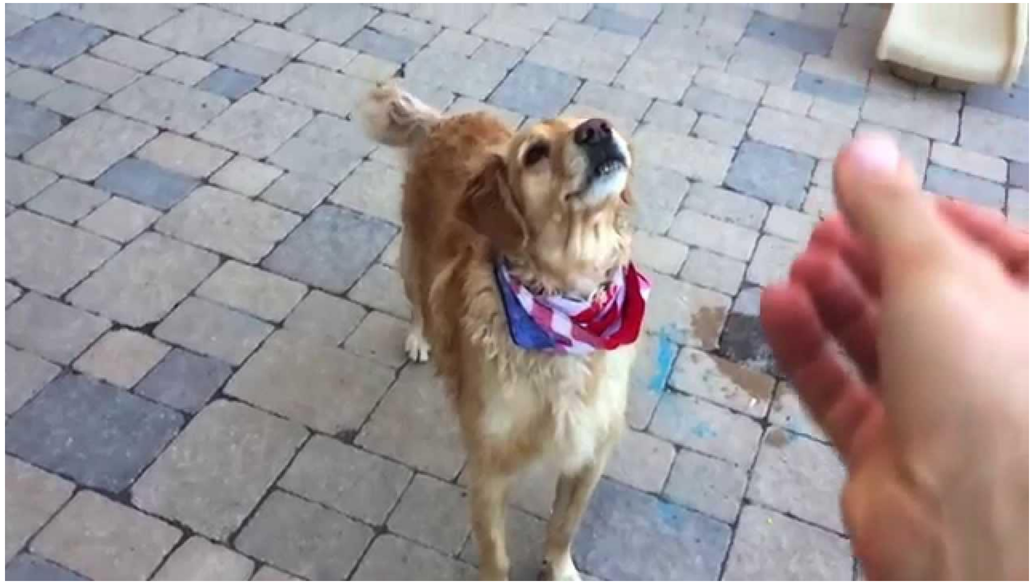
Using Differential Equations and Artificial Intelligence in Modeling Fritz

Team Number: 1148, Problem C
Lillie Mohn, Nathan Tam, Mark Longenberger
Coach: Cesar Martínez-Garza
Penn State University-Berks Campus, Reading, PA, USA
SIMIODE SCUDEM 2023

Overview

- Introduction of the Problem
- Initial Brainstorming Process
- Differential Modeling of Food Object
- General Assumptions
- Differential Modeling of Fritz
- Code
- Results
- Conclusion



The Problem

- Meet Fritz! He cannot catch food tossed to him

- Is Fritz clumsy?
- Is his owner bad at tossing food?

Initial Brainstorming

- 3D Probability Density Function for Modeling of Fritz's Likelihood of Catching Food

- Arc Length Parametrization of a Curve in R^3

$$s(t) = \int_0^t \left\| \mathbf{r}'(u) \right\| du$$

where $\mathbf{r}(t) = \langle \mathbf{x}(t), \mathbf{y}(t), \mathbf{z}(t) \rangle$, a parametrized curve through R^3 space

