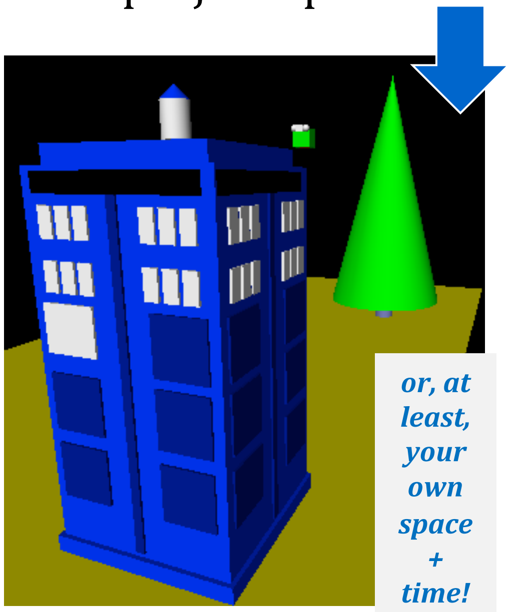
Intelligent CS?!

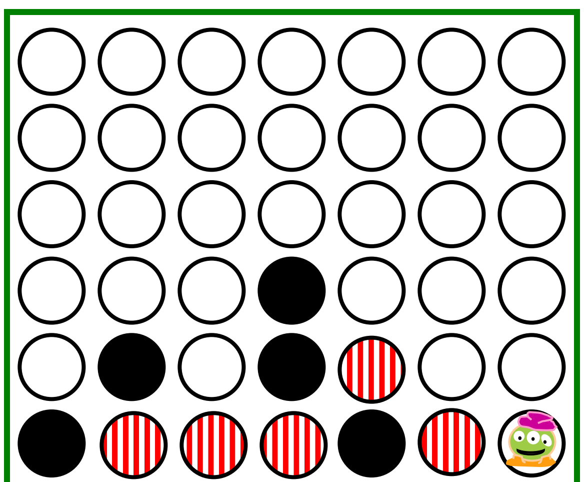
Final project option #1



Final project option #2

vPython + 2-ply AI!

Hw11 due **Tuesday** @ 22:22:22



b.playGame('human', o0rn), x3rn vs. 'human'

- X to move.

Is there a way to *ensure* a win?

If so, ***how far ahead***?

```
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | |X| | | |
| |X| |X|O| | |
|X|O|O|O|X|O| |
---------------
 0 1 2 3 4 5 6
```

Connect 4 AI ~ how *could* it work?

```
|X|X| | | | | |
|X|O| | |O| |X|
|O|O|X|O|X|O|O|
|X|X|X|O|O|X|X|
|O|O|O|X|O|O|X|
|O|X|X|X|O|X|O|
---------------
 0 1 2 3 4 5 6
```

`randomC4(ply=0)`

Who won?!

Oh, I won!

It could just play randomly... *Let's try!*

C4 AI ~ how *could* it work?

```
|X|X| | | | | |
|X|O| | |O| |X|
|O|O|X|O|X|O|O|
|X|X|X|O|O|X|X|
|O|O|O|X|O|O|X|
|O|X|X|X|O|X|O|
---------------
 0 1 2 3 4 5 6
```

Who won?!

```
while True:

    col = -1
    while b.allowsMove(col) == False:
        col = random.choice(range(7))

    b.addMove(ox, col)

    if ox == 'O':    ox = 'X'
    else:            ox = 'O'

    # check if game is over!
```

It could just play randomly... *Let's try!*

Or, it could always play as far <u>left</u> as possible... *Let's try that, too!*

C4 AI ~ how *could* it work?

```
|X|X| | | | | |
|X|O| | |O| |X|
|O|O|X|O|X|O|O|
|X|X|X|O|O|X|X|
|O|O|O|X|O|O|X|
|O|X|X|X|O|X|O|
---------------
 0 1 2 3 4 5 6
```

tiebreaking to the LEFT when possible...

```
|O|O|O| | | | |
|X|X|X| | | | |
|O|O|O| | | | |
|X|X|X| | | | |
|O|O|O| | | | |
|X|X|X|X| | | |
---------------
 0 1 2 3 4 5 6
```

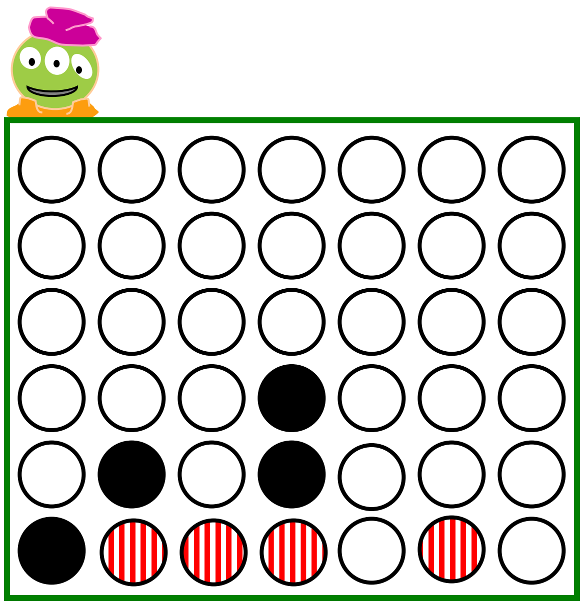
`leftC4(ply=0)`

not Overly strategic...

Or, it could always play as far left as possible... *Let's try that, too!*

C4 AI ~ how ***should*** it work?

I feel ahead of the game here...



```
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | |X| | | |
| |X| |X| | | |
|X|O|O|O| |O| |
---------------
 0 1 2 3 4 5 6
```

b12

b12.aiMove('O')

b12.aiMove('X')

It should **(1)** win and **(2)** *block* wins, if possible.

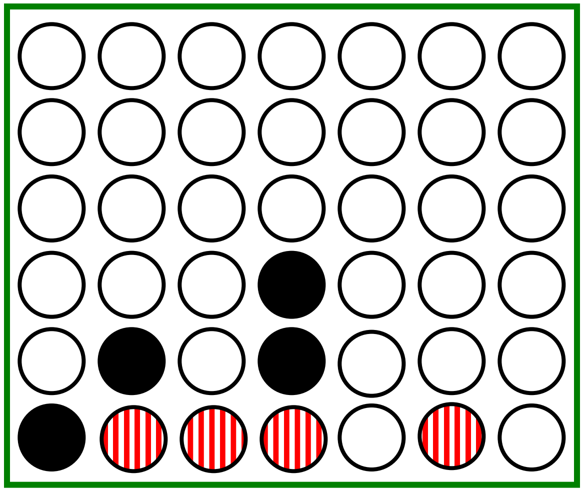
hw11pr2

Otherwise it should just play as well as it can... ?!

EC +
C4 tourney

Connect **4**, Part **2**

hw11pr2.py



```
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | |X| | | |
| |X| |X| | | |
|X|O|O|O| |O| |
---------------
 0 1 2 3 4 5 6
```

1 "ply"

`colsToWin( self, ox )`

`b.colsToWin('O')`

`b.colsToWin('X')`

2 "ply" + intuition-based tiebreaking

`aiMove( self, ox )`

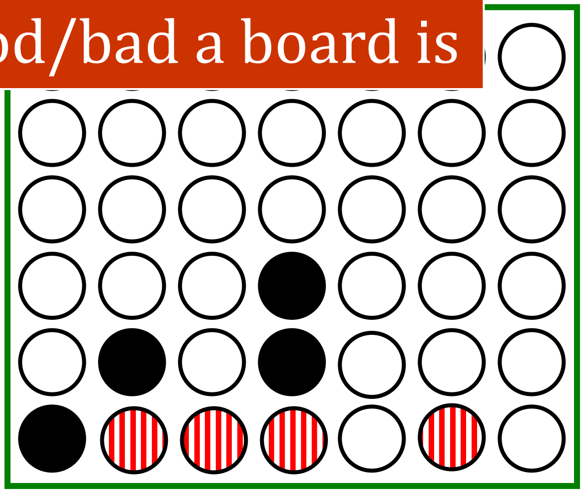
`b.aiMove('O')`

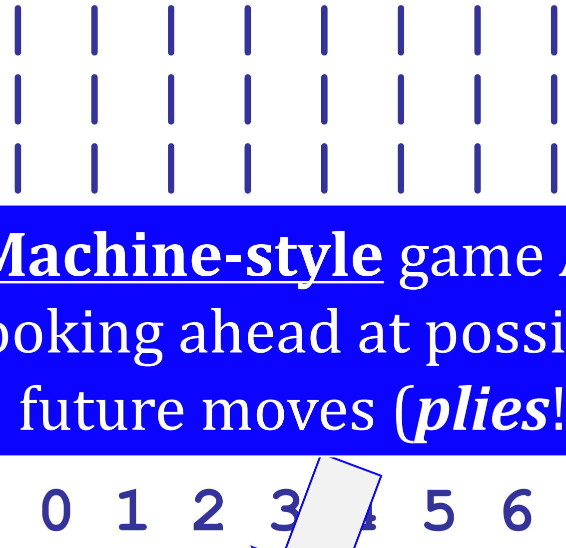
`b.aiMove('X')`

`hostGame( self )`

C4 AI ~ how *should* it work?

Human-style game AI: "intuitive" evaluation of how good/bad a board is



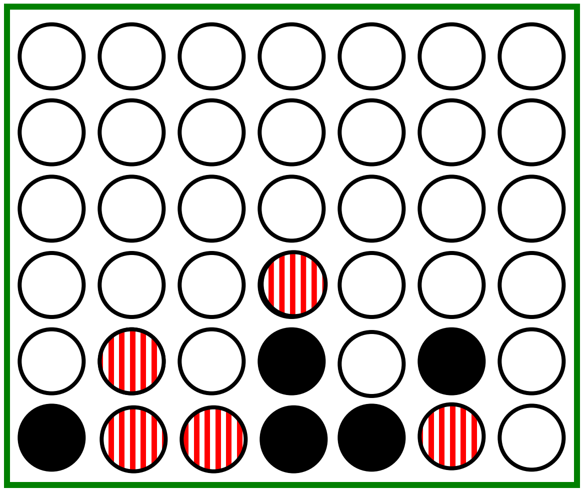


Machine-style game AI: looking ahead at possible future moves (***plies***!)

It should **(1)** win and **(2)** *block* wins, if possible.

Otherwise it should just play as well as it can... ?!

