Intelligent CS 5?

X to move.

Is there a way to ensure a win?

If so, how far ahead?

|X|X|X|X|X|O|O|
|X|X|X|X|X|X|X|
|X|X|X|X|X|O|O|

Who won?!

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

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

C4 AI ~ how *could* it work?

while True:

```python
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!

C4 AI ~ how *could* it work?

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!

Hw11 due Monday @ 11:59pm
Office hours Thursday: 2-4

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|

Who won?!

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

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

C4 AI ~ how *could* it work?

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!

C4 AI ~ how *could* it work?

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!

Who won?!
C4 AI ~ how **should** it work?

I feel ahead of the game here...

| | | | | | | |
| | | | | | | |
| | | | | | | |
| | |X| | | | |
| |X| |X| | | |
|X|O|O|O| |O| |

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

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

C4 AI ~ "intuitive" moves?

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

C4 AI ~ **lookahead** moves...

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

| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | |X| | | |
| |X| |X| | | |
|X|O|O|X|X|O| |

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

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

---

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 Gary Kasparov by two wins to one with three draws. Kasparov accused IBM of cheating and demanded a rematch, but IBM refused and dismantled Deep Blue. Kasparov had beaten a previous version of Deep Blue in 1996.

Contents [hide]
1 Origins
2 Deep Blue versus Kasparov
3 Affirmative
4 See also
5 Notes
6 References
7 Further reading
8 External links

Origins
[edit]
Plies ~ "turns to checkmate"  
(for any game)

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

How many moves ahead might we have to look?

Ply 0:

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Ply 1:

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Ply 2:

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Ply 3:

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

After Deep Blue...

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

Boston Herald - Monday, May 12, 1997
Author: Bill Hutchinson

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.
Connect 4, Part 2

hw11pr2.py

1 ply + 2 ply

what methods will help?

colsToWin( self, ox )

b.colsToWin('O')
b.colsToWin('X')

intuition-based
tiebreaking

what methods will help?

aiMove( self, ox )

b.aiMove('O')
b.aiMove('X')

hostGame( self )

C4 AI

Looking further ahead...?
Looking further ahead... ?

Final-project timeline:

All use objects + classes
All use nested data: 2d or more
All are open-ended somehow...

Start (2-3 functions): Mon. 4/22
Milestone: Mon. 4/29
Final Project: Fri. 5/3

for example...

The Player class

What data does a computer AI player need?

Player pForX

DATA MEMBERS

'X' string ox
'LEFT' string tbt
2 int ply

tiebreakType moves to look ahead

ox? tbt? ply?

Looking further ahead... !

How could we write a 3-ply lookahead?
What about 4-ply? N-ply?

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

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

... perhaps surprisingly, not so much.
Player's algorithms...

Board

__init__(self, ox, tbt, ply)
__init__(self, ox, tbt, ply)
__repr__(self)
oppCh(self)
scoreBoard(self, b)
scoresFor(self, b)
tiebreakMove(self, scores)
nextMove(self, b)

Player

__init__(self, ox, tbt, ply)
__repr__(self)

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

scoreBoard(self, b)

Returns a score for any board, b

A simple system:

<table>
<thead>
<tr>
<th></th>
<th>100.0 for a win</th>
<th>50.0 for anything else</th>
<th>0.0 for a loss</th>
</tr>
</thead>
</table>

Score for •
Score for o

scoresFor at 0 ply...

What should scoresFor return for • with ply == 0

0- ply means 0 moves are made!

0 ply is a Zen-like approach: *exist only in the present*

We still use -1 as the score into a full column.

0 ply

0- ply scores for •
**scoresFor at 1 ply...**

What should `scoresFor` return for 'O' with `ply == 1`?

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

```
1-ply means 1 move is made!
```

```
O1.scoresFor( b2 )
```

"Gotcha!"

I try 1 ply!

**scoresFor at 2 ply for 'O'**

What should `scoresFor` return for 'O' with `ply == 2`?

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

```
2-ply means 2 moves are made!
```

```
O2.scoresFor( b2 )
```

"Gotcha!" + "Uh Oh..."

**scoresFor at 2 ply for 'O'**

What should `scoresFor` return for 'O' with `ply == 2`?

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

