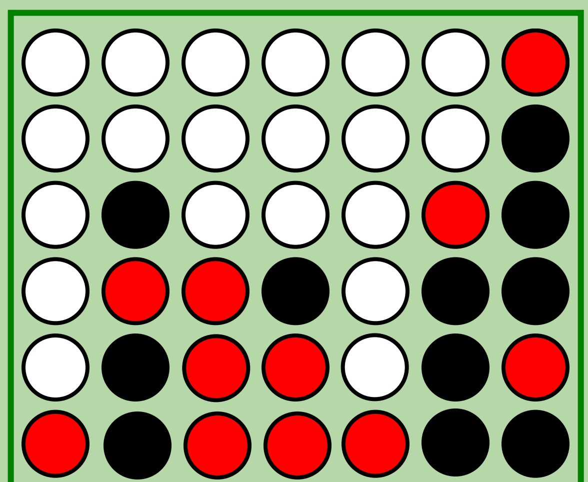
Example 2: LA-1

- A lookahead-1 player assesses the outcome of *only* the considered move.

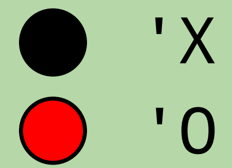
What scores change with the increased LA?
none of them!

LA-1 scores for ●

50	50	50	50	50	50	-1



● **to move**



Example 2: LA-2

- A lookahead-2 player looks 2 moves ahead.
  - what if I make this move, and then my opponent makes *its best move*?
  - ***note:*** we assume the opponent looks ahead 2 – 1 = 1 move

LA-1 scores for ●

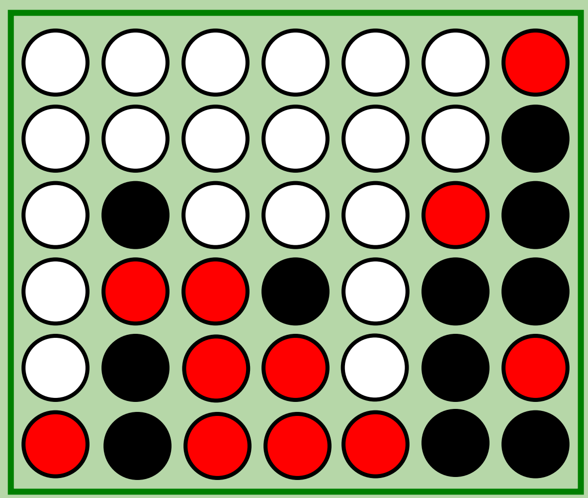
What would change?

50	50	50	50	50	50	-1

● 'X'

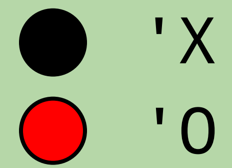
● 'O'

● **to move**



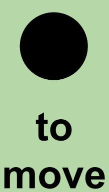
Example 2: LA-2

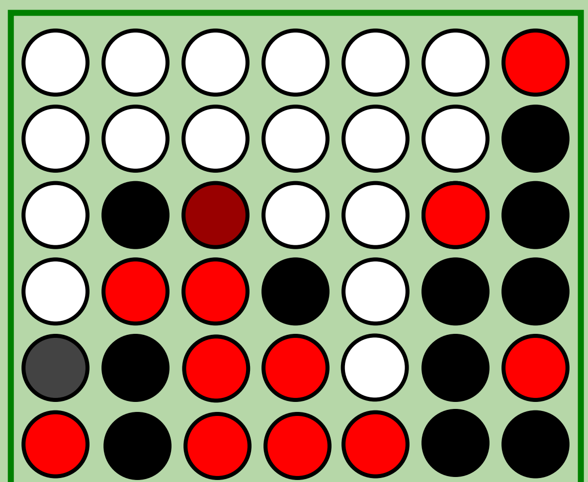
- A lookahead-2 player looks 2 moves ahead.
  - what if I make this move, and then my opponent makes *its best move*?
  - ***note:*** we assume the opponent looks ahead 2 – 1 = 1 move



LA-2 scores for ●

0	0	50	0	0	0	-1



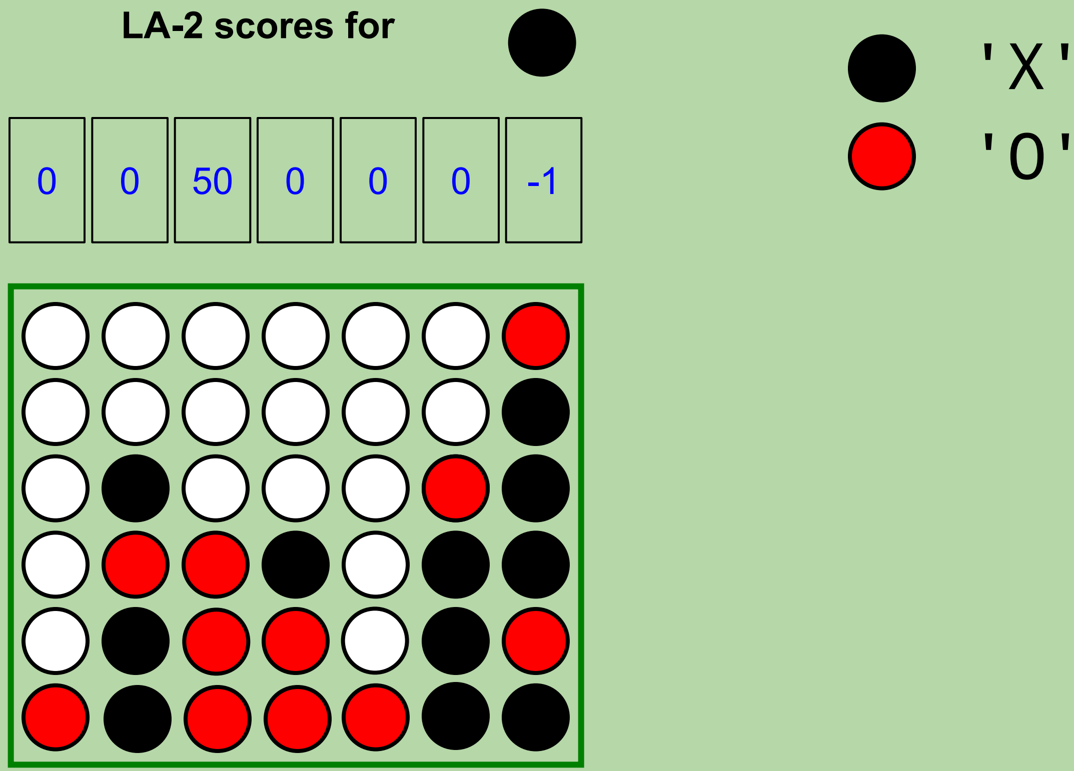


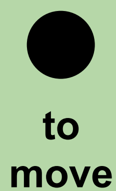
For example,
if black moves here →
red can win by going
to column 2 as shown.

LA-3!

- A lookahead-3 player looks 3 moves ahead.
  - what if I make this move, and then my opponent makes its best move, *and then I make my best subsequent move?*
  - ***note:*** we assume the opponent looks ahead 3 – 1 = 2 moves

What would change?