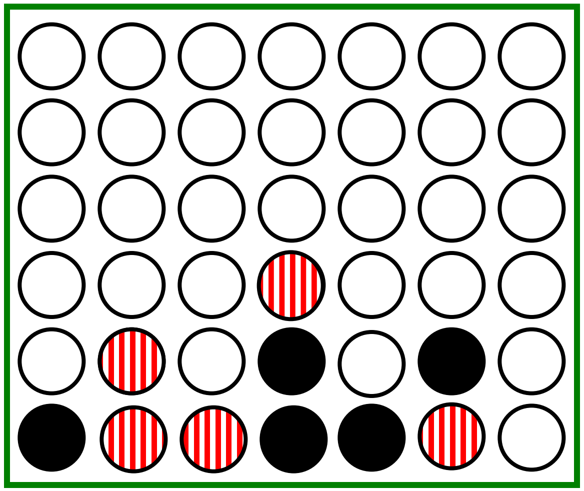
C4 AI ~ "intuitive" moves?



```
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | |O| | | |
| |O| |X| |X| |
|X|O|O|X|X|O| |
---------------
 0 1 2 3 4 5 6
```

If there isn't a win or loss... where should you go? *Why?*

C4 AI ~ "intuitive" moves?



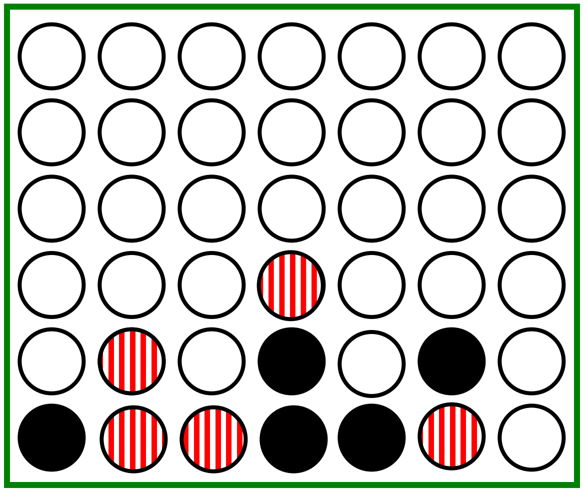
```
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | |O| | | |
| |O| |X| |X| |
|X|O|O|X|X|O| |
---------------
 0 1 2 3 4 5 6
```

Is there difference between these two?

```
for col in range(W):
  if b.allowsMove(col):
    return col
```

```
for col in [3,4,2,5,1,6,0]:
  if b.allowsMove(col):
    return col
```

C4 AI ~ "intuitive" moves?



```
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | |O| | | |
| |O| |X| |X| |
|X|O|O|X|X|O| |
---------------
 0 1 2 3 4 5 6
```

Difference: ***tie-breaking****!*

```
[0,1,2,3,4,5,6]
```

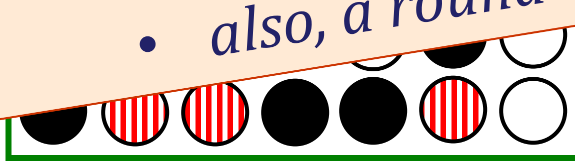
```
for col in range(W):
  if b.allowsMove(col):
    return col
```

```
for col in [3,4,2,5,1,6,0]:
  if b.allowsMove(col):
    return col
```

C4 AI ~ "intuitive" moves?

We'll run a C4 tournament with all of the `aiMove`s submitted...

- *(ex. cr.) better than random? +5*
- *also, a round-robin!*



```
|   |   |   |O|   |   |   |
|   |O|   |X|   |X|   |
|X|O|O|X|X|O|   |
---------------
 0 1 2 3 4 5 6
```

*If there isn't a win or loss... maybe we just haven't looked **far enough ahead**!?!*

Machine-style game AI: looking ahead at possible future moves (***plies***!)

"Plies" ~ turns of "lookahead"

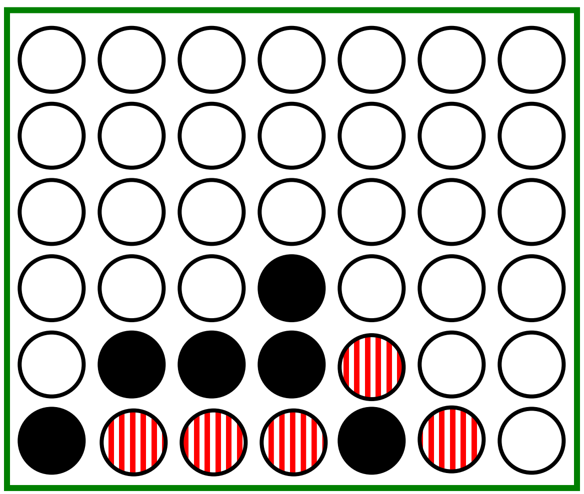
Machine-style game AI: looking ahead at possible future moves (***plies***!)

At <u>ZERO</u> ply, every allowable move looks the same!

<u>Zero ply</u> is <u>no</u> lookahead at all!

legal but random moves...

b1



At **zero ply**, a player will NOT see this win!!

```
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | |X| | | |
| |X|X|X|O| | |
|X|O|O|O|X|O| |
---------------
 0 1 2 3 4 5 6
```

b1

x0.scoresFor(b1)

zero_ply.scoresFor(b1)

One ply: check for <u>win</u>

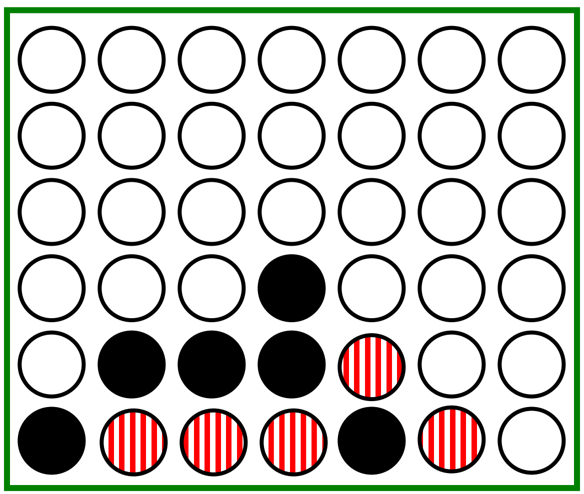
Machine-style game AI: looking ahead at possible future moves (***plies***!)

At <u>ONE</u> ply, the machine can detect <u>its own wins</u>!

wins when possible...

```
Imagine 'X' ● at ONE ply...
```

b1



At **one ply**, a player <u>WILL</u> see this win!!

```
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | |X| | | |
| |X|X|X|O| | |
|X|O|O|O|X|O| |
---------------
 0 1 2 3 4 5 6
```

b1

x0.scoresFor(b1)

zero_ply.scoresFor(b1)

Two plies: look to block

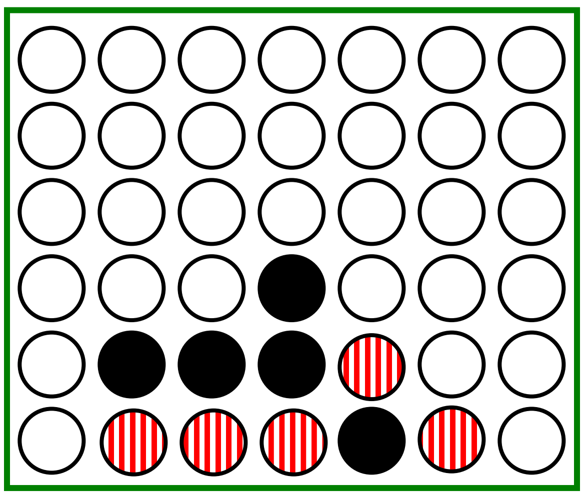
Machine-style game AI: looking ahead at possible future moves (***plies***!)

At TWO ply, the machine can detect opponent threats!

blocks when possible...

Imagine 'X' ⚫ at TWO ply...

b1b



At **two ply**, a player WILL see this threat!

```
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | |X| | | |
| |X|X|X|O| | |
| |O|O|O|X|O| |
---------------
 0 1 2 3 4 5 6
```

b1

x0.scoresFor(b1)

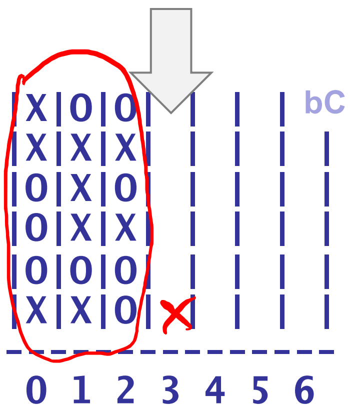
zero_ply.scoresFor(b1)

Plying our intuitions...

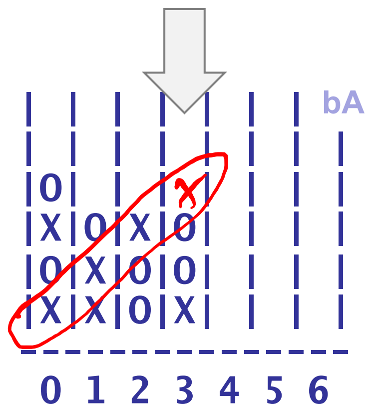
In all 4 of these boards, **X** will move to col 3, *even if <u>both players tiebreak to the **LEFT**</u>* Why?

Find + circle *the reason **why*** **X** *moves to col. #3 for each...*

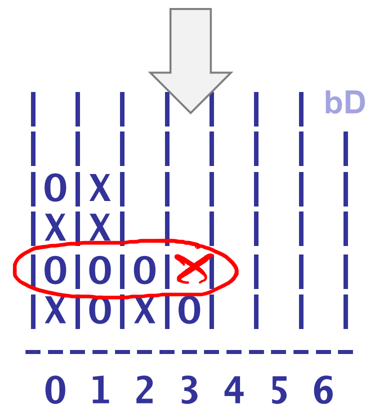
Name(s) ________________



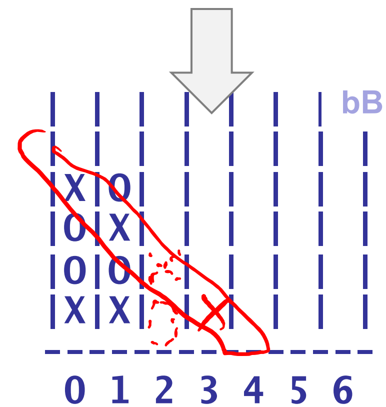
ply == 0



ply == 1



ply == 2

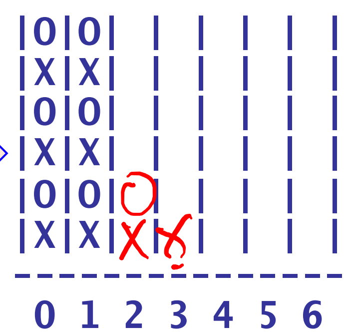


ply == 3

Plus: full-game challenges...