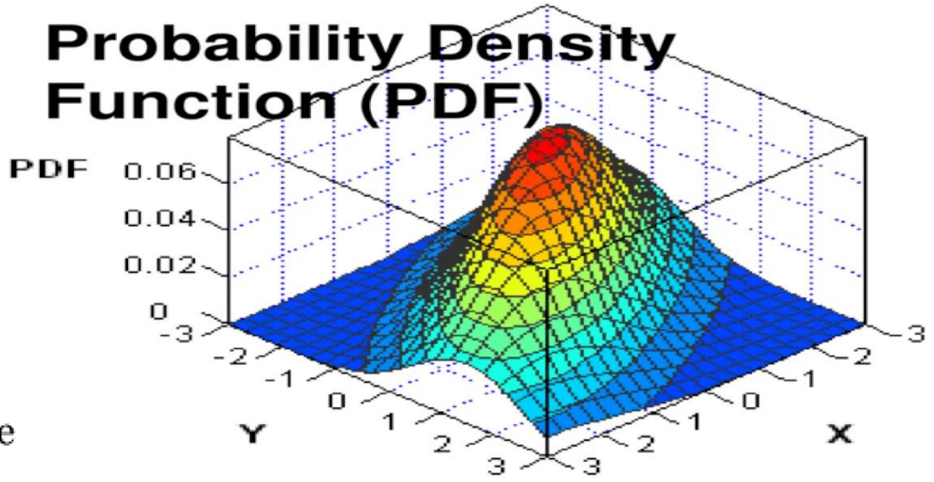


Photo credits: https://vdocuments.mx/probability-density-function-pdf.html

Differential Equations-Air Resistance

- Differential Equations (No air Resistance)

$$\ddot{y} = -9.8$$

$$\ddot{x} = 0$$

- Differential Equations (Air Resistance)