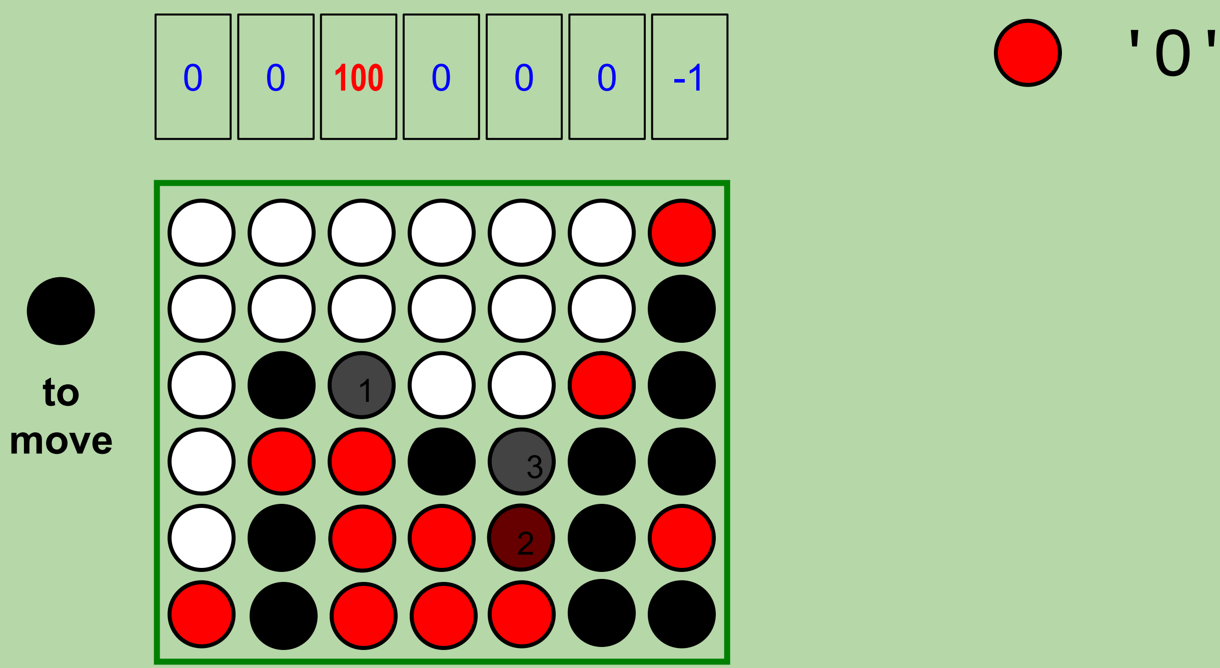




LA-3!

- A lookahead-3 player looks 3 moves ahead.
  - what if I make this move, and then my opponent makes its best move, *and then I make my best subsequent move?*
  - ***note:*** we assume the opponent looks ahead 3 – 1 = 2 moves