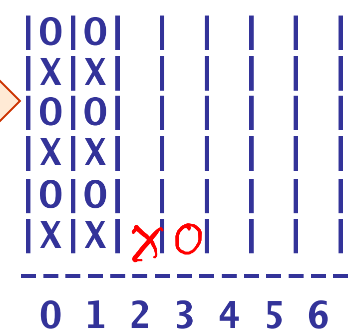
Challenge #1: What are the <u>next three</u> moves? It's **X**'s turn, and both **X** and **O** are playing at **1 ply**, tiebreaking LEFT?

next to move: X



Challenge #2: What are the <u>next three</u> moves? It's **X**'s turn, both **X** and **O** are at **2 ply**, tiebreaking to the LEFT? Who wins?

next to move: X



Plying our intuitions...

In all 4 of these boards, **X** will move to col 3, *even if <u>both players tiebreak to the</u>* **LEFT**

Try this on the back page first...

Find + circle *the reason* ***<u>why</u>*** *'X' moves to col. #3 for each...*

No lookahead! — bC

```
|X|O|O| | | | |
|X|X|X| | | | |
|O|X|O| | | | |
|O|X|X| | | | |
|O|O|O| | | | |
|X|X|O|X| | | |
---------------
 0 1 2 3 4 5 6
```

ply == 0

win, if possible — bA

```
| | | | | | | |
| | | | | | | |
|O| | |X| | | |
|X|O|X|O| | | |
|O|X|O|O| | | |
|X|X|O|X| | | |
---------------
 0 1 2 3 4 5 6
```

ply == 1

block, if possible — bD

```
| | | | | | | |
| | | | | | | |
|O|X| | | | | |
|X|X| | | | | |
|O|O|O|X| | | |
|X|O|X|O| | | |
---------------
 0 1 2 3 4 5 6
```

ply == 2

set up a "checkmate"! — bB

```
| | | | | | | |
| | | | | | | |
|X|O| | | | | |
|O|X| | | | | |
|O|O| | | | | |
|X|X| |X| | | |
---------------
 0 1 2 3 4 5 6
```

ply == 3

Challenge: What will happen if you run **X at 1 ply** and **O at 1 ply**, each tiebreaking LEFT?

next to move: X

```
|O|O| | | | | |
|X|X| | | | | |
|O|O| | | | | |
|X|X| | | | | |
|O|O|O| | | | |
|X|X|X|X| | | |
---------------
 0 1 2 3 4 5 6
```

Challenge #2: What about **2-ply** for each of **X** and **O**?

next to move: X

```
|O|O|X|X|O|O| |
|X|X|O|O|X|X| |
|O|O|X|X|O|O| |
|X|X|O|O|X|X| |
|O|O|X|O|O|O|O|
|X|X|X|O|X|X|X|
---------------
 0 1 2 3 4 5 6
```

Plying our intuitions...

In all 4 of these boards, **X** will move to col 3, *even if both players tiebreak to the* **LEFT**

Find + circle *the reason* ***why*** *'X' moves to col. #3 for each...*

Try this on the back page first...

No lookahead!

bC

```
|X|O|O| | | | |
|X|X|X| | | | |
|O|X|O| | | | |
|O|X|X| | | | |
```

bA

bD

PLY THESE NORTHWARD...

Ch...
w...
ru...
O at 1 ply, each tiebreaking LEFT?

```
|X|X| | | | | |
|O|O| | | | | |
|X|X| | | | | |
---------------
 0 1 2 3 4 5 6
```

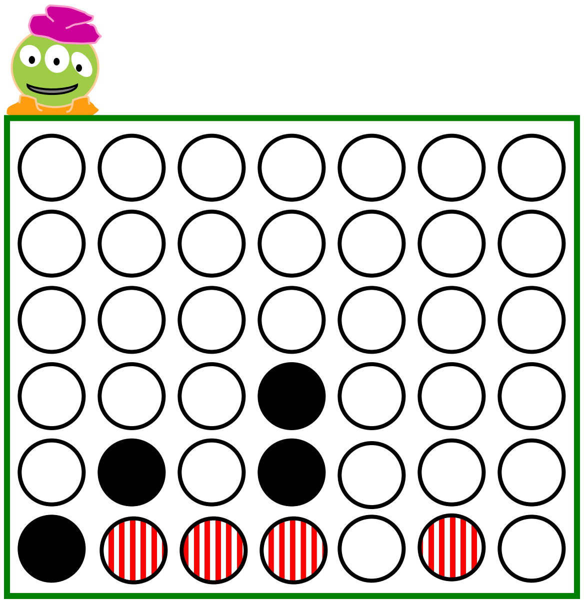
Challenge #2: What about **2-ply** for each of **X** and **O**?

```
|O|O| | | | | |
|X|X| | | | | |
|O|O| | | | | |
|X|X| | | | | |
|O|O| | | | | |
|X|X| | | | | |
---------------
 0 1 2 3 4 5 6
```

Let's try these!

C4 AI ~ *lookahead* moves...

I feel ahead of the game here...



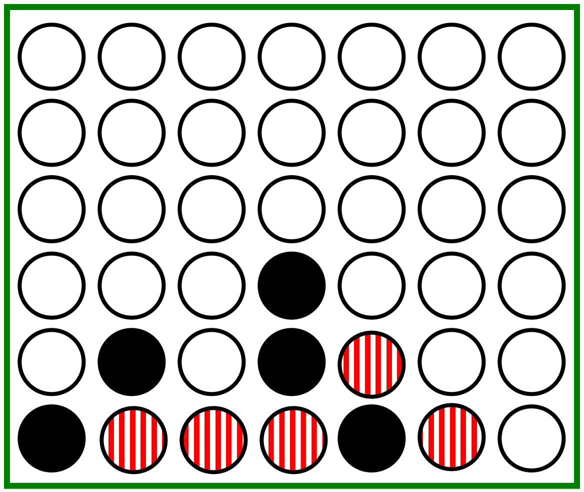
Both we – and machines – can look ahead ***much*** further than this!

0 1 2 3 4 5 6

It should **(1)** win and **(2)** *block* wins, when it can.

Otherwise it should just play as well as it can... ?!

How many ply?



How many moves ahead might we have to look?

```
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | |X| | | |
| |X| |X|O| | |
|X|O|O|O|X|O| |
---------------
 0 1 2 3 4 5 6
```

b0

x5.scoresFor(b0)

let run!

Looking further ahead… !!!

How many ply of lookahead would we need to play a ***perfect*** game of Connect Four?

And how is it going to “really work”?

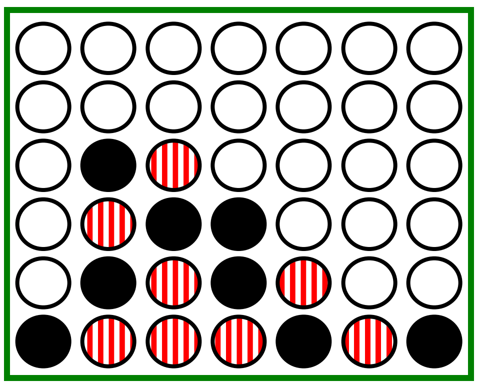
Arithmetizing C4…

‘x’

‘o’

A simple system:

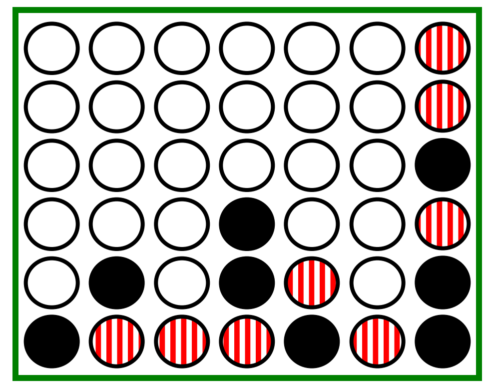
100.0	50.0	0.0
for a win	**for anything else**	**for a loss**



Score for ●

Score for

-1



Score for ●

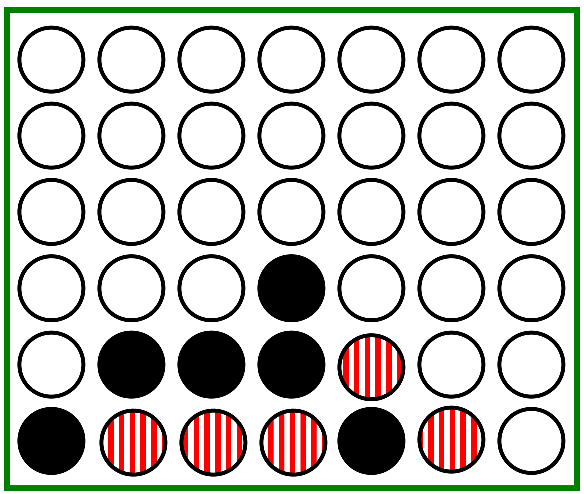
Score for

"*Plies*" ~ turns of "lookahead"

zero_ply is playing 'X' (black)

Every possible move will score a 50!

b1



At **zero ply**, a player will NOT see this win!!

```
 50 50 50 50 50 50 50
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | |X| | | |
| |X|X|X|O| | |
|X|O|O|O|X|O| |
---------------
 0 1 2 3 4 5 6
```

b1

x0.scoresFor(b1)

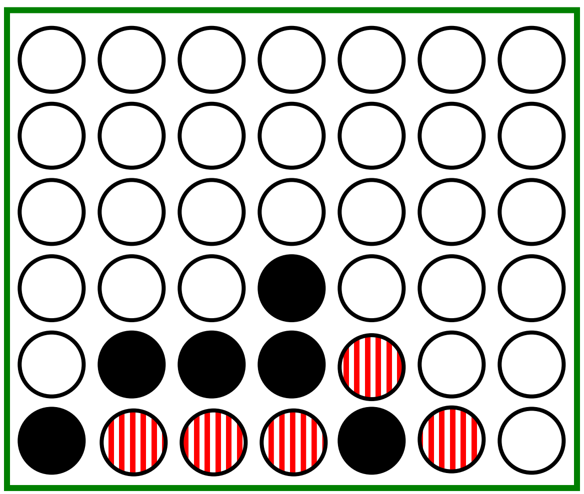
zero_ply.scoresFor(b1)