```
2-ply means 2 moves are made!
```

```
O2.scoresFor( b2 )
```

"Gotcha!" + "Uh Oh..."

**Try it!**

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...
```

```
scoresFor(b)
ox == 'O' and ply == 0
```

```
col 0 | col 1 | col 2 | col 3 | col 4 | col 5 | col 6
------|------|------|------|------|------|------
   |   |   |   |   |   |   
```

```
scoresFor(b)
ox == 'O' and ply == 1
```

```
col 0 | col 1 | col 2 | col 3 | col 4 | col 5 | col 6
------|------|------|------|------|------|------
   |   |   |   |   |   |   
```

```
scoresFor(b)
ox == 'O' and ply == 2
```

```
col 0 | col 1 | col 2 | col 3 | col 4 | col 5 | col 6
------|------|------|------|------|------|------
   |   |   |   |   |   |   
```

```
scoresFor(b)
ox == 'O' and ply == 3
```

```
col 0 | col 1 | col 2 | col 3 | col 4 | col 5 | col 6
------|------|------|------|------|------|------
   |   |   |   |   |   |   
```
**Idea: scoresFor**

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

In 1945, Alan Turing predicted that computers would be better chess players than people in ~ 50 years... and thus would have achieved intelligence.

**Humans play via "look-up table"**

An experiment (by A. deGroot) was performed in which chess positions were shown to novice and expert players for a few seconds...

- experts could reconstruct these perfectly
- novice players did far worse...

**Connecting Connect Four ...**

How complex are these games? Least? Most? ... to other strategy games.
Games' Branching Factors

On average, Connect 4 players have **seven choices** per move. Chess players have more, perhaps **around 40**, possible choices in a given move.

**Boundaries for qualitatively different games...**

<table>
<thead>
<tr>
<th>Game</th>
<th>Branching Factors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tic-tac-toe</td>
<td>4</td>
</tr>
<tr>
<td>Connect Four</td>
<td>7</td>
</tr>
<tr>
<td>Checkers</td>
<td>10</td>
</tr>
<tr>
<td>Reversi</td>
<td>30</td>
</tr>
<tr>
<td>Chess</td>
<td>40</td>
</tr>
<tr>
<td>Go</td>
<td>300</td>
</tr>
<tr>
<td>Arimaa</td>
<td>17,000</td>
</tr>
</tbody>
</table>

**only until 2016**

- **Connect 4 was solved in 1988.**
- **Checkers was solved in 2007.**

---

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.
Games' complexity ~ xkcd

3d graphics-based game using VPython

Go, vPool!

The vPool project

- Linear collisions should be somewhere ("walls")
- Spherical collisions should be somewhere ("points")
- You need "pockets" – or some other game objective
- You need user control of at least one object (mouse/kbd)

To now, VPython has eventually worked for everyone. See us for help!

Phunky Physics is welcome!
Spherical collisions

Zeroth approximation:
Stop \textbf{\textit{q}}. Undo any overlap.
Make \( r \).vel = \( q \).vel.

First approximation:
Stop \textbf{\textit{q}}. Undo any overlap.
Compute \( d = r \).pos – \( q \).pos
Make \( r \).vel = \( d \)

Second approximation:
Same as first, but
Make \( q \).vel = \( d \perp \), at 90° from \( d \)

vPool – physics?

Reality is just three eyes away!

Reality is just two eyes away!

Reality is just one eye away!

equations below...
The vPool project

• **Linear collisions** should be somewhere ("walls")
• **Spherical collisions** should be somewhere ("points")
• You need "pockets" – **or some other game objective**
• You need **user control** of at least one object (mouse/kbd)

---

VPython was designed to make 3d physics simulations simpler to program – as a result, the library is physics-free!

⇒ Phunky Physics is welcome!

To now, VPython has *eventually* worked for everyone. *See us for help!*

---

**Enjoy the projects!**

... *the graders certainly do!*

---

A few examples to get you thinking...

---

An unusual variation on VPool

---
Plying our intuitions...

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

Example

|X|0|0|
|X|X|X|
|0|X|0|
|0|0|X|
|X|X|0|
|X|X|X|

ply == 0

|0|0|0|
|X|X|X|
|0|0|0|
|0|0|X|
|X|X|0|
|X|X|X|

ply == 1

|0|X|
|X|X|
|0|0|
|0|0|
|X|O|
|X|

ply == 2

|O|O|O|
|X|X|
|O|O|
|O|O|
|X|O|
|X|

ply == 3

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

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

Example

|X|O|O|
|X|X|X|
|O|X|O|
|O|X|X|
|O|O|O|
|X|X|

ply == 0

|X|O|O|
|X|X|X|
|O|X|O|
|O|X|X|
|O|O|O|
|X|X|

ply == 1

|X|O|
|X|
|0|0|
|0|
|X|
|X|

ply == 2

|X|O|
|X|
|0|0|
|0|
|X|
|X|

ply == 3

Challenge: What about 2-ply each?