$$\ddot{y} - \frac{\rho C_d A}{2m}\dot{y}^2 = -9.8$$

$$\ddot{x} - \frac{\rho C_d A}{2m}\dot{x}^2 = 0$$

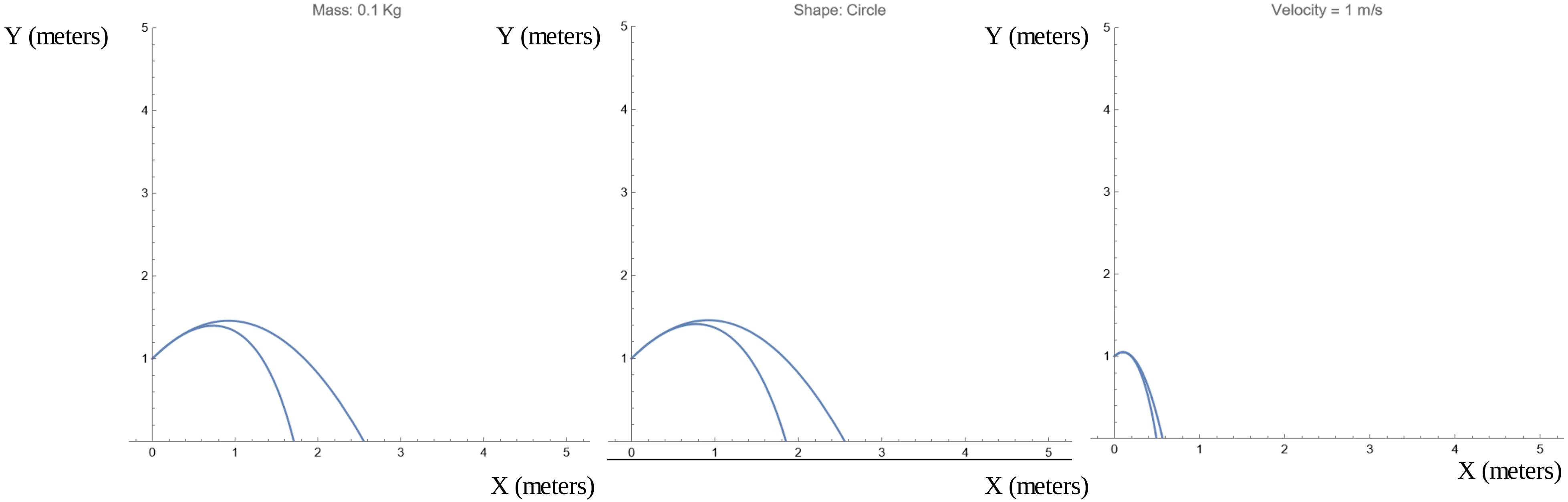
ρ - Air Density

C_d - Coefficient of Drag

A - Reference Area

m – Mass

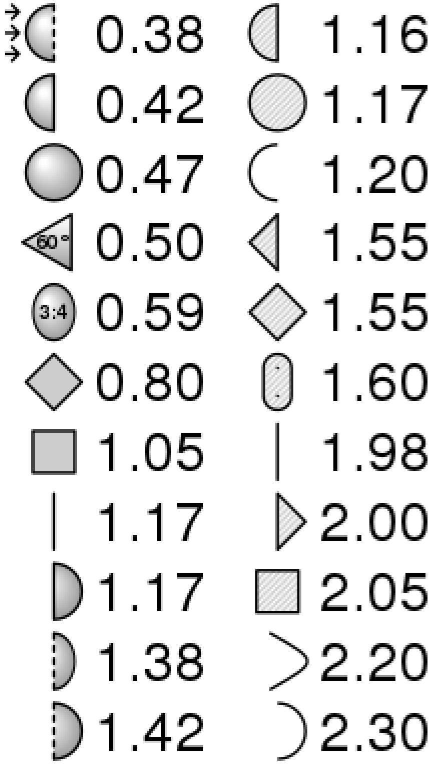
Differential Equations-Solutions



Changing Mass

Changing Shape

Changing Velocity



Air Density Remains Constant

Graphs Generated in Mathematica

Assumptions Moving Forward

- Air resistance will be neglected, therefore the size of the object does not matter

- Launch angle will remain constant

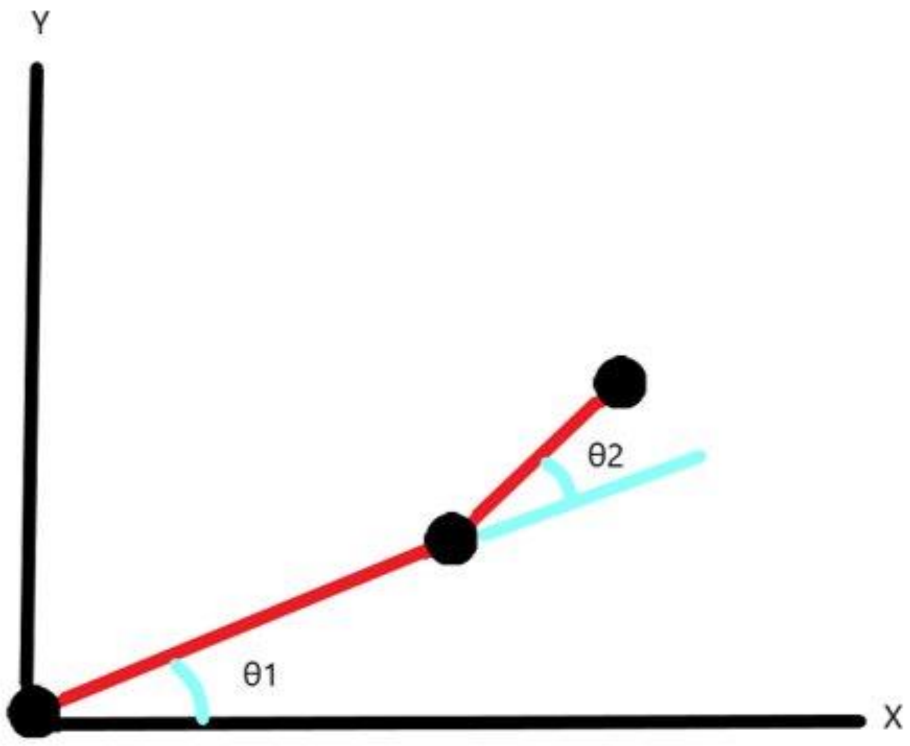
- Motion and model will be in two dimensions

- No wind

General Model - Using AI

- Reflects Learning
- Variables can be changed
- More complexity than Basic Ordinary Differential Equations
- Applicable to Modern Work Environment
- Follows Differential Motion