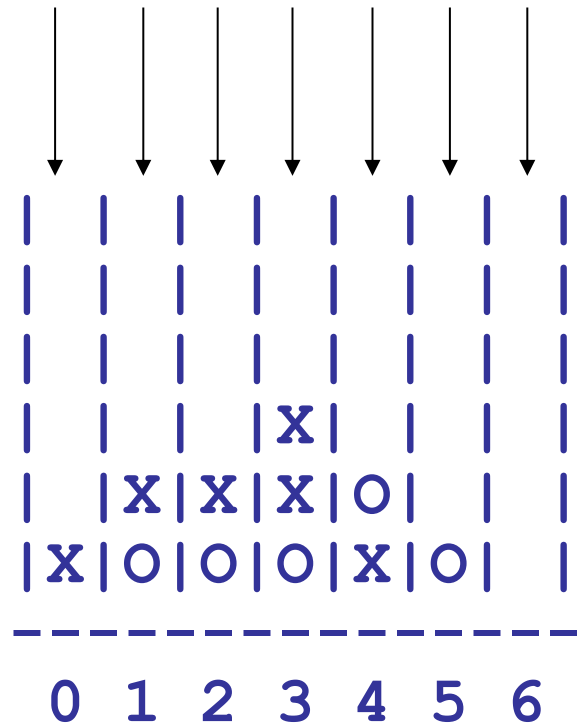
Zero Ply

zero_ply is playing 'X' (black)

```
zero_ply.scoresFor( b1 )  ⇨  [50,50,50,50,50,50,50]
```

b1



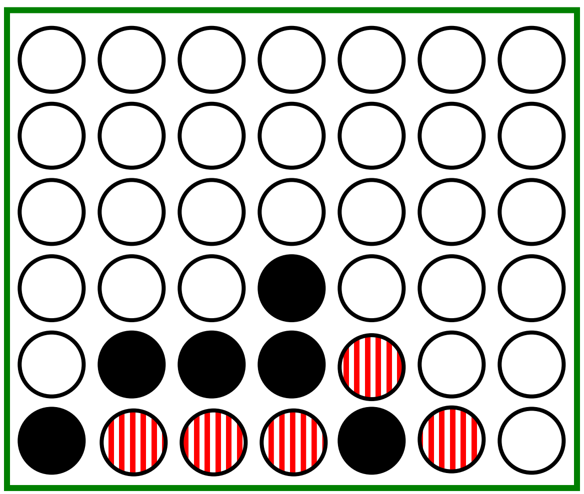


One Ply

```
one_ply is playing 'X' (black)
```

```
one_ply.scoresFor( b1 )  ➡  [100,50,50,50,50,50,50]
```

b1



At **one ply**, a player WILL see this win!!

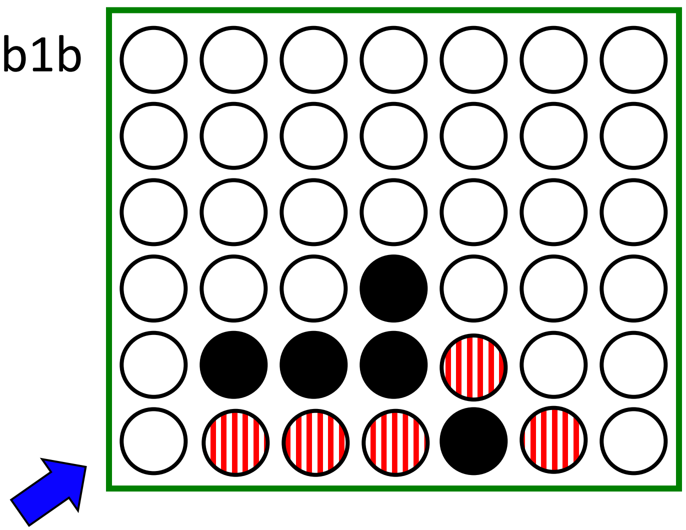
```
|   |   |   |   |   |   |   |
|   |   |   |   |   |   |   |
|   |   |   |   |   |   |   |
|   |   |   | X |   |   |   |
|   | X | X | X | O |   |   |
| X | O | O | O | X | O |   |
-----------------------------
  0   1   2   3   4   5   6
```

Two Ply

two_ply is playing 'X' (black)

```
two_ply.scoresFor( b1b ) ➡ [50, 0, 0, 0, 0, 0, 0]
```

b1b



At two ply, a player will see the chance for the OPPONENT ('O') to win

```
|   |   |   |   |   |   |   |
|   |   |   |   |   |   |   |
|   |   |   |   |   |   |   |
|   |   |   | X |   |   |   |
|   | X | X | X | O |   |   |
|   | O | O | O | X | O |   |
-----------------------------
  0   1   2   3   4   5   6
```

I want 3-ply!

Deep Blue (chess computer)

From Wikipedia, the free encyclopedia

Deep Blue was a chess-playing computer developed by IBM. On May 11, 1997, the machine, with human intervention between games, won the second six-game match against world champion Garry Kasparov by two wins to one with three draws.[1] Kasparov accused IBM of cheating and demanded a rematch, but IBM refused and dismantled Deep Blue.[2] Kasparov had beaten a previous version of Deep Blue in 1996.

Deep Blue

Contents [hide]

Origins [edit]

Deep Blue (chess computer)

From Wikipedia, the free encyclopedia

Deep Blue was a chess-playing computer developed by IBM. On May 11, 1997, the machine, with human intervention between games, won the second six-game match against world champion Garry Kasparov by two wins to one with three draws.[1] Kasparov accused IBM of cheating and demanded a rematch, but IBM refused and

Deep Blue, with its capability of evaluating 200 million positions per second, was the fastest computer to face a world chess champion. Today, in computer chess research and matches of world class players against computers, the focus of play has often shifted to software chess programs, rather than using dedicated chess hardware. Modern chess programs like Houdini, Rybka, Deep Fritz or Deep Junior are more efficient than the programs during Deep Blue's era. In a November 2006 match between Deep Fritz and world chess champion Vladimir Kramnik, the program ran on a computer system containing a dual-core Intel Xeon 5160 CPU, capable of evaluating only 8 million positions per second, but searching to an average depth of 17 to 18 plies in the middlegame thanks to heuristics; it won 4–2.[26][27]

One of the cultural im[...] of Deep Blue was the creation of a new game called Arimaa designed to be much more difficult for computers than che[...][22]

Origins [edit]

After Deep Blue…

After Deep Blue…

You lose, man - World chess champion falls to super computer

Boston Herald - Monday, May 12, 1997

Author: *Bill Hutchinson*

Would ***human*** *chess fade away?*

Watch out humans, the world will never be the same.

IBM's super-calculating computer Deep Blue made a statement for oppressed machines everywhere when it thundered to victory over mankind's greatest chess player, Garry Kasparov.

Deep Blue? Heck, call it Mr. Blue from now on.

In the New York City chess duel of Man vs. Machine, Deep Blue puzzled its human counterpart to a blood-boiling breakdown.

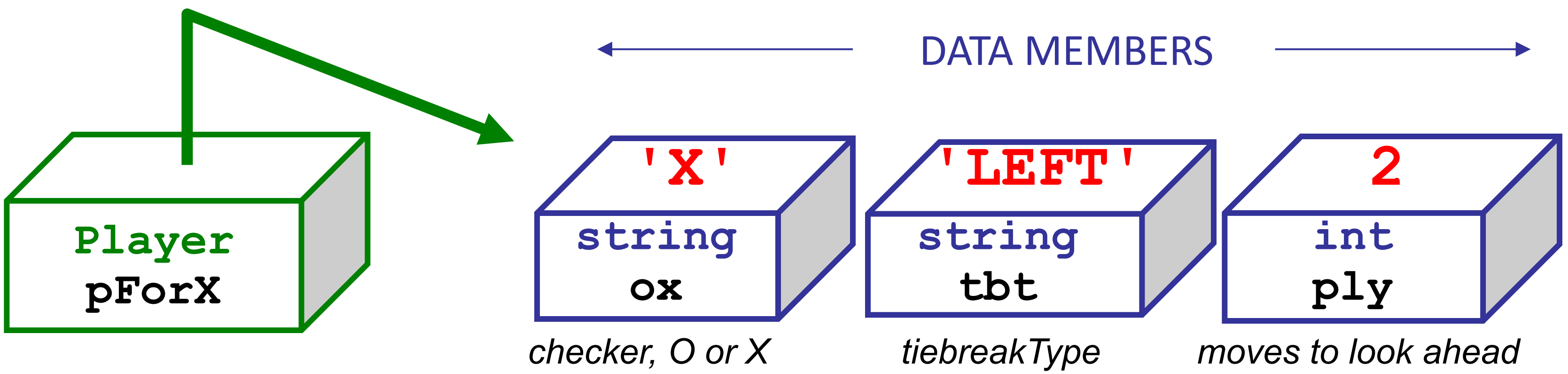
"I have to apologize for today's performance," the 34-year-old Russian Kasparov said after suffering the first chess defeat of his professional career. "I had no real energy to fight."

Deep Blue scored its 3 1/2 point to 2 1/2 point triumph in an astonishing 88-minutes. Kasparov shocked the chess world by resigning after only 19 moves with the black pieces.

The **`Player`** class

(EC for hw11 ~ by April 18)

What **data** does a computer AI player need?



ox? tbt? ply?

```
x = Player('X', 'LEFT', 42)
x0rn
o0rn
b.playGame( x0rn, o0rn )
```

... perhaps *surprisingly, not so much.*

Player's algorithms...

Board

__init__(self, width, height)

allowsMove(self, col)

addMove(self, col, ox)

delMove(self, col)

__repr__(self)

isFull(self)

winsFor(self, ox)

hostGame(self)

playGame(self, pForX, pForO)

Player

__init__(self, ox, tbt, ply)

__repr__(self)

oppCh(self)

scoreBoard(self, b)

scoresFor(self, b)

tiebreakMove(self, scores)

nextMove(self, b)

Demos?

So many ply!

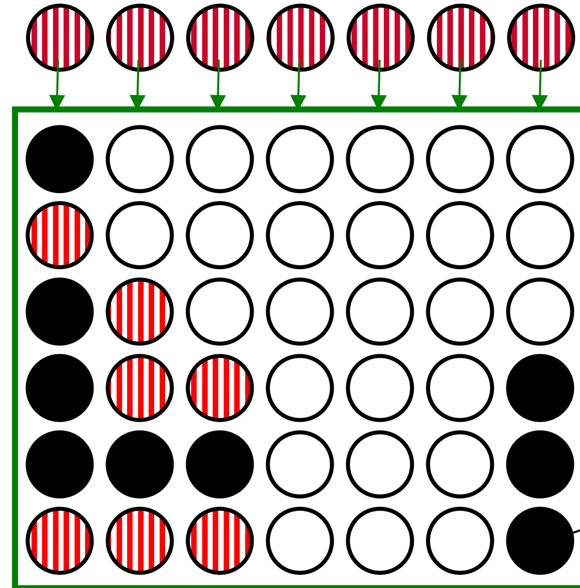
Quiz

Fill in the list of scores returned by **`scoresFor`**

The same move is evaluated at each ply... it's just evaluated farther into the future!