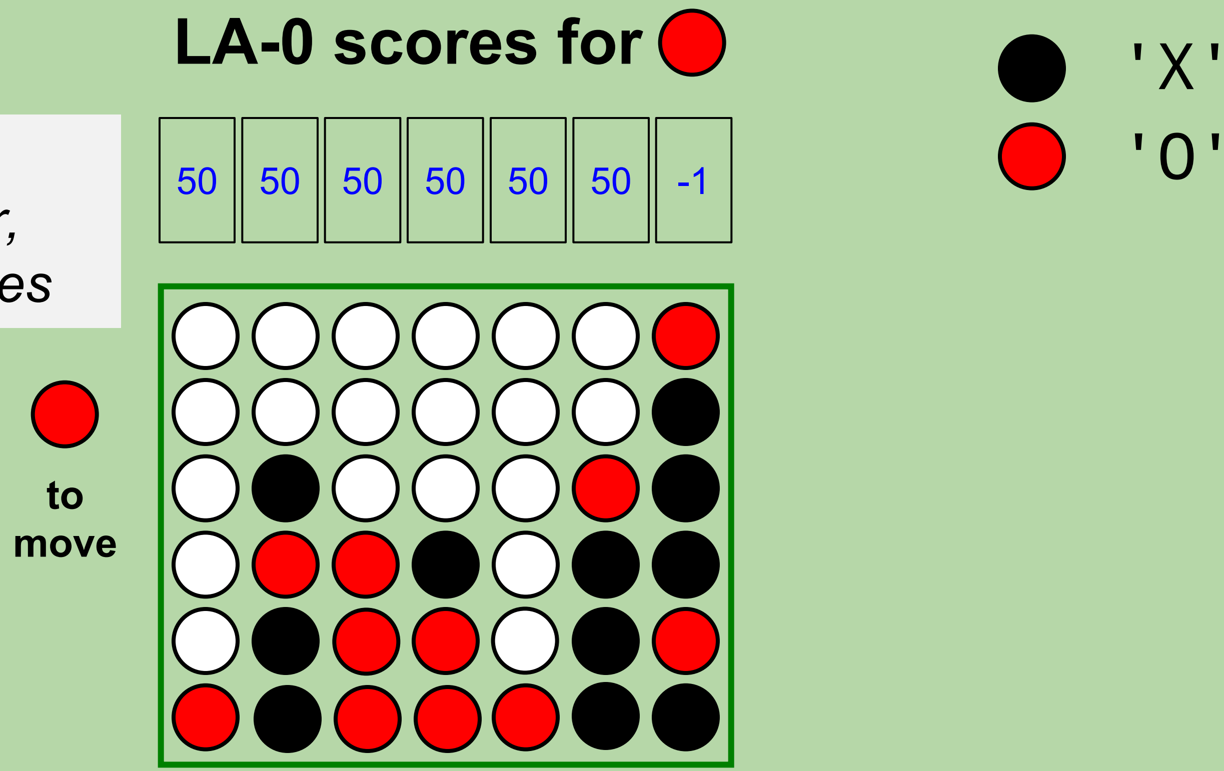
LA-3 scores for



Example 2: LA-0

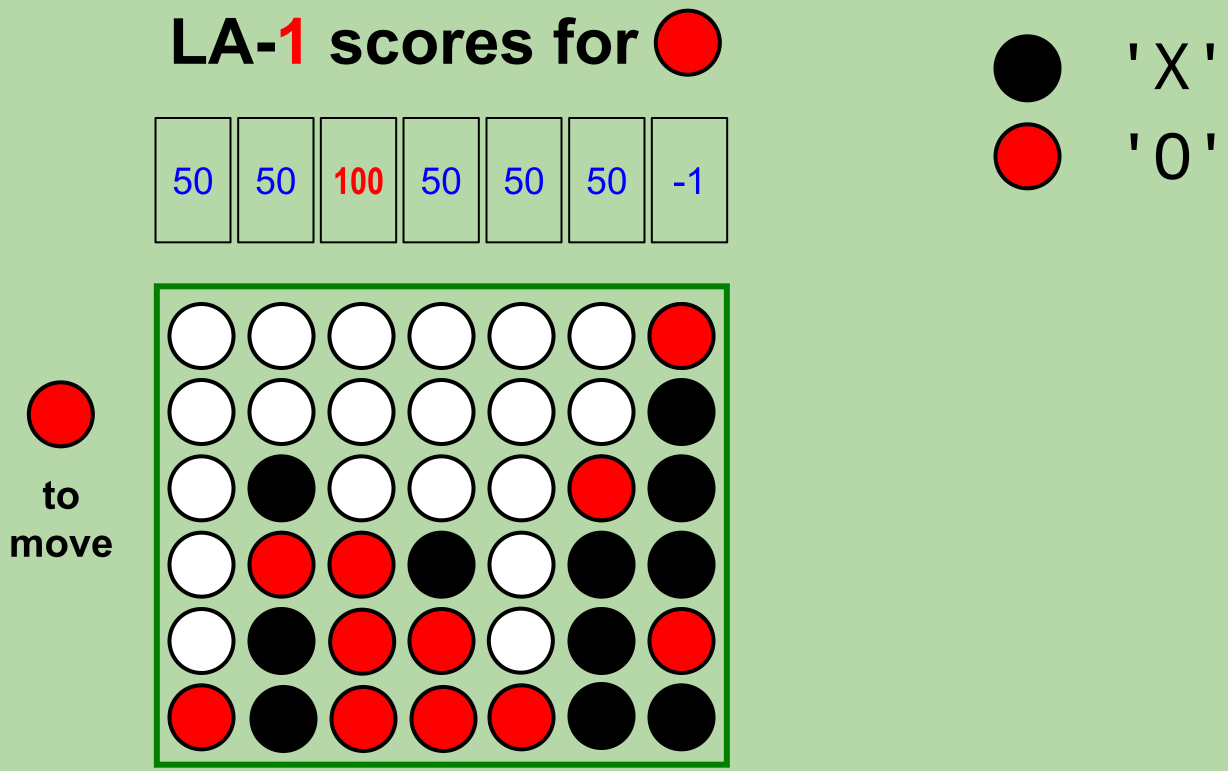
- A lookahead-0 player only assesses the current board (0 moves!).

same board,
different player,
same LA-0 scores



Example 2: LA-1

- A lookahead-1 player assesses the outcome of *only* the considered move.

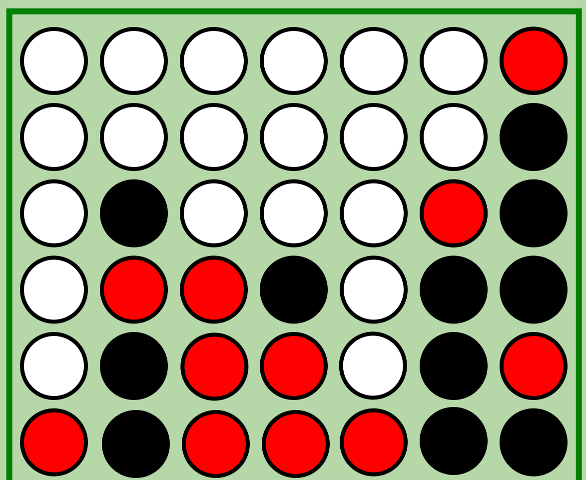


What Are the LA-2 Scores for ?

- Look 2 moves ahead. Assume the opponent looks 1 move ahead.

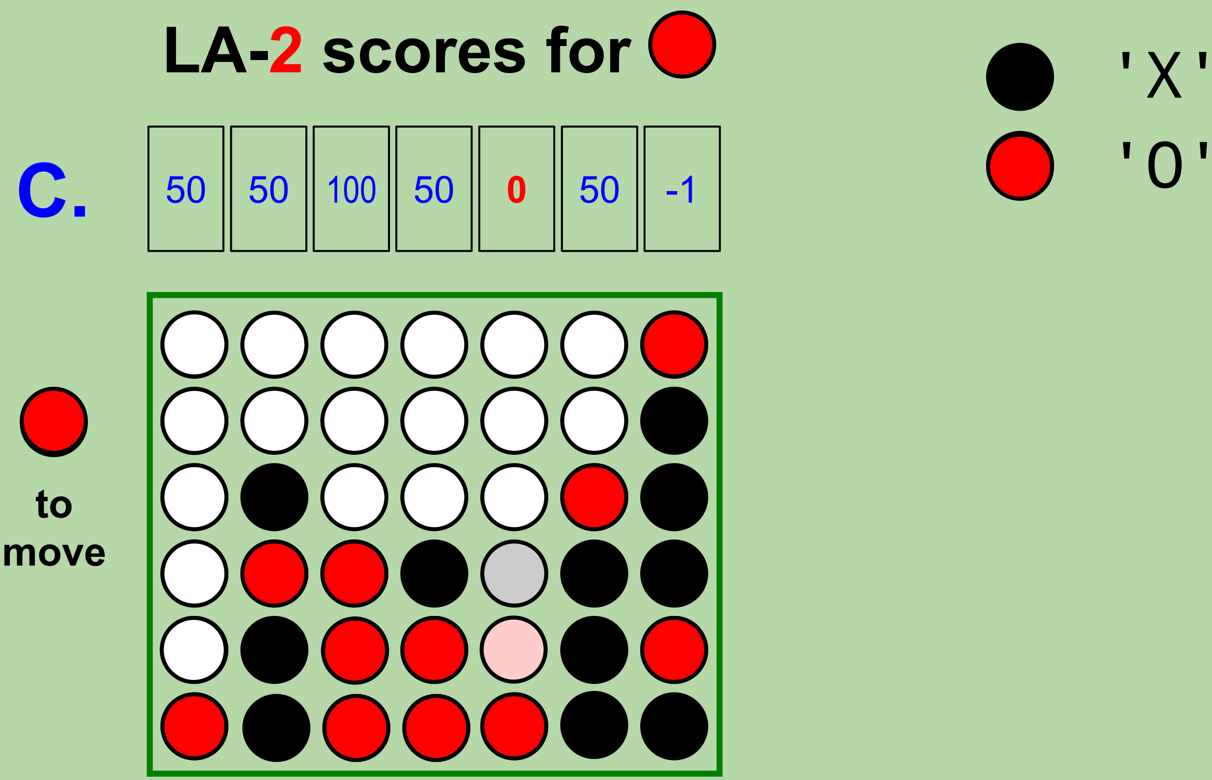
	50	50	100	50	50	50	-1	← LA-1 scores
A.	50	50	100	50	50	50	-1	← no change?
B.	0	0	100	0	0	0	-1	
C.	50	50	100	50	0	50	-1	

to move



Example 2: LA-2

- A lookahead-2 player looks 2 moves ahead.
  - what if I make this move, and then my opponent makes *its best move*?
  - ***note:*** we assume the opponent looks ahead 2 – 1 = 1 move



Example 2: LA-3

- A lookahead-3 player looks 3 moves ahead.
  - what if I make this move, and then my opponent makes its best move, *and then I make my best subsequent move?*
  - ***note:*** we assume the opponent looks ahead 3 – 1 = 2 moves

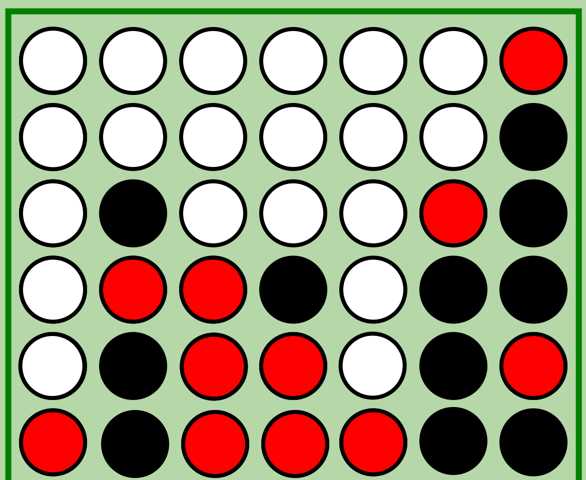
LA-2 scores for

'X'

'O'

What would change?

