X to move.

Is there a way to ensure a win?

If so, how far ahead?

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

Who won?!

Oh, I 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?

Let’s try!

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

```
if ox == '0': ox = 'X'
else: ox = '0'
```

# check if game is over!

Who won?!

Oh, I won!

It could just play randomly...

Or, it could always play as far left 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 | X |   |   |   |   |   |
| O | O | O |   |   |   |   |
| X | X |   |   |   |   |   |
| O | O | O |   |   |   |   |
| X | X | X |   |   |   |   |

---------------

0 1 2 3 4 5 6

Oh, I 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 **should** it work?

I feel ahead of the game here...

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

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

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 ~ "intuitive" moves?

We'll run a C4 tournament with all of the aiMoves submitted...

- (ex. cr.) better than random? +5
- also, based on round-robin results...

If there isn't a win or loss... where should you go? Why?
C4 AI ~ **lookahead** moves...

I feel ahead of the game here...

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

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)

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.

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

Origins [edit]
Deep Blue (chess computer)

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 better a

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 impacts of Deep Blue was the creation of a new game called Arimaa designed to be much more difficult for computers than chess.[22]
Deep Blue (chess computer)

Deep Blue was a chess-playing computer developed by IBM. On May 11, 1997, the machine...

Arimaa - Intuitively simple ... intellectually challenging
GAMES Magazine: 2011 Best Abstract Strategy Game. This deep and groundbreaking game is not new, but we praise Z-Man for launching a thematic set to ...

Arimaa Game Rules
arimaa.com/arimaa/learn/rulesIntro.html
Arimaa is designed so that it can easily be played using a standard chess set. To make the game easier to learn for someone who is not familiar with chess, the ...

Arimaa - Wikipedia
https://en.wikipedia.org/wiki/Arimaa
Arimaa is a two-player strategy board game that was designed to be playable with a standard chess set and difficult for computers while still being easy to learn and fun to play for humans. Wikipedia

Arimaa | Board Game | BoardGameGeek
https://boardgamegeek.com/boardgame/4616/arimaa
Arimaa, pronounced "a-ree-muh," is a game where stronger animals like elephants and camels freeze, push, and pull the weaker ones from the opposing team ...

Arimaa Free Strategy Game Software - Smart Games
www.smart-games.com/arimaa.html

One of the cultural impacts of Deep Blue was the creation of a new game called Arimaa designed to be much more difficult for computers than chess.}[22]
Plies ~ "turns to checkmate" (for any game)

How many moves ahead might we have to look?

b0

x5.scoresFor( b0 )
Plying our intuitions...

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

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

**Example**

<table>
<thead>
<tr>
<th>X</th>
<th>O</th>
<th>O</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>O</td>
<td>O</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>O</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>O</td>
<td>O</td>
<td>O</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

**Challenge:** What about 2-ply each?
Ply our intuitions...

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

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

Example

X: ply == 0,  
O: ply == 0,  
both: tbt == 'LEFT'

b.playGame( x0, o0 )

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

Challenge: What about 2-ply each?

Let's try it!!
Plying our intuitions...

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

Try this on the back page first...

Pass these forward...

X: ply == 0,
O: ply == 0,
both: tbt == 'LEFT'

b.playGame( x0, o0 )

In all 4 of these boards, X will move to col 3, even if both players tiebreak to the LEFT.
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.
The World Chess Championship Match 2016, held from 11 to 30 November, will be contested by 25 year old reigning champion Magnus Carlsen of Norway and his challenger, 26 year old Sergey Karjakin of Russia — and this is the first time that two players who have come of age in the computer era are fighting for the title and represent a generational shift in chess.
But, in practice...

In Norway, Chess Broadcast Spurs NFL-Like Fan Frenzy

THE WALL STREET JOURNAL.
ESSAY
Chess-Championship Results Show Powerful Role of Computers
The digital revolution has pushed human abilities to new heights

Computers have gone so far that the top human players are now those who most often play the moves that would be chosen by the best engines (which sport names like Houdini, Hiarcs and Rybka). Magnus Carlsen's biographers dub him the "hero of the computer era." Indeed, a study published on ChessBase.com earlier this year showed that in the tournament Mr. Carlsen won to qualify for the world championship match, he played more like a computer than any of his opponents.
still popular!

Humans adapt!

The Championship Results Show Powerful Role of

ChessBase.com earlier this year showed
 qualify for the world championship match, he
 opponents.
Connect 4, Part 2

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 )
Looking further ahead...?

More ply!

4th Final Project Option...
VPool

Looking further ahead... ?

C4 AI

4 Final Project Options

VPython

TextID

HB Ron Hagrid

Markov

Picobot!