Each row is different in at least 1 score...

b



b42

'X'

'O'

you are playing 'O'

	col 0	col 1	col 2	col 3	col 4	col 5	col 6
`scoresFor(b)` ox == 'O' and **ply == 0**	-1	50	50	50	50	50	50
`scoresFor(b)` ox == 'O' and **ply == 1**	-1	50	50	100	50	50	50
`scoresFor(b)` ox == 'O' and **ply == 2**	-1	0	0	100	0	0	50
`scoresFor(b)` ox == 'O' and **ply == 3**	-1	0	0	100	0	0	100

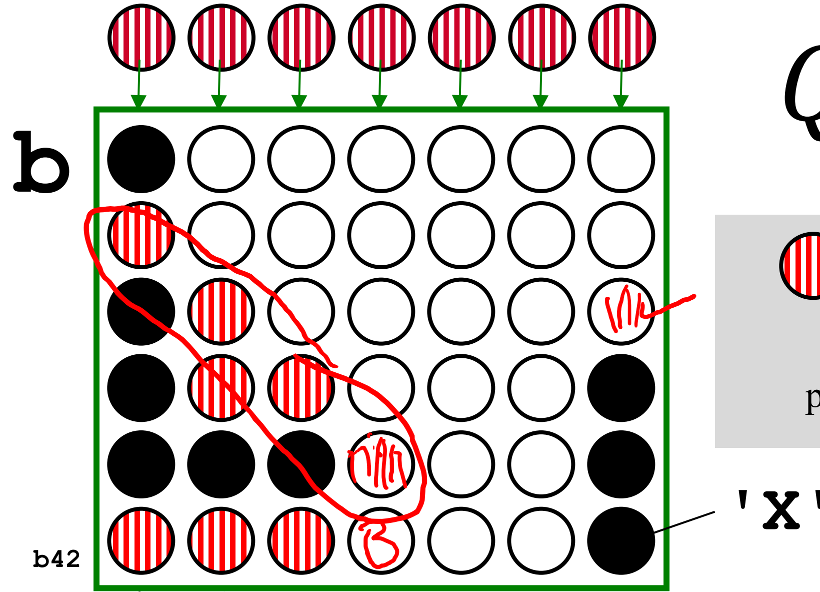
So many ply!

Quiz

Fill in the list of scores returned by **scoresFor**

The same move is evaluated at each ply... it's just evaluated farther into the future!

Each row is different in at least 1 score...



'O'

you are playing 'O'

	col 0	col 1	col 2	col 3	col 4	col 5	col 6
scoresFor(b) ox == 'O' and **ply == 0**	-1	50	50	50	50	50	50
scoresFor(b) ox == 'O' and **ply == 1**	-1	50	50	100	50	50	50
scoresFor(b) ox == 'O' and **ply == 2**	-1	0	0	100	0	0	50
scoresFor(b) ox == 'O' and **ply == 3**	-1	0	0	100	0	0	100

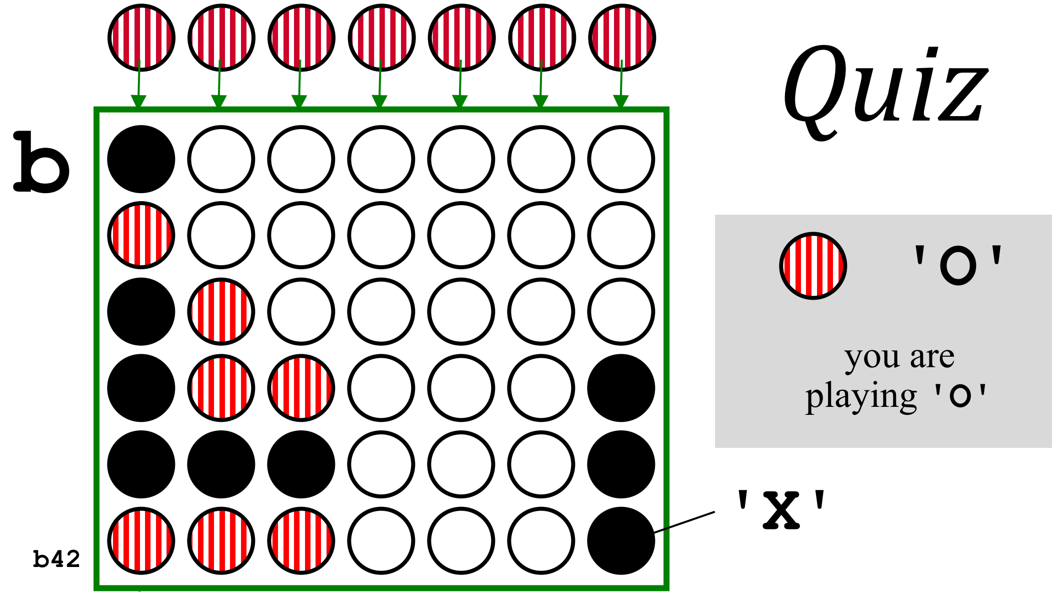
So many ply!

Quiz

Fill in the list of scores returned by **scoresFor**

The same move is evaluated at each ply... it's just evaluated farther into the future!

Each row is different in at least 1 score...



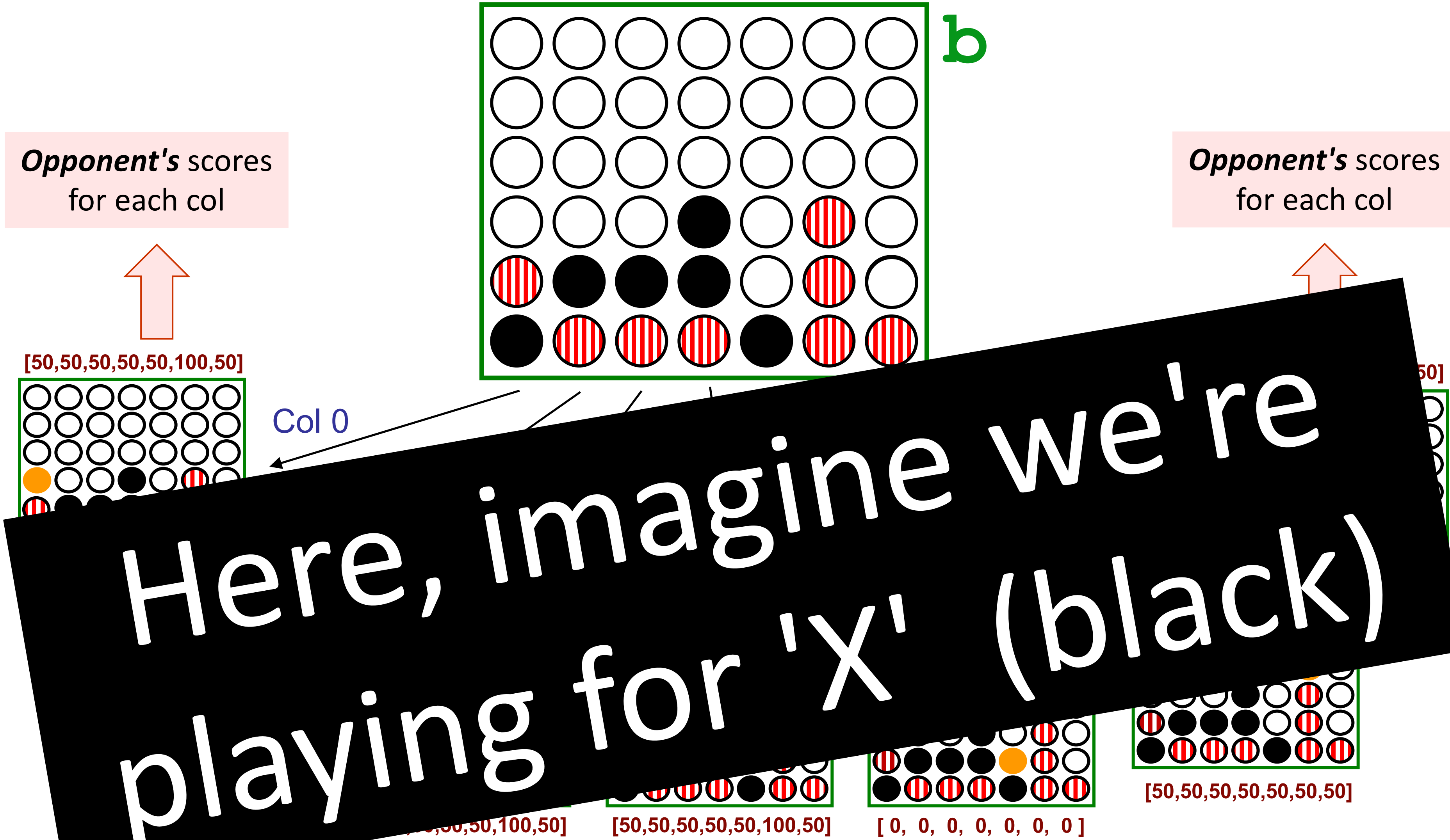
	col 0	col 1	col 2	col 3	col 4	col 5	col 6
scoresFor(b) ox == 'O' and **ply == 0**	-1	50	50	50	50	50	50
scoresFor(b) ox == 'O' and **ply == 1**	-1	50	50	100	50	50	50
scoresFor(b) ox == 'O' and **ply == 2**	-1	0	0	100	0	0	50
scoresFor(b) ox == 'O' and **ply == 3**	-1	0	0	100	0	0	?

scoresFor

Minimax!

