Crushing Athletic Defeat

Pomona (PPI): The CS "NP-Complete Penguins" suffered a devastating blow here Saturday when they lost to the Physics "Lightspeed Rhinos" in the national gymnastics finals. The Penguins were leading the competition until the final round, having built on the outstanding flips performed by their star, Petunia "Pirouette" Penguin, and the leaps of the renowned Peter "Popup" Prancer. The Rhinos, by contrast, had had difficulty from the beginning. They upset the judges' table when they first entered, and sullied the gymnasium floor during their admittedly thrilling synchronized charging performance.

But disaster struck when one of the Rhinos "accidentally" gored Petunia and another stepped on Peter, squashing him flat. "I'm terribly sorry," apologized the perpetrator. "I just didn't see him through my beady little eyes."

Deprived of their "A" team, the Penguins were forced to replace them with an uncoordinated CS professor, who fell flat on his face and ensured the Penguin loss.
Inheritance

class Person(object):
    def __init__(self, first, last):
        self.firstName = first
        self.lastName = last

    def asleep(self, time):
        return time > 20 or time < 4 # MILITARY TIME IN HOURS

    def __repr__(self):
        return self.firstName + " " + self.lastName

>>> julie = Person("Julie", "Medero")
>>> julie
Julie Medero
>>> julie.asleep(2)
True
class Person(object):
    def __init__(self, first, last):
        self.firstName = first
        self.lastName = last

    def asleep(self, time):
        return time > 20 or time < 4 # MILITARY TIME IN HOURS

    def __repr__(self):
        return self.firstName + " " + self.lastName

class Student(Person):
    def __init__(self, first, last, age):
        super(Student, self).__init__(first, last)
        self.age = age

    def asleep(self, time):
        return 3 <= time <= 11

    def __repr__(self):
        return Person.__repr__(self) + ", " + str(self.age) + " years old"

>>> s = Student("Sue", "Persmart", 18)
>>> s
Sue Persmart, 18 years old
>>> s.asleep(2)
False

Sleeping until 11 AM!?
class Person(object):
    def __init__(self, first, last):
        self.firstName = first
        self.lastName = last

    def asleep(self, time):
        return time > 20 or time < 4 # MILITARY TIME IN HOURS

    def __repr__(self):
        return self.firstName + " " + self.lastName

class Student(Person):
    def __init__(self, first, last, age):
        super(Student, self).__init__(self, first, last)
        self.age = age

    def asleep(self, time):
        return 3 <= time <= 11

    def __repr__(self):
        return Person.__repr__(self) + ", " + str(self.age) + " years old"

class Mudder(Student):
    def __init__(self, first, last, age, dorm):
        super(Mudder, self).__init__(self, first, last, age)
        self.dorm = dorm

    def asleep(self, time):
        return False

>>> wally = Mudder("wally", "wart", 42, "west")

>>> wally
?

>>> wally.asleep()
?
Default Arguments

```python
class Student(object):
    def __init__(self, firstName, lastName, 
                 school = "HMC", major = "undeclared")
```

```python
>>> where = Student("Carmen", "Sandiego")
>>> stu = Student("Stu", "Dious", "PIT")
>>> anna = Student("Anna", "Litik", major = "Physics")
>>> elmo = Student("Elmo")
>>> bigBird = Student("Big", "Bird", firstName = "Tweety")
>>> bart = Student(school="PIT", "Bart", "Simpson")
```

In my experience, arguments are usually default of deperson who started them!
class Vector(object):
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def magnitude(self):
        blah, blah, blah
        return ...

    def normalize(self):
        mag = self.magnitude()
        newVector = Vector(
            self.x/mag, self.y/mag)
        self = newVector

Python Forbids Personality Transplants!

class Vector(object):
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def magnitude(self):
        blah, blah, blah
        return ...

    def normalize(self):
        mag = self.magnitude()
        self.x = self.x/mag
        self.y = self.y/mag
Assuming this all works out, the image in this tweet is also a valid ZIP archive, containing a multipart RAR archive, containing the complete works of Shakespeare.

This technique also survives twitter's thumbnailer :P
Connect Four!
Rules for Connect Four

Tic-Tac-Toe with stacking
7x6 board
Checkers slide down onto top of column
Four in a row (including diagonal) wins
A C4Board Class

class C4Board:
    width = 7
    height = 6
    def __init__(self):
        self.board = [C4Board.width*[0]
                      for i in range(C4Board.height)]
    ...

Artificial Intelligence

Catch-all term for anything that “appears human”

- Eliza
- Chess programs
- Expert systems

Many approaches
- N-ply lookahead:
  - good for games
Looking into the Future, Part I

Basic idea
- Guess a move
- See whether it makes your position better
- Repeat for many guesses, pick best result

What defines “better”? 
- Simple answer: use board evaluation function
Evaluating a Connect-4 Board

What are some ways to numerically rate a Connect-4 board?
If opponent has won, 0
If I have won, 100
Otherwise…

?
Looking Into the Future, Part II

Problem: evaluating a board is HARD
Solution: look at where the board leads

For each possible move:
  Make the move in “trial mode”
For each possible opponent's response:
  Make that move in “trial mode”
  Evaluate that board instead \(\text{Call ourselves recursively}\)
Choose opponent's best option
  This is the “worst case” for us
Our best move is “best of the worst cases”
(1) For each possible move
(2) Add it to the board
(3) Ask OPPONENT ('O') to score each board
(4) Which score would the opponent choose?

Worksheet problem: What should you ('X') assign for your score? (self's score)
What about if 'O' could look ahead one more move?
Limiting Recursion

When do we stop?

That's easy:

- When we find path that leads to a win
  - No matter what opponent does
- When all paths lead to losses (sigh)
- When we have explored all possible moves
How many boards in the tic-tac-toe move tree?

How many unique boards in the tic-tac-toe move tree?
The Move Tree

How many unique boards in the tic-tac-toe move tree?
Explosive Growth

Connect-4 move tree has $< 7^{42}$ nodes
  Roughly $311 \times 10^{33}$

Tree size depends on two parameters:
  Branching factor per move, B
  Maximum moves in game, N

Approximately $B^N$ possibilities
  Chess: $B \approx 35$, $N \approx 150$
  Backgammon: $B \approx 300$, $N \approx 50$

I can do that in my head!
Limiting Growth

Simplest strategy: stop looking ahead after evaluating N moves (“plies”)

Alternative: stop after $t$ seconds elapsed

Trickier: prune move tree

  Don't follow a path if first move is way worse

  Don't explore if already found better choice

Alpha/Beta pruning
The Curse of the AI Researcher

AI is always defined as “something a lot smarter than what my computer can do”

Corollary: if my computer can do it, it's not artificial intelligence

⇒ Every time we achieve AI, people decide it's not really AI!