50	50	100	50	0	50	-1



to move

LA-3!

- A lookahead-3 player looks 3 moves ahead.
  - what if I make this move, and then my opponent makes its best move, *and then I make my best subsequent move?*
  - ***note:*** we now assume the opponent looks ahead 2 moves

What would change?
nothing

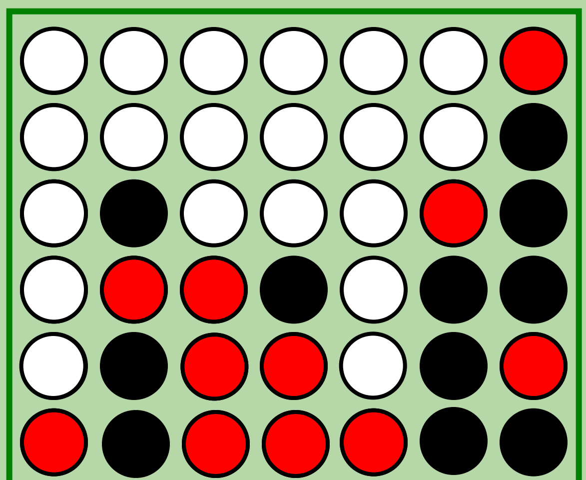
LA-3 scores for

50	50	100	50	0	50	-1

'X'

'O'

to move



LA-4!

- A lookahead-4 player looks 4 moves ahead.
  - assumes the opponent looks ahead 4 – 1 = 3 moves

LA-3 scores for

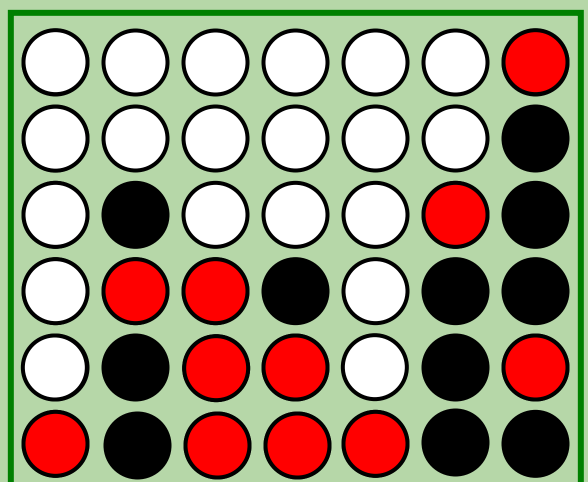
'X'

'O'

What would change?

50	50	100	50	0	50	-1

to move

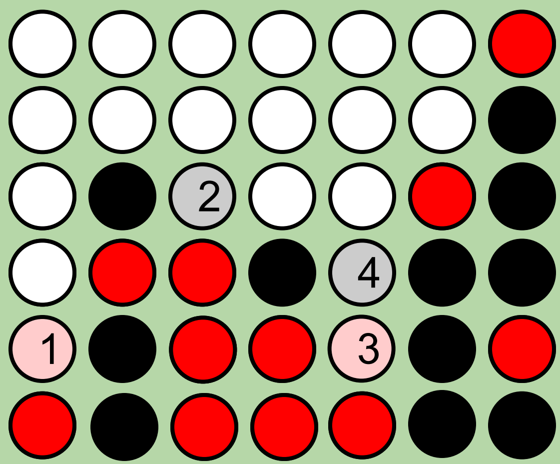


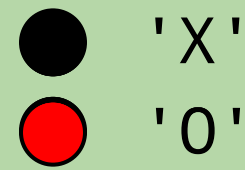
LA-4!

- A lookahead-4 player looks 4 moves ahead.
  - assumes the opponent looks ahead 4 – 1 = 3 moves

LA-4 scores for

0	0	100	0	0	0	-1



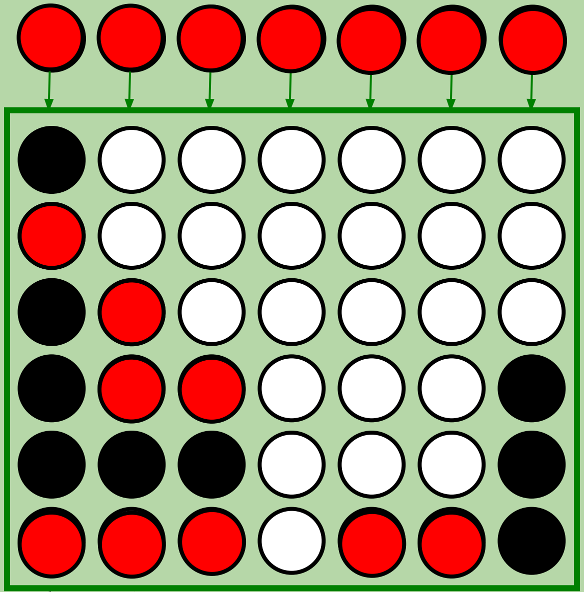


Consider column 0:

1. 'O' moves there.
2. 'X' moves to 2.
3. 'O' moves to 4 to block a diagonal win.
4. 'X' still wins horizontally!

Same thing holds for the other col's with new 0s.

Try It!



'O'

you - `self` - is playing 'O'

'X'

LA-0 scores for 'O':

Looks 0 moves into the future

col 0	col 1	col 2	col 3	col 4	col 5	col 6

LA-1 scores for 'O':

Looks 1 move into the future

col 0	col 1	col 2	col 3	col 4	col 5	col 6

LA-2 scores for 'O':

Looks 2 moves into the future

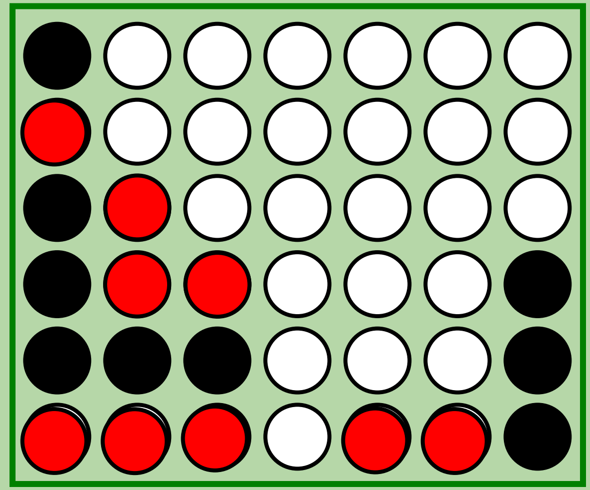
col 0	col 1	col 2	col 3	col 4	col 5	col 6

LA-3 scores for 'O':

Looks 3 moves into the future

col 0	col 1	col 2	col 3	col 4	col 5	col 6

Solutions



'O'

you - `self` - is playing 'O'

'X'

	col 0	col 1	col 2	col 3	col 4	col 5	col 6
LA-0 scores for **'O'**: Looks 0 moves into the future	-1	50	50	50	50	50	50
LA-1 scores for **'O'**: Looks 1 move into the future	-1	50	50	100	50	50	50
LA-2 scores for **'O'**: Looks 2 moves into the future	-1	0	0	100	0	0	50
LA-3 scores for **'O'**: Looks 3 moves into the future	-1	0	0	100	0	0	100

scores_for – the AI in AIPlayer!

```
def scores_for(self, board):
    """ MUST return a list of scores – one for each column!!
    """
    scores = [50] * board.width

    for col in range(board.width):

        ???

    return scores
```

Suppose you're playing with LA 2...