(self) 'X'

new 'X'



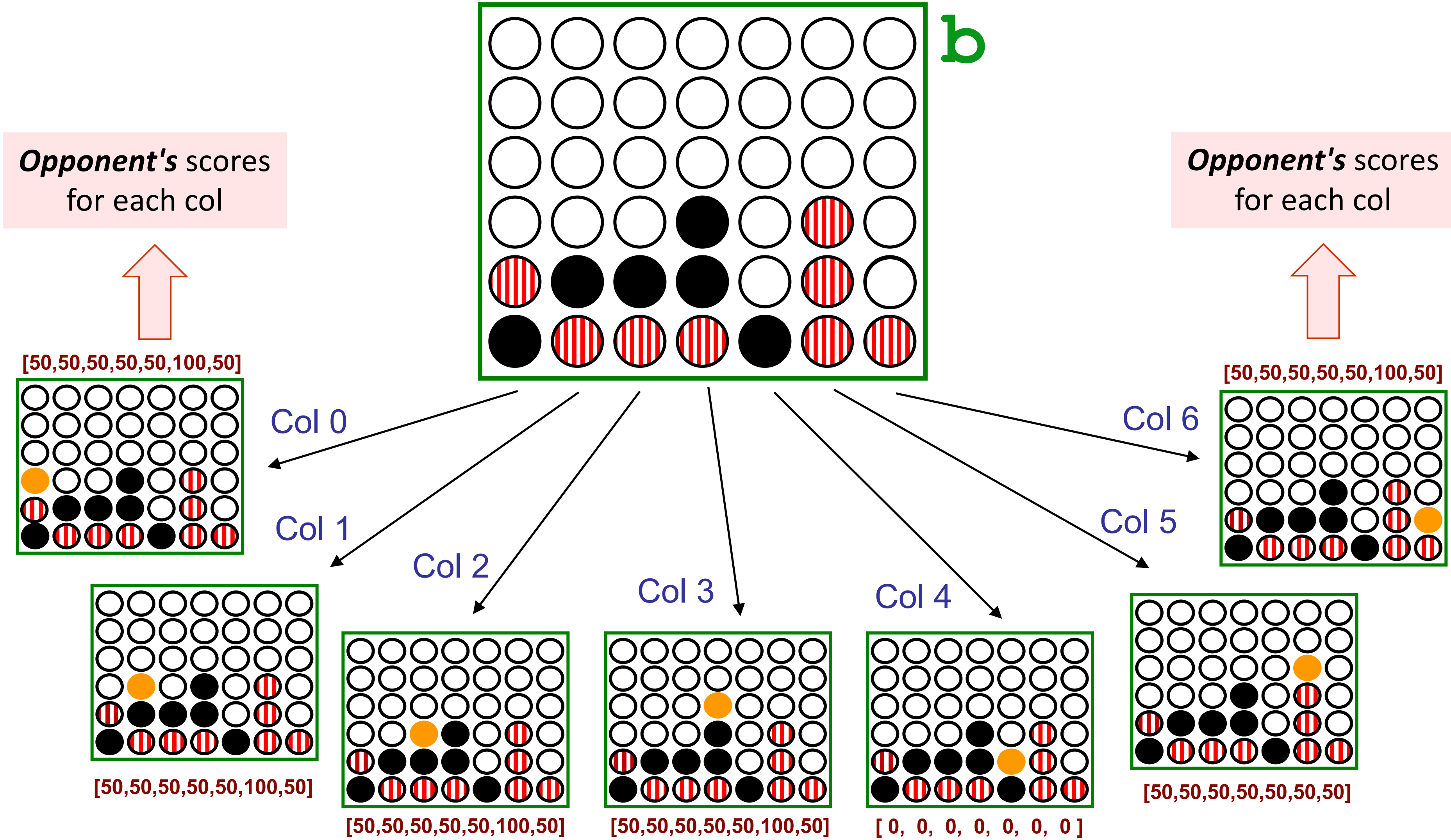
Here, imagine we're playing for 'X' (black)

scoresFor

Minimax!

(self) 'X' ●

new 'X' ●



scoresFor

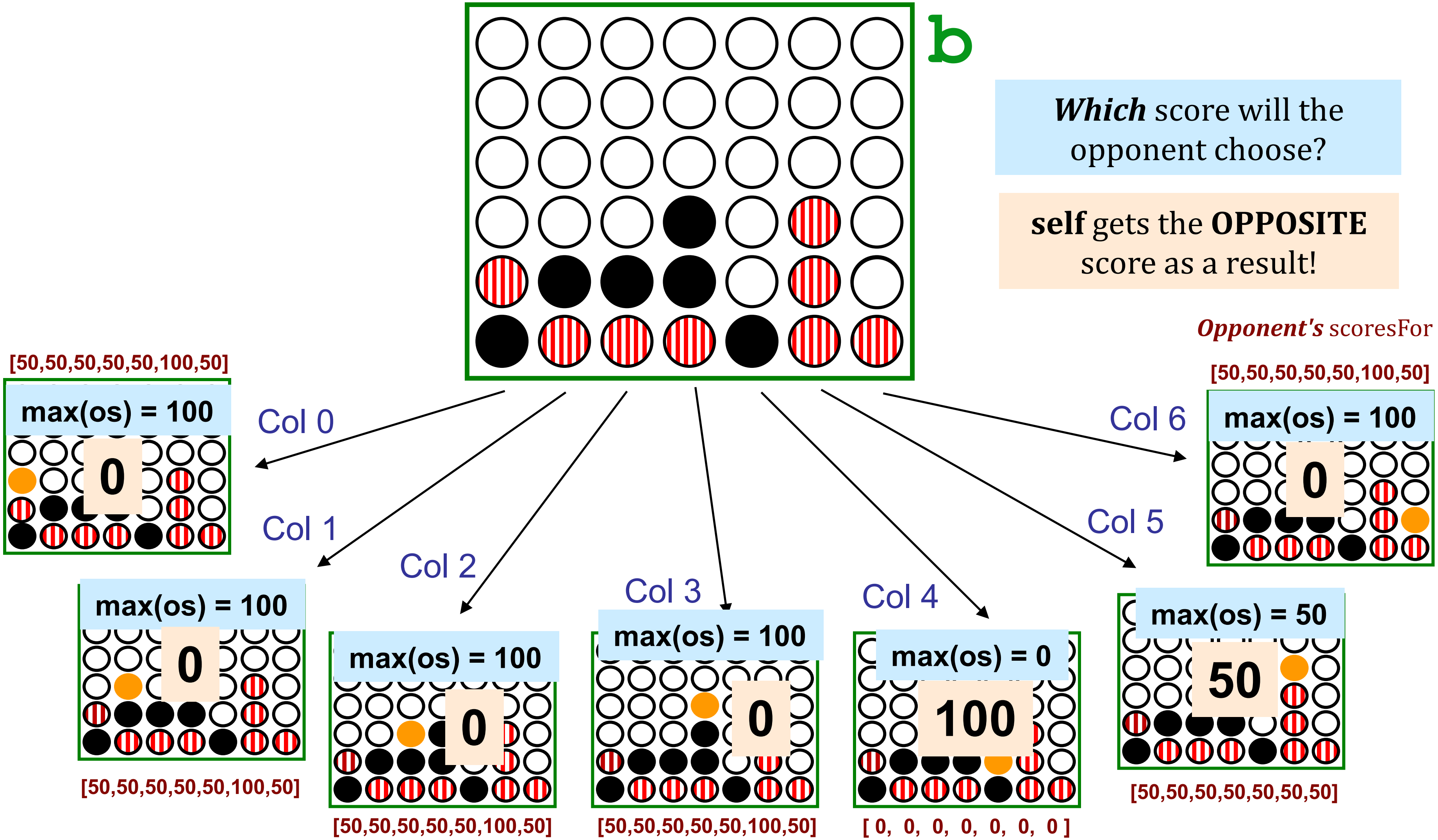
Minimax!

(self) 'X'

new 'X'

Which score will the opponent choose?

self gets the **OPPOSITE** score as a result!



scoresFor

(self) 'X'

new 'X'

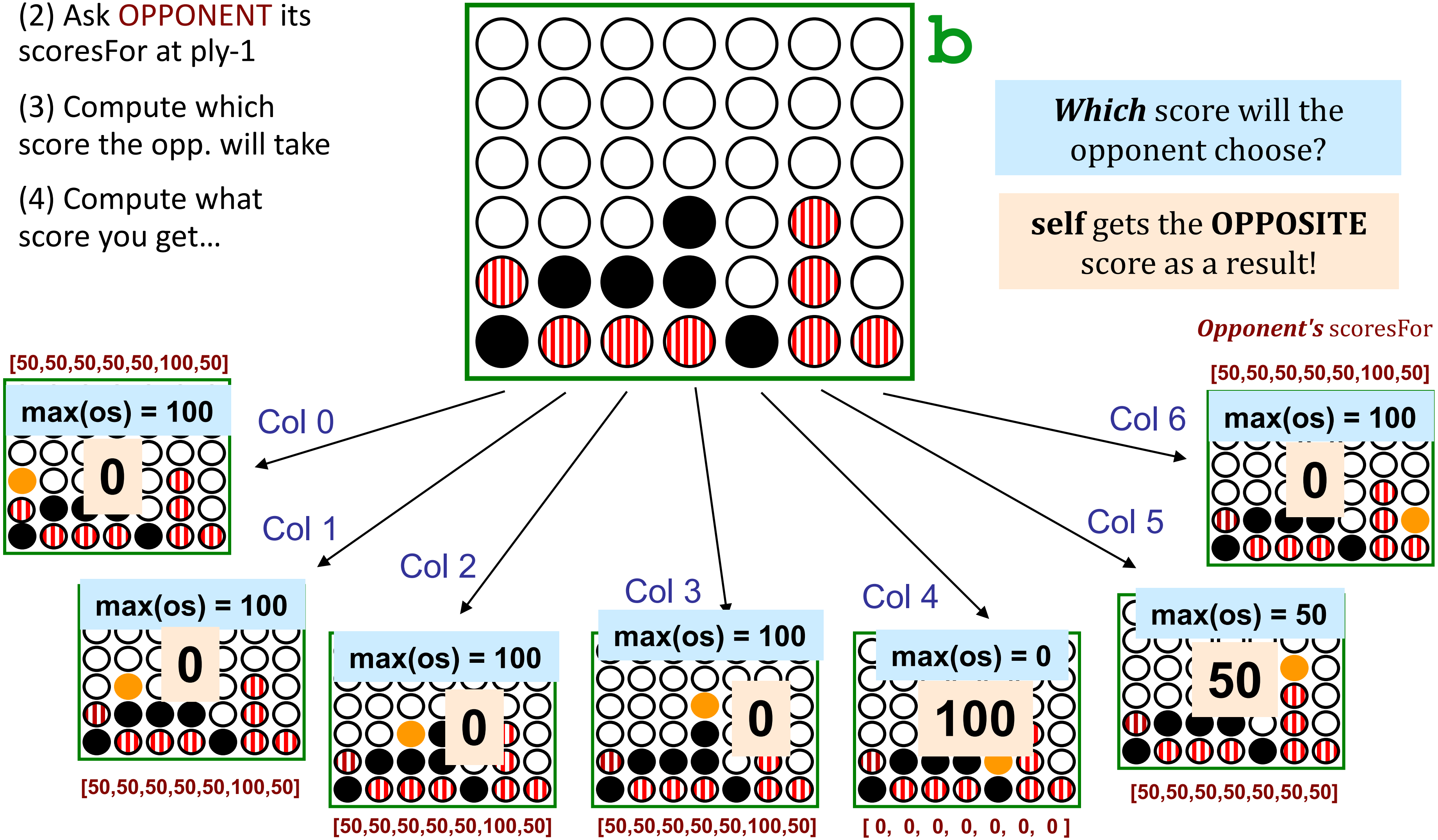
(0) Suppose you're playing at 2 ply...

(1) Make ALL moves!