General Layout - Fritz



- Two Link Robot
  - Simplification
  - Ease of Programming

- Kinematic Equations

$$\begin{pmatrix} x_1 \\ y_1 \end{pmatrix} = \begin{pmatrix} l_1 \cos(\theta_1) + \ l_2 \cos(\theta_1 + \theta_2) \\ l_1 \sin(\theta_1) + \ l_2 \sin(\theta_1 + \theta_2) \end{pmatrix}$$

$$0 \leq \ \theta_1 \leq \pi$$

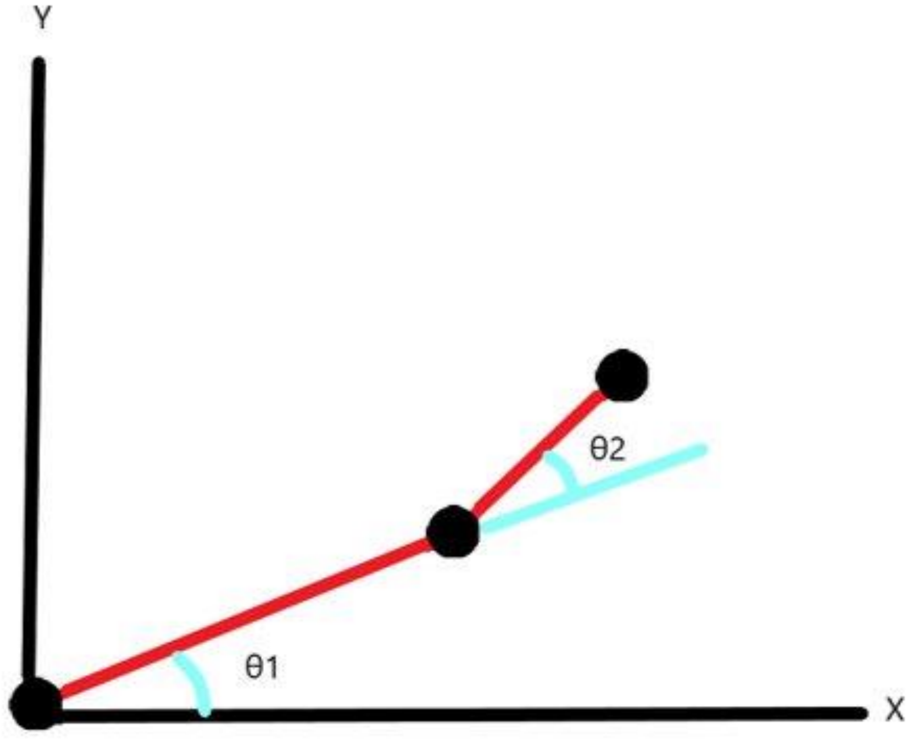
$$-\frac{\pi}{2} \leq \ \theta_2 \ \leq \frac{\pi}{2}$$

- Inverse Kinematics

$$\theta_2 = \cos^{-1}\left(\frac{x^2 + y^2 - {l_1}^2 - {l_2}^2}{2 l_1 l_2}\right)$$

$$\theta_1 = \tan^{-1}\left(\frac{y}{x}\right) - \tan^{-1}\left(\frac{l_1 \sin(\theta_2)}{l_1 + l_2 \cos(\theta_2)}\right)$$

Differential Motion - Fritz



Starting with Kinematic Equations:

$$\begin{pmatrix} x_1 \\ y_1 \end{pmatrix} = \begin{pmatrix} l_1\cos(\theta_1) + \ l_2\cos(\theta_1+\theta_2) \\ l_1\sin(\theta_1) + \ l_2\sin(\theta_1+\theta_2) \end{pmatrix} \qquad \begin{matrix} 0 \le \ \theta_1 \le \pi \\ -\frac{\pi}{2} \le \ \theta_2 \ \le \frac{\pi}{2} \end{matrix}$$

Remember Angles are a Function of Time:

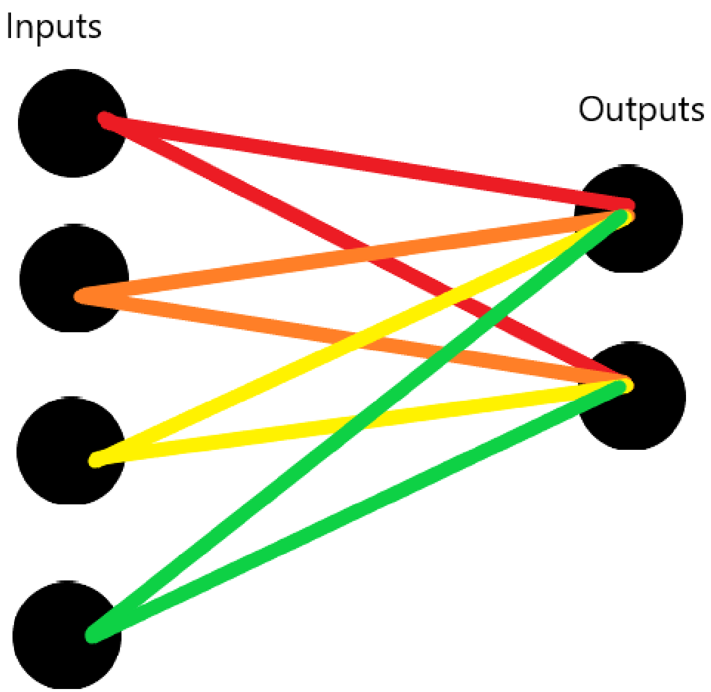
$$\theta_1 = \theta_1(t) \quad \theta_2 = \theta_2(t)$$

Computing Derivative W.R.T. Time:

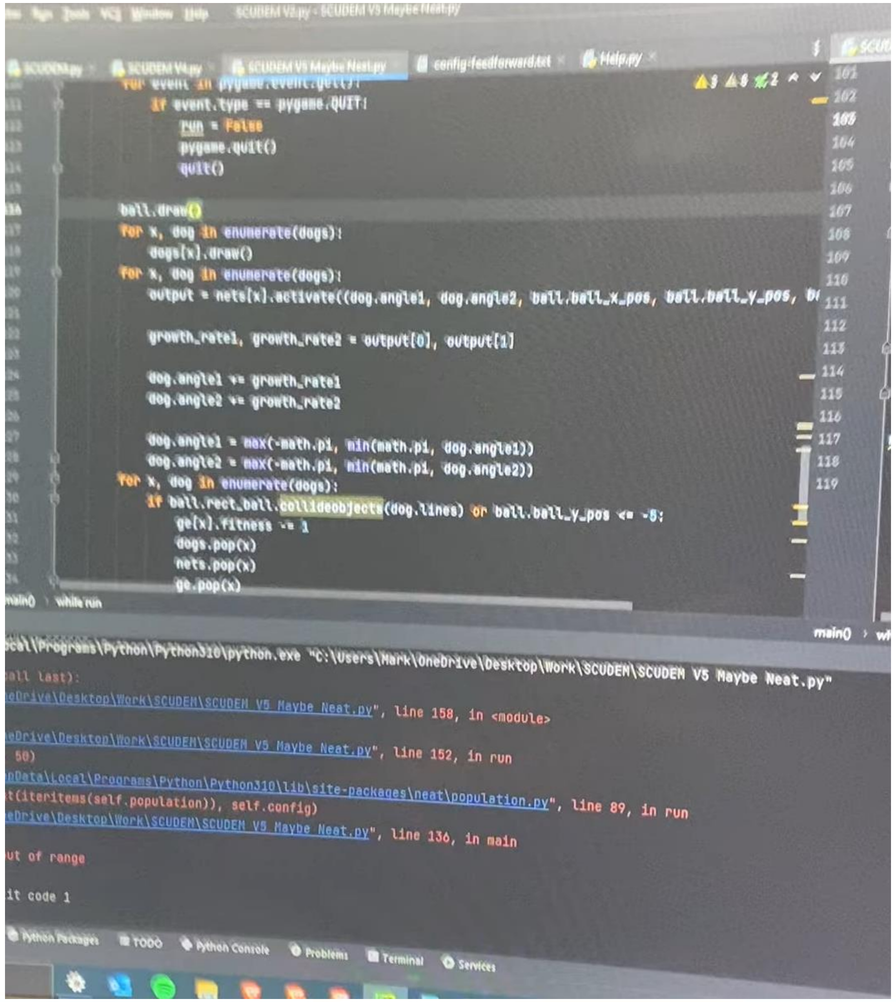
$$\begin{pmatrix} \dot{x} \\ \dot{y} \end{pmatrix} = \begin{matrix} -l_1\dot{\theta}_1\sin(\theta_1) - l_2(\dot{\theta}_1+\dot{\theta}_2)\sin\,(\theta_1+\theta_2) \\ l_1\dot{\theta}_1\cos(\theta_1) + l_2(\dot{\theta}_1+\dot{\theta}_2)\cos\,(\theta_1+\theta_2) \end{matrix}$$