For each column:

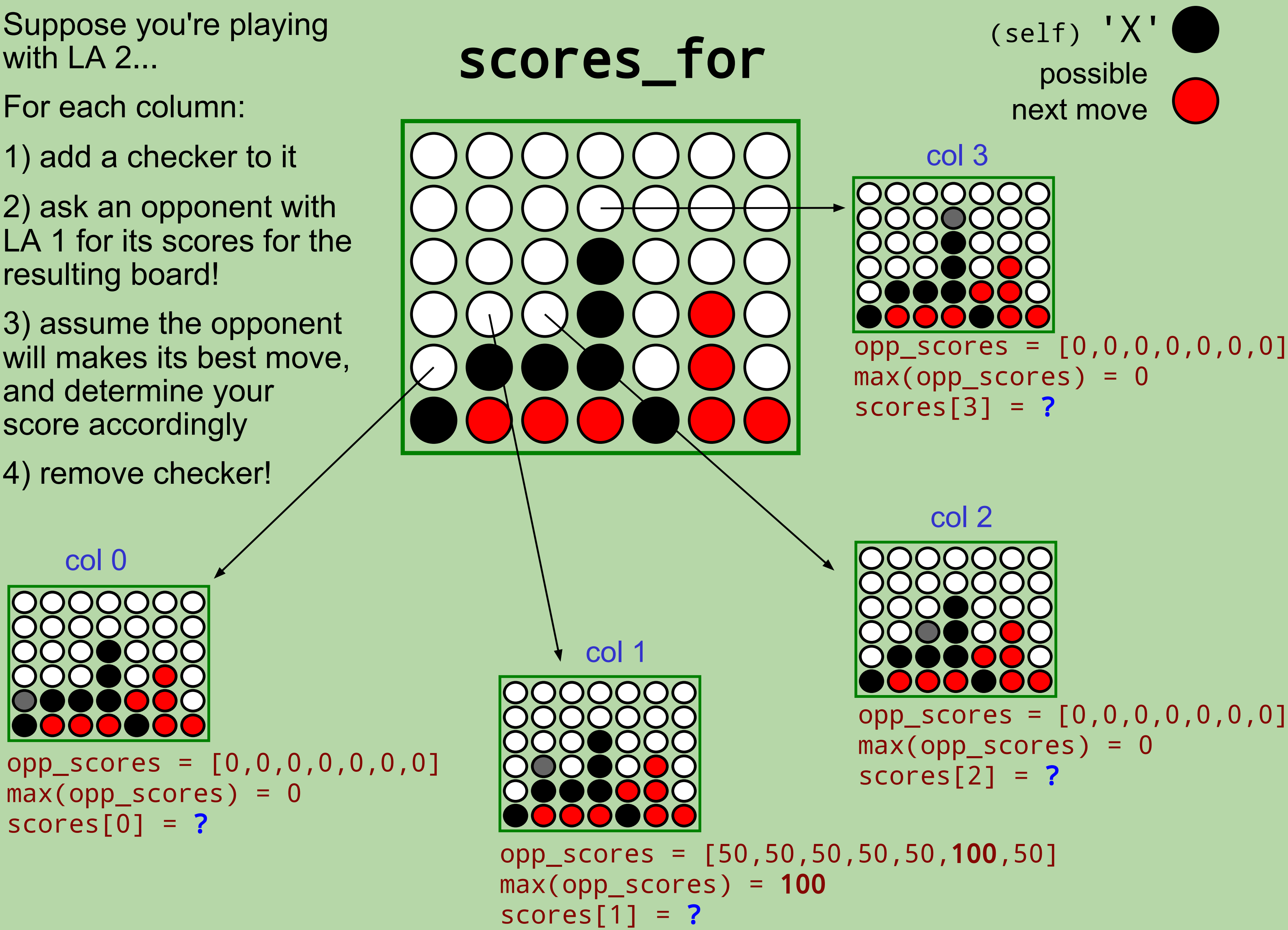
1) add a checker to it

2) ask an opponent with LA 1 for its scores for the resulting board!

3) assume the opponent will makes its best move, and determine your score accordingly

4) remove checker!

scores_for



scores_for

Suppose you're playing with LA 2...

For each column:

1) add a checker to it

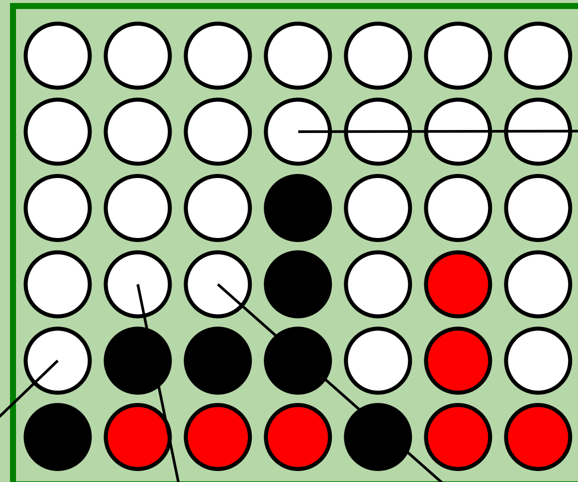
2) ask an opponent with LA 1 for its scores for the resulting board!

3) assume the opponent will makes its best move, and determine your score accordingly

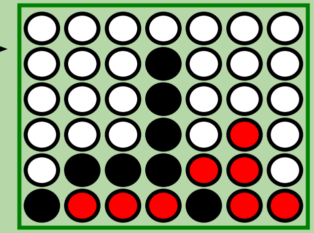
4) remove checker!