(2) Ask OPPONENT its scoresFor at ply-1

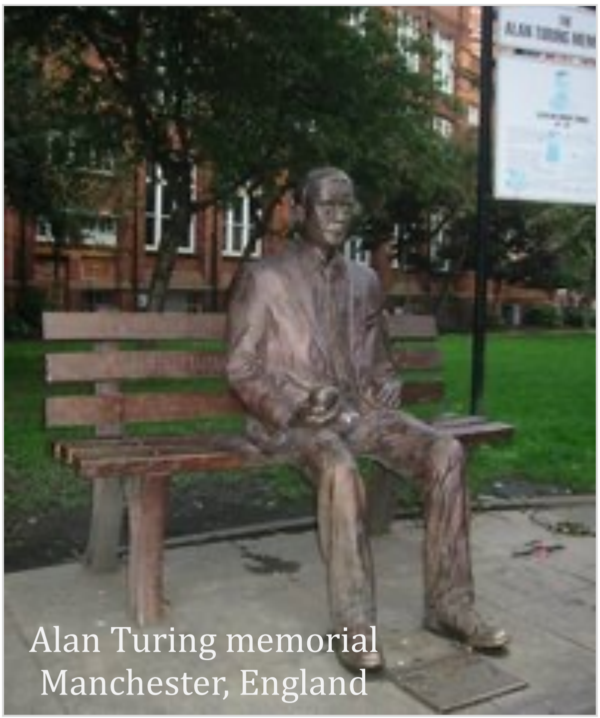
(3) Compute which score the opp. will take

(4) Compute what score you get...



Strategic thinking $\stackrel{?}{==}$ intelligence

Two-player games have been a key focus of AI
as long as computers have been around…



Alan Turing memorial
Manchester, England

In **1945**, Alan Turing predicted that computers would be better chess players than people in **~ 50 years…**

and thus would have achieved intelligence.

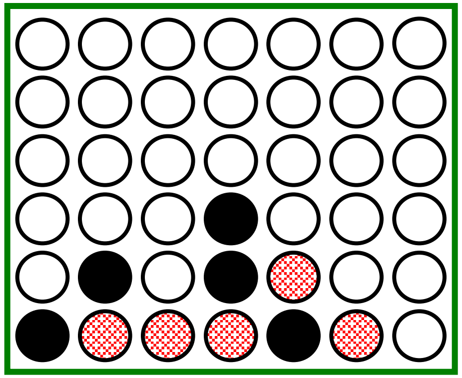
Remarkable **timing**!

and even more remarkable **premise**!!!

Strategic thinking != intelligence

computers

good at looking to find winning combinations of moves



humans

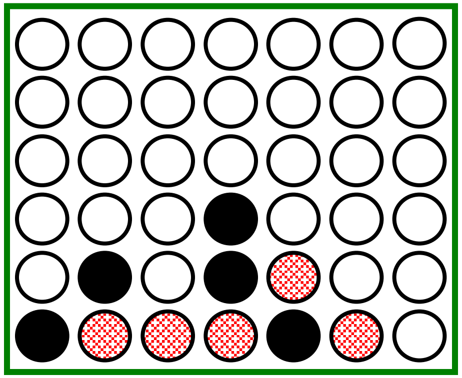
good at evaluating the strength of a board for a player

... humans and computers have different relative strengths in these games.

Humans play via "look-up table"

A. deGroot, a psychologist & chess player, experimented:
Chess-game positions were shown to chess **novices** and chess **experts** ... each for a couple of seconds...

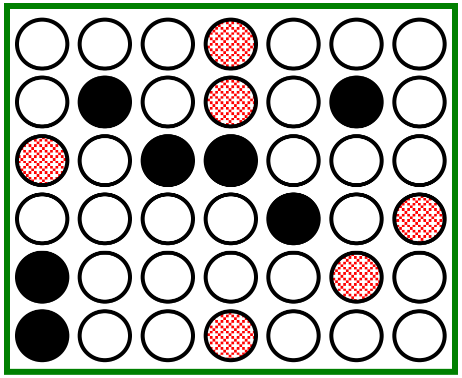
- experts reconstructed these (near) perfectly
- novice players did far worse...



Humans play via "look-up table"

A. deGroot, a psychologist & chess player, experimented:

Chess-game positions were shown to chess **novices** and chess **experts** ... each for a couple of seconds...



- experts reconstructed these (near) perfectly
- novice players did far worse...

Random chess-piece positions, not from a game, were also shown to the two groups:

- experts and novices did ***equally badly*** reconstructing them!

Connecting Connect Four ...

Connect 4

How complex are these games? *Least?* *Most?*

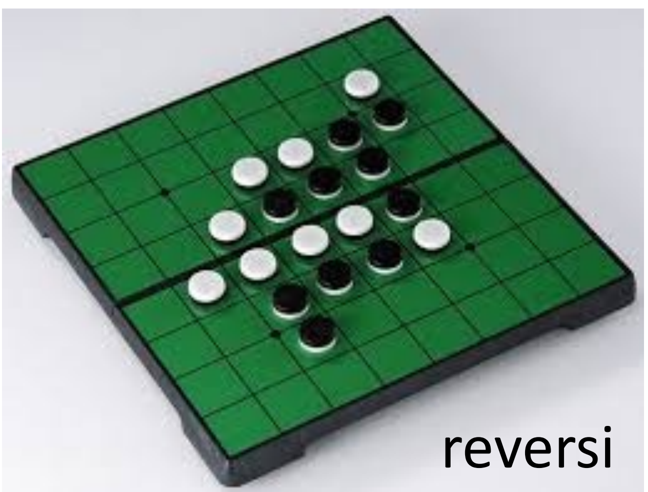
... to other strategy games.

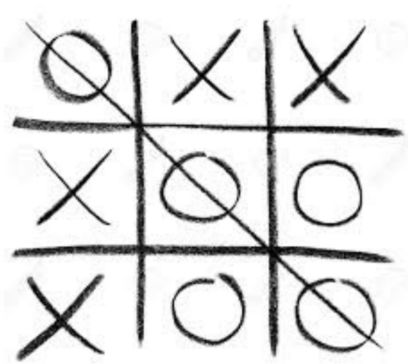
Connecting Connect Four ...


checkers

chess

Connect 4


reversi


tic-tac-toe


Go

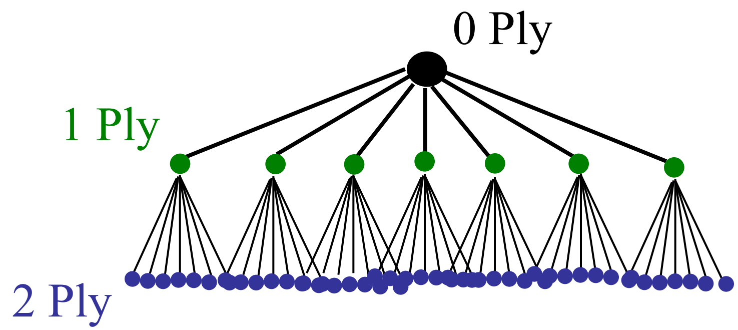
Rank these six games from least complex (1) to most complex (6)

... to other strategy games.

Games' Branching Factors

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

Chess players have more, **around 40 choices** per ply (on average, not every time)



Boundaries for ***qualitatively*** different games…

Branching Factors

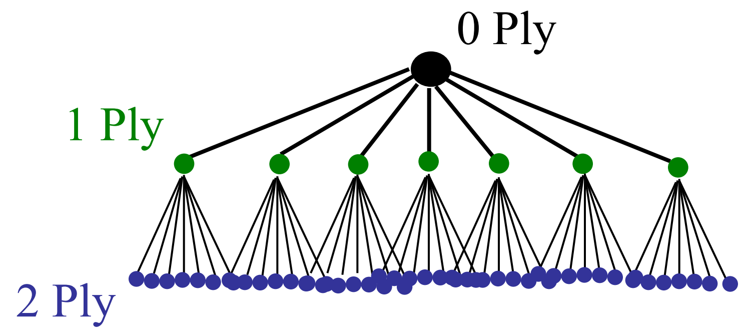
for different two-player games

"solved" games	Tic-tac-toe	4
	Connect Four	7
	Checkers	10
computer-dominated	Reversi	30
	Chess	40
human-dominated	Go	300
	Arimaa	17,000

Games' Branching Factors

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

Chess players have more, **around 40 choices** per ply (on average, not every time)



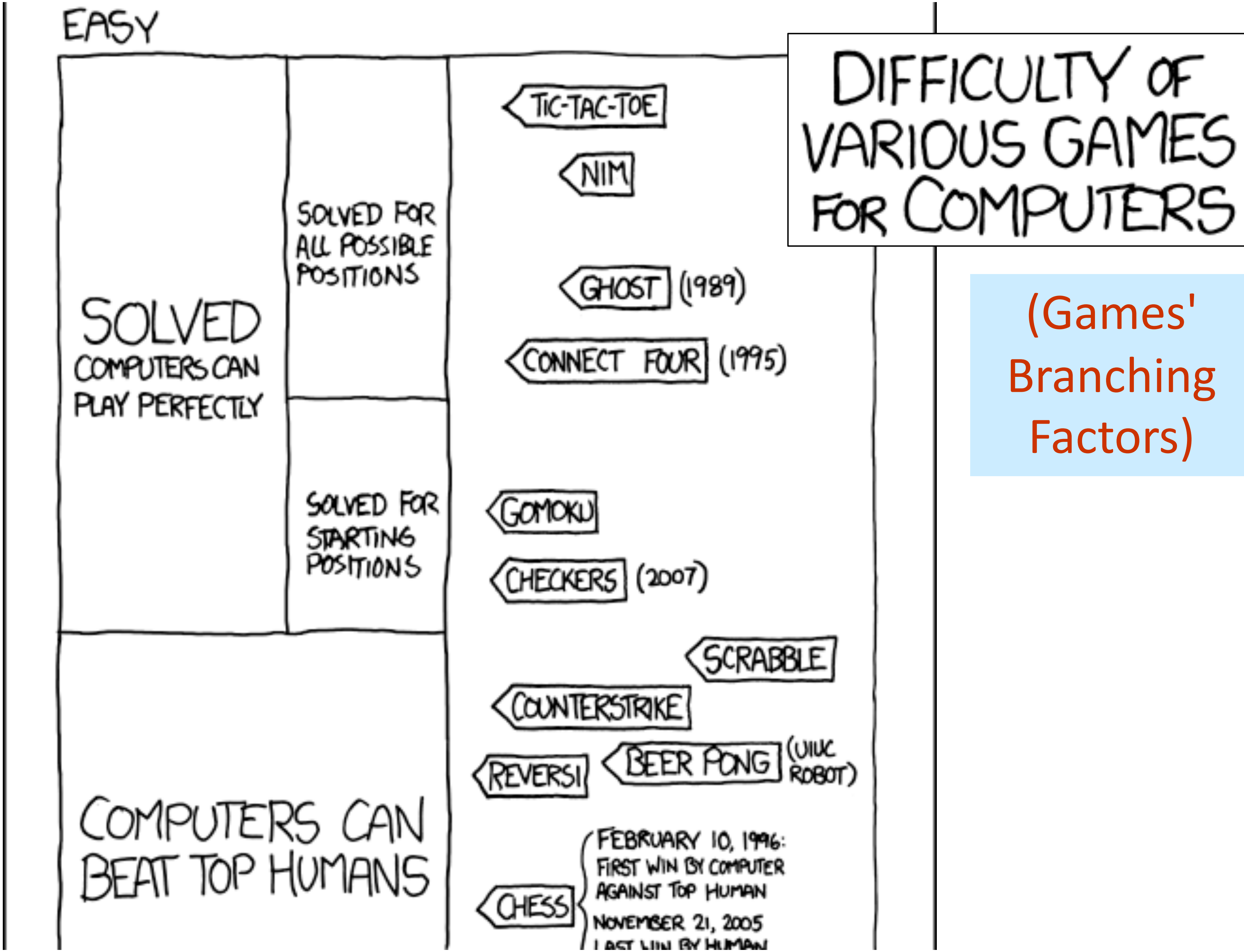
Boundaries for ***qualitatively*** different games…