CFour

C4	AI
Final-project timeline:

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

Progress + Dates:

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?

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

... perhaps *surprisingly, not so much.*
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?
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?
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 \textit{arithmetize} an activity.

- paraphrasing Alan Perlis
scoreBoard(self, b)

Returns a score for any board, b

A simple system:

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

Score for ● 100
Score for ● 50
Score for ○ 0
scoressFor at 0 ply...

What should scoressFor return for \( \bullet \) with \( \text{ply} == 0 \)?

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

0- ply scores for \( \bullet \):

\[-1 \ 50 \ 50 \ 50 \ 50 \ 50 \]

0- ply means 0 moves are made!

to move

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

\[ o0.\text{scoressFor}(b2) \]
I try 1 ply!

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

What should `scoresFor` return for \( \bullet \) with `ply == 1`?

"Gotcha!"

1-ply scores for:

```
- 50 50 50 100 100 50
```

1-ply means 1 move is made!

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

To move

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

\( o1.scoresFor( b2 ) \)
What about 3-ply?

What should `scoresFor` return for a 2-ply lookahead to win or block the opponent's win?

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

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

2- and 3-ply scores for...
A 2-ply lookahead player will see a way to win or block the opponent's win. "Gotcha!" + "Uh Oh..."
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...

---

**Try it!**

---

So many ply!

---

```
Try it!

'X'

you are playing 'O'

---

Fill in the list of scores returned by `scoresFor`

---

Try it!

```

---

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

So many ply!

```
Try it!
```

Try it! you are playing 'O'

<table>
<thead>
<tr>
<th></th>
<th>col 0</th>
<th>col 1</th>
<th>col 2</th>
<th>col 3</th>
<th>col 4</th>
<th>col 5</th>
<th>col 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>scoresFor(b)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ox == 'O' and ply == 0</td>
<td>-1</td>
<td>50</td>
<td>50</td>
<td>50</td>
<td>50</td>
<td>50</td>
<td>50</td>
</tr>
<tr>
<td>scoresFor(b)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ox == 'O' and ply == 1</td>
<td>-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>scoresFor(b)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ox == 'O' and ply == 2</td>
<td>-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>scoresFor(b)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ox == 'O' and ply == 3</td>
<td>-1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
So many ply!

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...

```
<table>
<thead>
<tr>
<th>ox == 'O' and ply == 0</th>
<th>col 0</th>
<th>col 1</th>
<th>col 2</th>
<th>col 3</th>
<th>col 4</th>
<th>col 5</th>
<th>col 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>50</td>
<td>50</td>
<td>50</td>
<td>50</td>
<td>50</td>
<td>50</td>
<td>50</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>ox == 'O' and ply == 1</th>
<th>col 0</th>
<th>col 1</th>
<th>col 2</th>
<th>col 3</th>
<th>col 4</th>
<th>col 5</th>
<th>col 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>50</td>
<td>50</td>
<td>100</td>
<td>50</td>
<td>50</td>
<td>50</td>
<td>50</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>ox == 'O' and ply == 2</th>
<th>col 0</th>
<th>col 1</th>
<th>col 2</th>
<th>col 3</th>
<th>col 4</th>
<th>col 5</th>
<th>col 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>100</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>50</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>ox == 'O' and ply == 3</th>
<th>col 0</th>
<th>col 1</th>
<th>col 2</th>
<th>col 3</th>
<th>col 4</th>
<th>col 5</th>
<th>col 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>100</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>b0</td>
</tr>
</tbody>
</table>
```

Quiz

- 'O' 'X'

you are playing 'O'

So many ply!
**Idea:** `scoresFor`

```
Opponent's scoresFor

[50,50,50,50,50,100,50]
Col 0

[50,50,50,50,100,50]
Col 1

Col 2

Col 3

Col 4

Col 5

Col 6

[50,50,50,50,50,50,100,50]
```
Idea: \textit{scoresFor}

Which score will the opponent choose?

\textbf{self} gets the\textbf{ OPPOSITE} score as a result!

\textbf{Opponent's} \textit{scoresFor}

\textbf{max(os) = 100}

\textbf{self} \text{ 'X'}

\textbf{new \text{ '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…

Which score will the opponent choose?

self gets the OPPOSITE score as a result!
Strategic thinking $\equiv$ intelligence

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.
Strategic thinking ≠ intelligence

...humans and computers have different relative strengths in these games.
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...
Humans play via "look-up"

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...

Random chess positions (not legal ones) were then shown to the two groups

- experts and novices did equally badly at reconstructing them!

Adriaan de Groot
Connecting Connect Four ...

