What’s up this week?

• Text compression algorithms other than Huffman!
• Problem 1 (Lab): 3D Graphics!
• Problem 2: Connect 4 AI
  – Recursion
  – Loops
  – Object-oriented
Data Compression Beyond Huffman!

- Lempel-Ziv
- Arithmetic Coding
Lempel-Ziv

Alphabet: a, b, c

Encode:
• aabaaabaaaa
• ababcbababa
Lempel-Ziv

Encode: abbabba

Decode: 131241

Alphabet: a, b
Lempel-Ziv

Demo: LZ versus Huffman!

gzip, WinZip, compress
Arithmetic Coding

<table>
<thead>
<tr>
<th></th>
<th>(2^{-1})</th>
<th>(2^{-2})</th>
<th>(2^{-3})</th>
<th>(2^{-4})</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

\[
\begin{align*}
1 & \quad 0 \quad 0 \quad 0 \quad = \quad 1/2 \\
1 & \quad 1 \quad 0 \quad 0 \quad = \quad 3/4 \\
0 & \quad 1 \quad 0 \quad 1 \quad = \quad 5/16 
\end{align*}
\]
Arithmetic Coding

<table>
<thead>
<tr>
<th>Letter</th>
<th>freq</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>0.6</td>
</tr>
<tr>
<td>p</td>
<td>0.2</td>
</tr>
<tr>
<td>a</td>
<td>0.1</td>
</tr>
<tr>
<td>m</td>
<td>0.1</td>
</tr>
</tbody>
</table>

Encode: “sps”

Hold on to your socks! This could knock them off!
Arithmetic Coding

<table>
<thead>
<tr>
<th>Letter</th>
<th>freq</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>0.6</td>
</tr>
<tr>
<td>p</td>
<td>0.2</td>
</tr>
<tr>
<td>a</td>
<td>0.1</td>
</tr>
<tr>
<td>m</td>
<td>0.1</td>
</tr>
</tbody>
</table>

Encode: “sps”
Arithmetic Coding

<table>
<thead>
<tr>
<th>Letter</th>
<th>freq</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>0.6</td>
</tr>
<tr>
<td>p</td>
<td>0.2</td>
</tr>
<tr>
<td>a</td>
<td>0.1</td>
</tr>
<tr>
<td>m</td>
<td>0.1</td>
</tr>
</tbody>
</table>

Encode: “sps”
Arithmetic Coding

<table>
<thead>
<tr>
<th>Letter</th>
<th>freq</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>0.6</td>
</tr>
<tr>
<td>p</td>
<td>0.2</td>
</tr>
<tr>
<td>a</td>
<td>0.1</td>
</tr>
<tr>
<td>m</td>
<td>0.1</td>
</tr>
</tbody>
</table>

Encode: “sps”
Arithmetic Coding

<table>
<thead>
<tr>
<th>Letter</th>
<th>freq</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>0.6</td>
</tr>
<tr>
<td>p</td>
<td>0.2</td>
</tr>
<tr>
<td>a</td>
<td>0.1</td>
</tr>
<tr>
<td>m</td>
<td>0.1</td>
</tr>
</tbody>
</table>

Encode: “sps”

\[
0.011 = \frac{1}{4} + \frac{1}{8} = 0.375
\]
Arithmetic Coding

<table>
<thead>
<tr>
<th>Letter</th>
<th>freq</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>0.5</td>
</tr>
<tr>
<td>p</td>
<td>0.5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Letter</th>
<th>freq</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>0.1</td>
</tr>
<tr>
<td>p</td>
<td>0.9</td>
</tr>
</tbody>
</table>
Arithmetic Coding

<table>
<thead>
<tr>
<th>Letter</th>
<th>freq</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>0.5</td>
</tr>
<tr>
<td>p</td>
<td>0.5</td>
</tr>
</tbody>
</table>

0 \( \frac{1}{4} \) 1/2 3/4 1

ss sp ps pp

How many bits are used to encode “pp”?

<table>
<thead>
<tr>
<th>Letter</th>
<th>freq</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>0.1</td>
</tr>
<tr>
<td>p</td>
<td>0.9</td>
</tr>
</tbody>
</table>

How many bits are used to encode “pp” in this case?
Arithmetic Coding

<table>
<thead>
<tr>
<th>Letter</th>
<th>freq</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>0.5</td>
</tr>
<tr>
<td>p</td>
<td>0.5</td>
</tr>
</tbody>
</table>

How many bits are used to encode “pp”?