Branching Factors

for different two-player games

	Game	Branching factor
"solved" games	Tic-tac-toe	4
	Connect Four	7
	Checkers	10
computer-dominated	Reversi	30
	Chess	40
human-dominated	Go	300
	Arimaa	17,000

only until 2016 → human-dominated



(Games' Branching Factors)

A Knowledge-based Approach of Connect-Four

The Game is Solved: White Wins

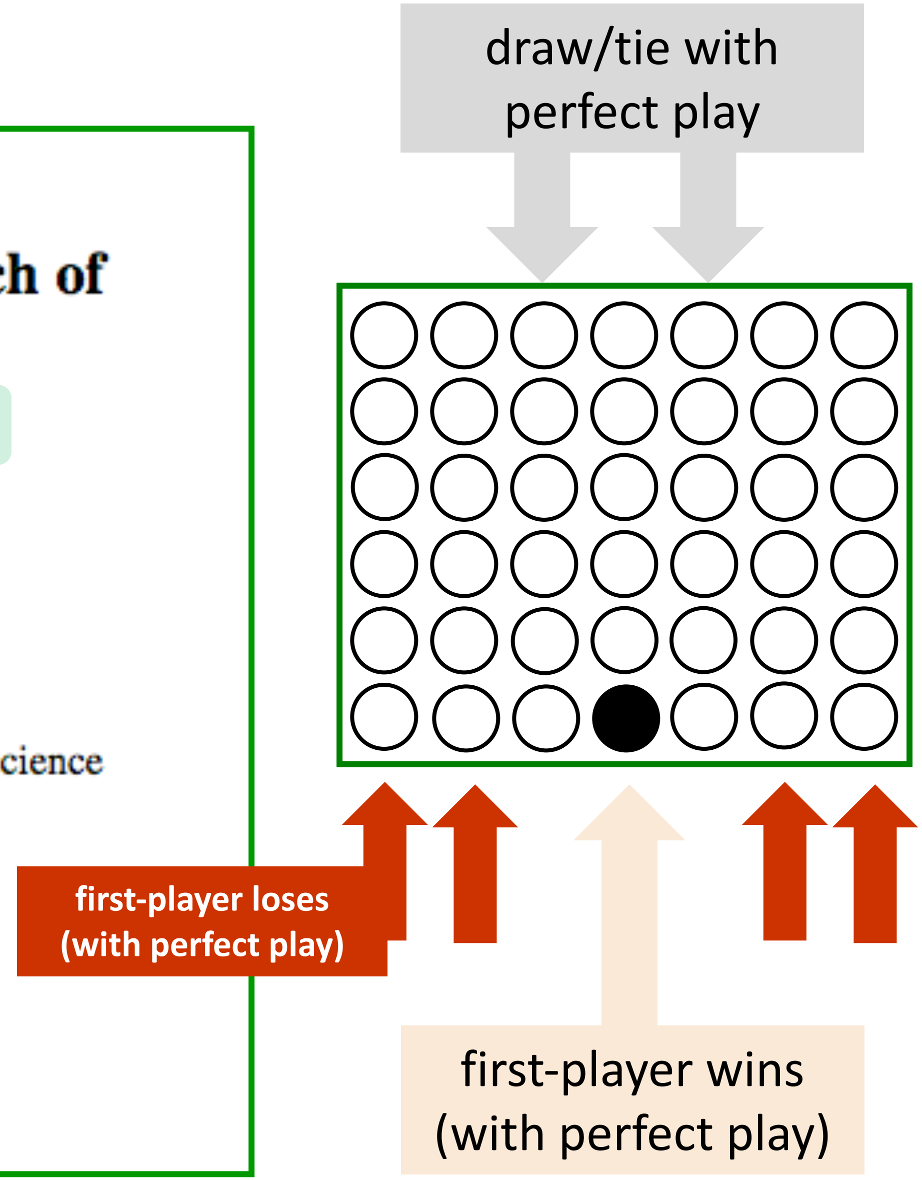
Victor Allis

Department of Mathematics and Computer Science

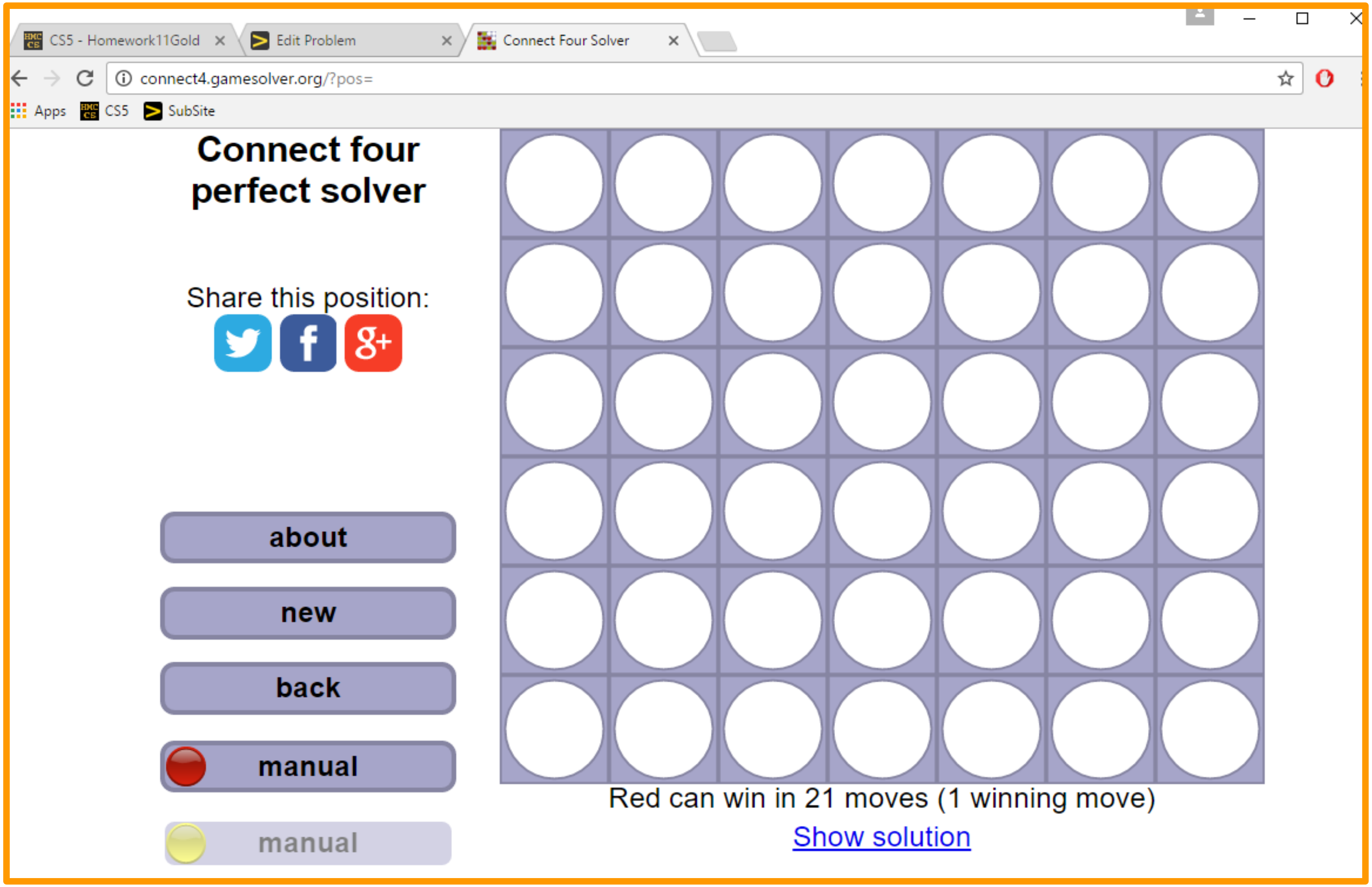
Vrije Universiteit

Amsterdam, The Netherlands

Masters Thesis, October 1988 †



Connect 4 was solved in **1988**.


CS5 - Homework11Gold
Edit Problem
Connect Four Solver
connect4.gamesolver.org/?pos=
Apps
CS5
SubSite
Connect four
perfect solver
Share this position:
about
new
back
manual
manual
Red can win in 21 moves (1 winning move)
Show solution

Science 14 September 2007:
Vol. 317. no. 5844, pp. 1518 - 1522
DOI: 10.1126/science.1144079

RESEARCH ARTICLES

Checkers Is Solved

Jonathan Schaeffer,* Neil Burch, Yngvi Björnsson,† Akihiro Kishimoto,‡ Martin Müller, Robert Lake, Paul Lu, Steve Sutphen

The game of checkers has roughly 500 billion billion possible positions (5×10^{20}). The task of solving the game, determining the final result in a game with no mistakes made by either player, is daunting. Since 1989, almost continuously, dozens of computers have been working on solving checkers, applying state-of-the-art artificial intelligence techniques to the proving process. This paper announces that checkers is now solved: Perfect play by both sides leads to a draw. This is the most challenging popular game to be solved to date, roughly one million times as complex as Connect Four. Artificial intelligence technology has been used to generate strong heuristic-based game-playing programs, such as Deep Blue for chess. Solving a game takes this to the next level by replacing the heuristics with perfection.

Checkers was solved in **2007**.