`(self)` 'X'

possible next move

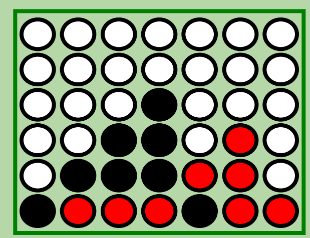


col 3



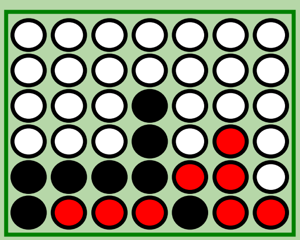
```
opp_scores = [0,0,0,0,0,0,0]
max(opp_scores) = 0
scores[3] = 100
```

col 2



```
opp_scores = [0,0,0,0,0,0,0]
max(opp_scores) = 0
scores[2] = 100
```

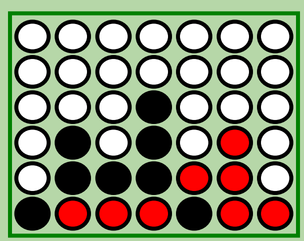
col 0



```
opp_scores = [0,0,0,0,0,0,0]
max(opp_scores) = 0
scores[0] = 100
```

A loss for my opponent is a win for me!

col 1



```
opp_scores = [50,50,50,50,50,100,50]
max(opp_scores) = 100
scores[1] = 0
```

A win for my opponent is a loss for me!

scores_for

Suppose you're playing with LA 2...

(self) 'X'

possible next move

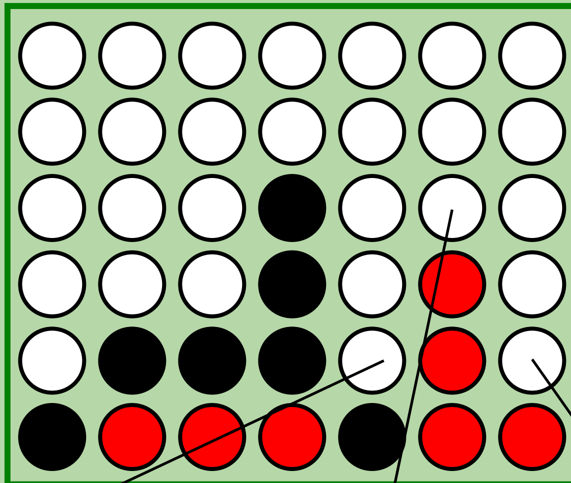
For each column:

1) add a checker to it

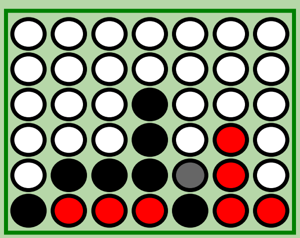
2) ask an opponent with LA 1 for its scores for the resulting board!

3) assume the opponent will makes its best move, and determine your score accordingly

4) remove checker!

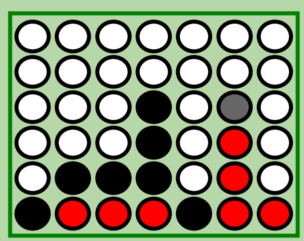


col 4



```
opp_scores = [0,0,0,0,0,0,0]
max(opp_scores) = 0
scores[4] = 100
```

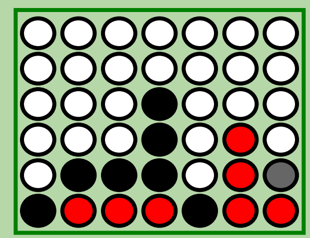
col 5



```
opp_scores = [50,50,50,50,50,50,50]
max(opp_scores) = 50
scores[5] = 50
```

A draw for my opponent is a draw for me!

col 6



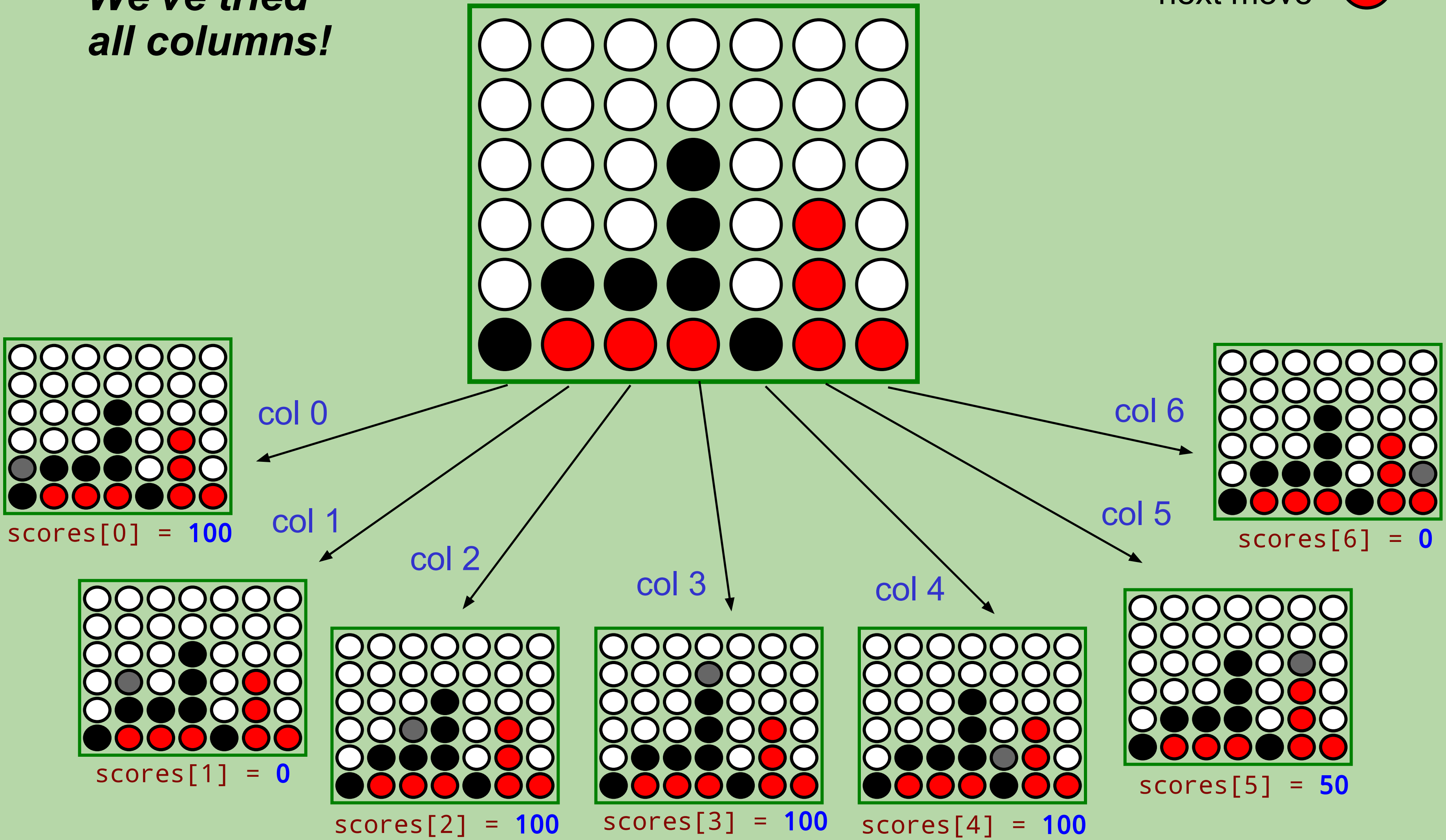
```
opp_scores =
[50,50,50,50,50,100,50]
max(opp_scores) = 100
scores[6] = 0
```

Suppose you're playing
with LA 2...