<table>
<thead>
<tr>
<th>Letter</th>
<th>freq</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>0.1</td>
</tr>
<tr>
<td>p</td>
<td>0.9</td>
</tr>
</tbody>
</table>

How many bits are used to encode “pp” in this case?
Arithmetic Coding and IBM

- Patent 4,122,440 — (IBM) Filed March 4, 1977, Granted 24 October 1978 (Now expired)
- Patent 4,286,256 — (IBM) Granted 25 August 1981 (presumably now expired)
- Patent 4,467,317 — (IBM) Granted 21 August 1984 (presumably now expired)
- Patent 4,652,856 — (IBM) Granted 4 February 1986 (presumably now expired)
- Patent 4,891,643 — (IBM) Filed 15 September 1986, granted 2 January 1990
- Patent 4,905,297 — (IBM) Granted 27 February 1990
- Patent 4,935,882 — (IBM) Granted 19 June 1990
- Patent 4,989,000 — Filed 19 June 1989, granted 29 January 1991
- Patent 5,099,440 — (IBM) Filed 5 January 1990, granted 24 March 1992
- Patent 5,272,478 — (Ricoh) Filed 17 August 1992, granted 21 December 1993

JPEG and bzip2 opted for Huffman!
The 3D Version of Shapes

GlowScript 2.3 VPython

floor = box(length=4, height=0.5, width=4, color=color.blue)

ball = sphere(pos=vector(0,4,0), color=color.red)
bball.vel = vector(0,-1,0)

RATE = 30
dt = 1.0/RATE
while True:
    rate(RATE)
    ball.pos = ball.pos + ball.vel*dt
    if ball.pos.y < 1:
        ball.vel.y = -ball.vel.y
    else:
        ball.vel.y = ball.vel.y - 9.8*dt
Demo of Ran’s Zombie Game!

Super cheezy!
class robot:
    def __init__(self, position=vector(0,0,0), heading=vector(0,0,1)):
        self.base = position
        self.heading = vector(norm(heading))
        self.parts = []
        self.speed = 0

    def step(self):
        self.base = self.base + self.heading * self.speed
        for part in self.parts:
            part.pos = part.pos + self.heading * self.speed

    def turn(self, angle):
        theta = radians(angle)
        self.heading = rotate(self.heading, angle=theta, axis=vector(0,1,0))
        for part in self.parts:
            part.rotate(angle=theta, axis=vector(0,1,0), origin=self.base)

class ranbot(robot):
    def __init__(self, pos=vector(0,0,0), heading=vector(0,0,1), speed=0.3):
        robot.__init__(self, pos, heading)
        self.body = cylinder(pos = self.base+vector(0, 0.5, 0), axis=vector(0, 6, 0), radius=1, color=color.red)
        self.head = box(pos= vector(0,7,0)+self.base, length = 2, width = 2, height = 2, color=color.green)
        self.nose = cone(pos = vector(0,7,1)+self.base, radius = 0.5, axis=vector(0,0,1), color=color.yellow)
        self.wheel1 = cylinder(pos = self.base + vector(1, 1, 0), axis=vector(0.5, 0, 0), radius = 1, color=color.blue)
        self.wheel2 = cylinder(pos = self.base + vector(-1, 1, 0), axis=vector(-0.5, 0, 0), radius = 1, color=color.blue)
        self.parts = [self.body, self.head, self.nose, self.wheel1, self.wheel2]
        self.speed = speed

class zombie(robot):
    def __init__(self, position=vector(0,0,0), heading=vector(0,0,1)):
        robot.__init__(self, position, heading)
        self.body = cylinder(pos = self.base, axis=vector(0, 4, 0), radius=1, color=color.green)
        self.arm1 = cylinder(pos = self.base+vector(0.6,3,0), axis=vector(0,0,2), radius=0.3, color=color.yellow)
        self.arm2 = cylinder(pos = self.base+vector(-0.6,3,0), axis=vector(0,0,2), radius=0.3, color=color.yellow)
        self.halo = ring(pos = self.base+vector(0, 5, 0), axis=vector(0,1,0), radius=1, color=color.yellow)
        self.head = sphere(pos = self.base+vector(0, 4.5, 0), radius = 0.5, color=color.white)
        self.parts = [self.body,self.arm1, self.arm2, self.halo, self.head]
Math and random packages are automatically imported as
```
from math import *
from random import *
```

def makeZombies(numZombies):
    zombies = []
    for i in range(numZombies):
        theta = 360 * random()
        r = GROUND_RADIUS * random()
        x = r * cos(theta)
        z = r * sin(theta)
        zom = zombie(position=vector(x, 0, z))
        zom.speed = 0.3
        zombies.append(zom)
    return zombies

```