AI Implementation

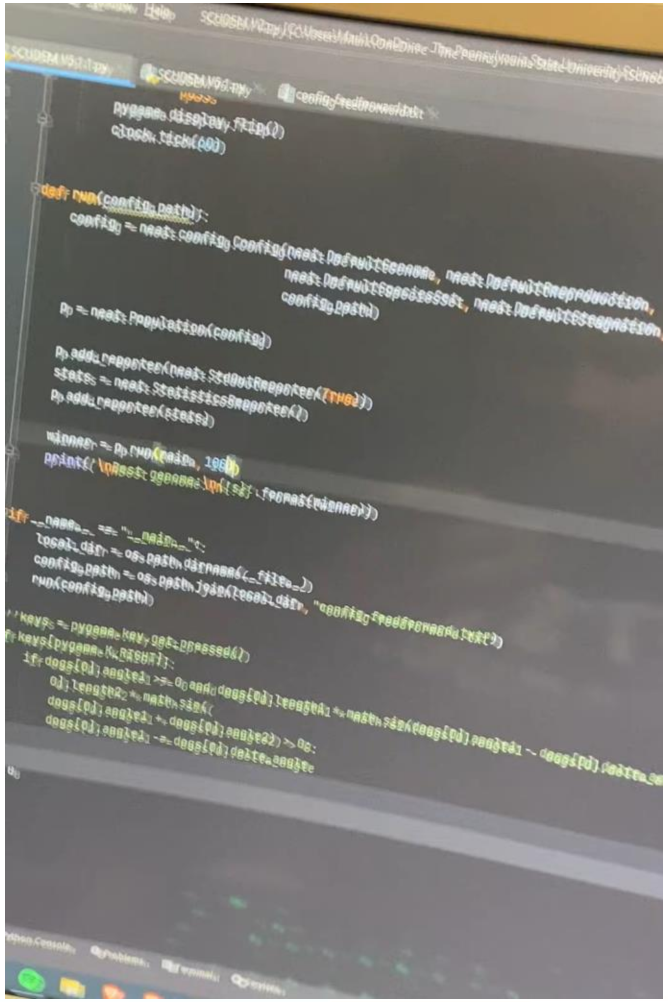
- **N**euro**E**volution of **A**ugmented **T**opologies (NEAT)
- Inputs
- Outputs
- Activation Function
- Population Size
- Fitness Function
- Max Generations



Failures



if event.type == pygame.QUIT:
run = False
pygame.quit()
quit()
ball.draw()
for x, dog in enumerate(dogs):
dogs[x].draw()
growth_rate1, growth_rate2 = output[0], output[1]
dog.angle1 += growth_rate1
dog.angle2 += growth_rate2
dog.angle1 = max(-math.pi, min(math.pi, dog.angle1))
dog.angle2 = max(-math.pi, min(math.pi, dog.angle2))
dogs.pop(x)
nets.pop(x)
ge.pop(x)



def run(config_path):
p = neat.Population(config)
stats = neat.StatisticsReporter()
p.add_reporter(stats)
run(config_path)

PennState

Values for AI

- **Inputs** – Theta 1 & 2, Fritz to Food Distance, Food X&Y Position and Time
- **Outputs-** Increase or Decrease to Theta 1 &2
- **Activation Function –** Hyperbolic Tangent
- **Population Size -** 100 Per Generation
- **Fitness Function** – "Reward" for Catches, Scaled "Punishment" for Misses
- **Max Generations –** Approx. 500 Generations