***We've tried
all columns!***

scores_for

(self) 'X'
possible
next move



`scores = [100, 0, 100, 100, 100, 50, 0]`

scores_for – the AI in AIPlayer!

```
def scores_for(self, board):
    """ MUST return a list of scores – one for each column!!
    """
    scores = [50] * board.width

    for col in range(board.width):
        if col is full:
            use -1 for scores[col]
        elif already win/loss:
            use appropriate score (100 or 0)
        elif lookahead is 0:
            use 50
        else:
            try col – adding a checker to it
            create an opponent with self.lookahead – 1
            opp_scores = opponent.scores_for(...)
            scores[col] = ???
            remove checker

    return scores
```

scores_for – the AI in AIPlayer!

```
def scores_for(self, board):
    """ MUST return a list of scores – one for each column!!
    """
    scores = [50] * board.width

    for col in range(board.width):
        if col is full:
            use -1 for scores[col]
        elif already win/loss:
            use appropriate score (100 or 0)
        elif lookahead is 0:
            use 50
        else:
            try col – adding a checker to it
            create an opponent with self.lookahead – 1
            opp_scores = opponent.scores_for(...)
            scores[col] = ???
            remove checker


    return scores
```

scores_for – the AI in AIPlayer!

```
def scores_for(self, board):
    """ MUST return a list of scores – one for each column!!
    """
    scores = [50] * board.width

    for col in range(board.width):
        if col is full:
            use -1 for scores[col]
        elif already win/loss:
            use appropriate score (100 or 0)
        elif lookahead is 0:
            use 50
        else:
            try col – adding a checker to it
            create an opponent with self.lookahead – 1
            opp_scores = opponent.scores_for(...)
            scores[col] = ???
            remove checker


    return scores
```

scores_for – the AI in AIPlayer!

```
def scores_for(self, board):
    """ MUST return a list of scores – one for each column!!
    """
    scores = [50] * board.width

    for col in range(board.width):
        if col is full:
            use -1 for scores[col]
        elif already win/loss:
            use appropriate score (100 or 0)
        elif lookahead is 0:
            use 50
        else:
            try col – adding a checker to it
            create an opponent with self.lookahead – 1
            opp_scores = opponent.scores_for(...)
            scores[col] = ???
            remove checker


    return scores
```

scores_for – the AI in AIPlayer!

```
def scores_for(self, board):
    """ MUST return a list of scores – one for each column!!
    """
    scores = [50] * board.width

    for col in range(board.width):
        if col is full:
            use -1 for scores[col]
        elif already win/loss:
            use appropriate score (100 or 0)
        elif lookahead is 0:
            use 50
        else:
            try col – adding a checker to it
            create an opponent with self.lookahead – 1
            opp_scores = opponent.scores_for(...)
            scores[col] = ???
            remove checker


    return scores
```

scores_for – the AI in AIPlayer!

```
def scores_for(self, board):
    """ MUST return a list of scores – one for each column!!
    """
    scores = [50] * board.width

    for col in range(board.width):
        if col is full:
            use -1 for scores[col]
        elif already win/loss:
            use appropriate score (100 or 0)
        elif lookahead is 0:
            use 50
        else:
            try col – adding a checker to it
            create an opponent with self.lookahead – 1
            opp_scores = opponent.scores_for(...)
            scores[col] = ???
            remove checker


    return scores
```

scores_for – the AI in AIPlayer!

```
def scores_for(self, board):
    """ MUST return a list of scores – one for each column!!
    """
    scores = [50] * board.width

    for col in range(board.width):
        if col is full:
            use -1 for scores[col]
        elif already win/loss:
            use appropriate score (100 or 0)
        elif lookahead is 0:
            use 50
        else:
            try col – adding a checker to it
            create an opponent with self.lookahead – 1
            opp_scores = opponent.scores_for(...)
            scores[col] = ???
            remove checker


    return scores
```

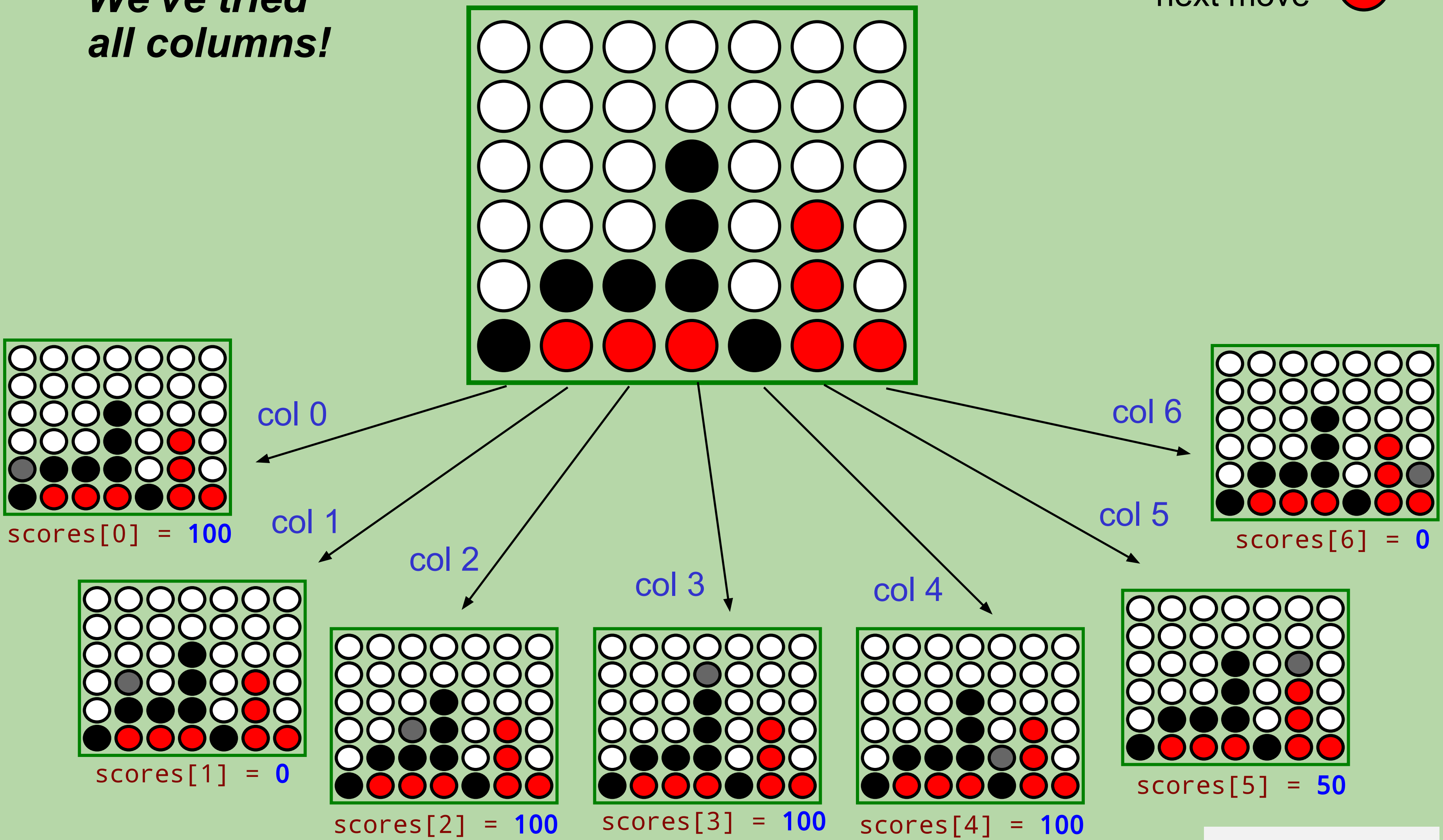
Suppose you're playing with LA 2...