```
scene.bind('keydown', process)  # "Register" a function called process
```

def process(event):
    keyCode = event.which  # event.which is the numerical code of the key pressed
    if keyCode == 39:  # right arrow
        userbot.turn(5)
    elif keyCode == 37:  # left arrow
        userbot.turn(-5)
    elif keyCode == 38:  # up arrow
        userbot.speed += 0.1
    elif keyCode == 40:  # down arrow
        userbot.speed -= 0.1
    elif keyCode == 32:  # space bar
        ballInAction = True
The game...

GROUND RADIUS = 50
INITIAL ZOMBIES = 20

Global variables set somewhere near the top of the file for easy changing!

ground = cylinder(pos=vector(0,-1,0), axis=vector(0,1,0), radius = GROUND RADIUS)
userbot = ranbot()
zombies = makeZombies(INITIAL ZOMBIES)

while True:
    rate(30)
    userbot.step()
    if mag(userbot.base) >= GROUND RADIUS: userbot.turn(180)
    for zom in zombies:
        if mag(zom.base - userbot.base) < 2:
            zom.speed = 3
            zom.step()
    if mag(zom.base) >= GROUND RADIUS: zom.turn(180 + random() * 10 - 20)
Connect 4

Player 1

Player 2
Artificial Intelligence and Games

Player 1

Player 2
Artificial Intelligence and Games

Player 1

Player 2
Artificial Intelligence and Games
Artificial Intelligence and Games

Player 1

Player 2
Artificial Intelligence and Games

Player 1

Player 2
Artificial Intelligence and Games

Player 1

Player 2
Artificial Intelligence and Games

Player 1

Player 2
Artificial Intelligence and Games

Player 1

Player 2
k-ply lookahead

X is the human player (Player 1)
O is the computer player (Player 2)

Consider “Connect 3” on
board with 5 columns and 6
rows

1-ply lookahead

Fine for O
Fine for O
Fine for O
Fine for O
Fine for O
New! Improved! Now with 2-ply lookahead!

“Connect 3”

When examining X’s possible moves, expect the worst!
New! Improved! Now with 2-ply lookahead!

“Connect 3”

O moves

O moves

Bad for O

Fine for O  Bad for O  Fine for O  Fine for O  Fine for O

When examining X’s possible moves, expect the worst!
New! Improved! Now with 2-ply lookahead!

“Connect 3”

O moves

X _ X _ _

O moves

Bad for O

X _ X _ _

X moves

Fine for O  Bad for O  Fine for O  Fine for O  Fine for O

When examining X’s possible moves, expect the worst!
New! Improved! Now with 2-ply lookahead!

“Connect 3”

When examining x’s possible moves, expect the worst!
New! Improved! Now with 2-ply lookahead!

“Connect 3”

When examining X’s possible moves, expect the worst!
New! Improved! Now with 2-ply lookahead!

“Connect 3”

O
X _ X _ _

O moves

Bad for O
Fine for O
Bad for O
Bad for O
Bad for O
New! Improved! Now with 2-ply lookahead!

“Connect 3”

When examining X’s possible moves, expect the worst!

Fine for X  Good for X  Fine for X  Fine for X  Fine for X
New! Improved! Now with 2-ply lookahead!

"Connect 3"

When examining X’s possible moves, expect the worst!
New! Improved! Now with 2-ply lookahead!

“Connect 3”

When examining X’s possible moves, expect the worst!
An object-oriented design

```python
>>> b = Board(7, 6)
>>> px = Player('X', 'LEFT', 3)
>>> po = Player('O', 'LEFT', 2)
>>> playGame(b, px, po)

# Lots of boards omitted...

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

---------------
0 1 2 3 4 5 6

O wins!
```

ply number

A Player will spawn other Players recursively!

LEFT, RIGHT, or RANDOM
The Complexity of Search

Consider 5-ply lookahead in Connect 4. How many “nodes” are explored?

\[ 7^5 = 16807 \]

8-ply?

\[ 7^8 = 5.7 \text{ million} \]

Looking all the way to the end?

\[ 10^{14} \]
The Complexity of Search

Chess: $10^{50}$
Connect 6: $10^{172}$
Go (19x19): $10^{171}$

These problems are apparently “harder” than NP-hard! (their general forms are PSPACE-hard)