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

... to other strategy games.
Connecting Connect Four ...

checkers  
chess  
reversi  
tic-tac-toe  
Go  

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...

Branching Factors for different two-player games

<table>
<thead>
<tr>
<th>Game</th>
<th>Branching Factor</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>

“solved” games

computer-dominated

human-dominated
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...**

**Branching Factors** for different two-player games

<table>
<thead>
<tr>
<th>Game</th>
<th>Branching Factor</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>

“solved” games

**computer-dominated**

only until 2016

**human-dominated**
Connect 4 was solved in 1988.

**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 †

draw/tie with perfect play

first-player loses (with perfect play)

first-player wins (with perfect play)
Checkers was solved in 2007.
DIFFICULTY OF VARIOUS GAMES FOR COMPUTERS

SOLVED
COMPUTERS CAN PLAY PERFECTLY

SOLVED FOR ALL POSSIBLE POSITIONS

EASY

SOLVED FOR STARTING POSITIONS

(Games' Branching Factors)

COMPUTERS CAN BEAT TOP HUMANS

TIC-TAC-TOE

NIM

GHOST (1989)

CONNECT FOUR (1995)

GOMOKU

CHECKERS (2007)

SCRABBLE

COUNTERSTRIKE

REVERSi

BEER PONG (UIUC ROBOT)

CHESS

FEBRUARY 10, 1996:
FIRST WIN BY COMPUTER AGAINST TOP HUMAN

NOVEMBER 21, 2005:
1 ACT WIN BY HUMAN
COMPUTERS CAN BEAT TOP HUMANS

COMPUTERS STILL LOSE TO TOP HUMANS
(BUT FOCUSED R&D COULD CHANGE THIS)

COMPUTERS MAY NEVER OUTPLAY HUMANS

DIFFICULTY OF VARIOUS GAMES FOR COMPUTERS

unlikely
How Google's AlphaGo Beat a Go World Champion
Mastering the game of Go without human knowledge


doi:10.1038/nature24270
Received: 07 April 2017
Accepted: 13 September 2017
Published online: 18 October 2017

How Google's AlphaGo Beat a Go World Champion

unlikely
IT'S SATURDAY! WHAT DO YOU WANT TO DO?
ANYTHING BUT PLAY AN ORGANIZED SPORT.
WANT TO PLAY CALVINBALL?
YEAH!
NO SPORT IS LESS ORGANIZED THAN CALVINBALL!
NEW RULE! NEW RULE.
IF YOU DON'T TOUCH THE 30-YARD BASE WICKET WITH THE FLAG, YOU HAVE TO HOP ON ONE FOOT.

I DIDN'T SEE YOU TOUCH THE OPPOSITE POLE! YOU HAVE TO DECLARE IT!
I DECLARED IT OPPOSITELY BY NOT DECLARING IT. START SINGING.

DIFFICULTY OF VARIOUS GAMES FOR COMPUTERS
Go, vPool!

Let's play!

I'll take your cue.

A few constraints...

- need ≥4 physically interacting objects
- allow the user to direct 1+ objects, either by keyboard or mouse or both
- needs a game goal + be winnable!
- must detect some "linear" and some "spherical" collisions and implement their results on the motion of the objects

... it's not really very constrained at all!
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!*
Spherical collisions

Zeroth approximation:

Stop $q$. Undo any overlap.

Make $\mathbf{r}.\mathbf{vel} = \mathbf{q}.\mathbf{vel}$.
Spherical collisions

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

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

Reality is just two eyes away!
Spherical collisions

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

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

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

Reality is just one eye away!
vPool – physics?

vPool – physics?


Therefore, the classical calculation only holds true when the speed of both colliding bodies is much lower than the speed of light (about 300 million m/s).

Two- and three-dimensional

For the case of two colliding bodies in two-dimensions, the overall velocity of each body must be split into two perpendicular velocities: one tangent to the common normal surfaces of the colliding bodies at the point of contact, the other along the line of collision. Since the collision only imparts force along the line of collision, the velocities that are tangent to the point of collision do not change. The velocities along the line of collision can then be used in the same equations as a one-dimensional collision. The final velocities can then be calculated from the two new component velocities and will depend on the point of collision. Studies of two-dimensional collisions are conducted for many bodies in the framework of a two-dimensional gas.

equations below...
The vPool project

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

- **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)

⇒ Phunky Physics is welcome!

A few examples to get you thinking...
Enjoy the projects! ... the graders certainly do!

An unusual variation on VPool 🐌
Enjoy the projects!

An unusual variation on VPool... the graders certainly do!

Questions? Thoughts?

Let's chat!

Good luck with hw#11!