We've tried all columns!

scores_for

(self) 'X'

possible next move



`scores=[100, 0, 100, 100, 100, 50, 0]`

What should `next_move` return?

scores_for – the AI in AIPlayer!

```
def scores_for(self, board):
    """ MUST return a list of scores – one for each column!!
    """
    scores = [50] * board.width

    for col in range(board.width):
        if col is full:
            use -1 for scores[col]
        elif already win/loss:
            use appropriate score (100 or 0)
        elif lookahead is 0:
            use 50
        else:
            try col – adding a checker to it
            create an opponent with self.lookahead – 1
            opp_scores = opponent.scores_for(...)
            scores[col] = ???
            remove checker


    return scores
```

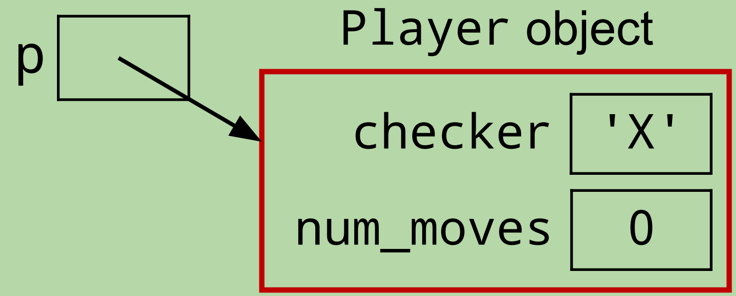
scores_for – the AI in AIPlayer!

```
def scores_for(self, board):
    """ MUST return a list of scores – one for each column!!
    """
    scores = [50] * board.width

    for col in range(board.width):
        if col is full:
            use -1 for scores[col]
        elif already win/loss:
            use appropriate score (100 or 0)
        elif lookahead is 0:
            use 50
        else:
            try col – adding a checker to it
            create an opponent with self.lookahead – 1
            opp_scores = opponent.scores_for(...)
            scores[col] = ???
            remove checker


    return scores
```

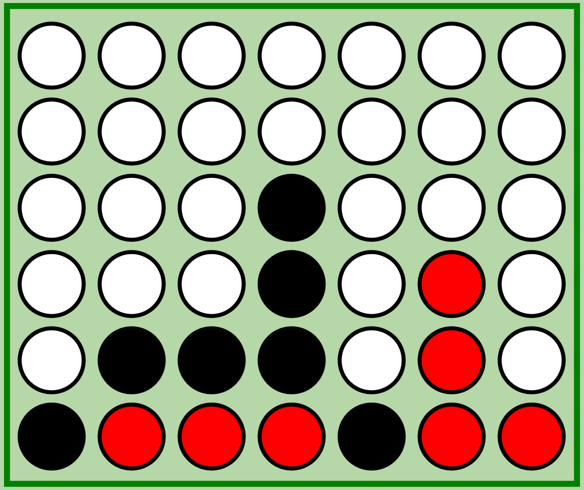
RandomPlayer, AIPlayer Class

```
class Player:

    def __init__(self, checker):
        ...

    def __repr__(self):
        ...

    def opponent_checker(self):
        ...

    def next_move(self, board):
        self.num_moves += 1
        scores = self.scores_for(board)
        return ???
```

p = Player('X')



Breaking Ties

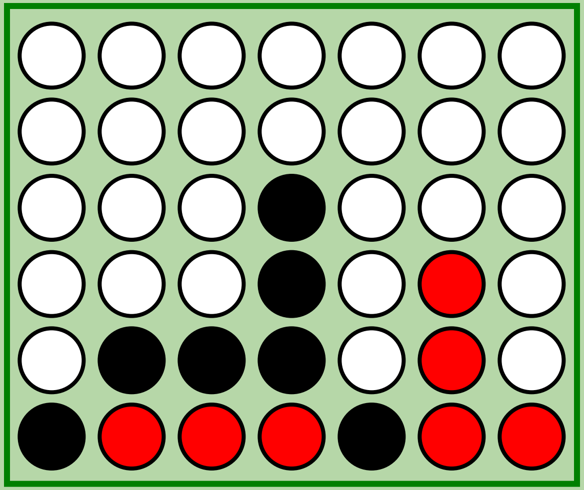


0 1 2 3 4 5 6

```
scores = [100, 0, 100, 100, 100, 50, 0]
```

- possible moves: `???`

Breaking Ties



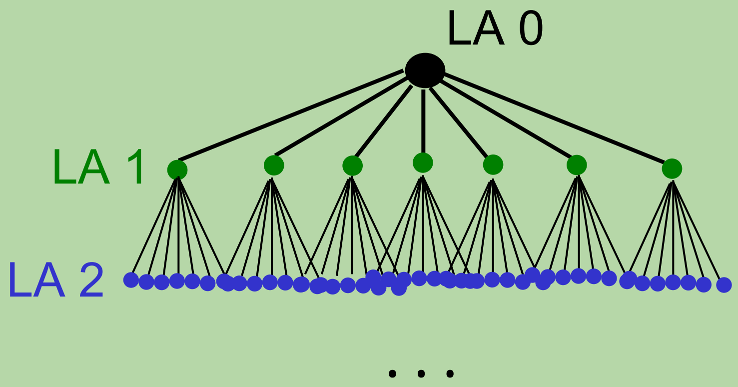
```
0     1     2        3       4       5     6
scores = [100, 0, 100, 100, 100, 50, 0]
```

- possible moves: `[0, 2, 3, 4]`
- `self.tiebreak == 'LEFT': return 0`
- `self.tiebreak == 'RIGHT': return 4`
- `self.tiebreak == 'RANDOM': choose at random!`

Connect Four Complexity

How Many Outcomes Are Considered?

- On average, Connect 4 players have **seven** choices per move.



- LA-0 player considers 1 outcome.
  - the current board
- LA-1 player considers 7 outcomes.
- LA-2 player considers 7^2 outcomes.
  - each of its 7 moves, followed by each of its opponent's 7 moves
- LA-*n* player considers 7^n outcomes.

it's okay if your times are longer!

- As LA increases, time taken by `next_move` grows exponentially!

```
>>> AIPlayer('X','RANDOM',5).next_move(Board(6,7))  # 1.1 sec
>>> AIPlayer('X','RANDOM',6).next_move(Board(6,7))  # 7.1 sec
>>> AIPlayer('X','RANDOM',7).next_move(Board(6,7))  # 49.1 sec
>>> AIPlayer('X','RANDOM',8).next_move(Board(6,7))  # 341.8 sec
>>> AIPlayer('X','RANDOM',9).next_move(Board(6,7))  # ~40 min!!
```