Code Part 1

```python
import pygame
import math
from random import randint
import neat
import os

SCREEN_WIDTH = 1366
SCREEN_HEIGHT = 768
screen = pygame.display.set_mode((SCREEN_WIDTH, SCREEN_
pygame.display.set_caption("SCUDEM")
clock = pygame.time.Clock()

# "Constant Parameters
# dog
dog_angle1 = 1
dog_angle2 = 0
dog_length1 = 100
dog_length2 = 50
dog_base_pos_x = 200  # Relative to Cord System
dog_base_pos_y = 0  # Relative to Cord System
dog_delta_angle = 0.05
dog_mouth_rect = 0
dog_lines = 0
# ball
ball_x_pos = 1000
ball_y_pos = 100
ball_angle = .5
ball_velocity = 9
ball_rect_ball = 0
catches = 0


def to_pygame(x, y):
    return x, -y + SCREEN_HEIGHT - 30


def distance_calc(dog, ball):
    dog_pos = dog.base_pos_x + (
            dog.length1 * math.cos(dog.angle1) + dog.length2 * math.cos(dog.angle1 + dog.angle2)), -1 * (
                       dog.length1 * math.sin(dog.angle1) + dog.length2 * math.sin(
                   dog.angle1 + dog.angle2)) + SCREEN_HEIGHT
    ball_pos = ball.ball_x_pos - 5, -ball.ball_y_pos + SCREEN_HEIGHT
    distance = ((dog_pos[0] - ball_pos[0]) ** 2 + (dog_pos[1] - ball_pos[1]) ** 2) ** 1 / 2
    return distance


1 usage
class Dog:
    def __init__(self, color):
        self.angle1 = dog_angle1
        self.angle2 = dog_angle2
        self.length1 = dog_length1
        self.length2 = dog_length2
        self.base_pos_x = dog_base_pos_x  # Relative to Cord System
        self.base_pos_y = dog_base_pos_y  # Relative to Cord System
        self.delta_angle = dog_delta_angle
        self.mouth_rect = dog_mouth_rect
        self.lines = dog_lines
        self.color = color
        self.distance = 100000000
        self.min_distance = 1000000000000
        self.dog_x_pos = 0
        self.dog_y_pos = 0
        self.caught = False

    def draw(self):
        def to_robot(x, y):
            return x + self.base_pos_x, -y + SCREEN_HEIGHT - 30

        pygame.draw.line(screen, (0, 0, 0), to_pygame( x: 0, -10), to_pygame(SCREEN_WIDTH, -10), width=10)
        pygame.draw.circle(screen, self.color, to_robot( x: 0,  y: 0), radius=10)
        line_body = pygame.draw.line(screen, self.color, start_pos=to_robot( x: 0,  y: 0),
                                     end_pos=to_robot(self.length1 * math.cos(self.angle1),
                                                      self.length1 * math.sin(self.angle1)), width=5)
        pygame.draw.circle(screen, self.color,
                           to_robot(self.length1 * math.cos(self.angle1),
                                    self.base_pos_y + self.length1 * math.sin(self.angle1)), radius=10)
        line_head = pygame.draw.line(screen, self.color,
                                     to_robot(self.length1 * math.cos(self.angle1),
                                              self.length1 * math.sin(self.angle1)),
                                     to_robot(self.length1 * math.cos(self.angle1) + self.length2 * math.cos(
                                         self.angle1 + self.angle2),
                                              self.length1 * math.sin(self.angle1) + self.length2 * math.sin(
                                                  self.angle1 + self.angle2)),
                                     width=5)
        pygame.draw.circle(screen, self.color,
                           to_robot(self.length1 * math.cos(self.angle1) + self.length2 * math.cos(
                               self.angle1 + self.angle2),
                                    self.length1 * math.sin(self.angle1) + self.length2 * math.sin(
                                        self.angle1 + self.angle2)), radius=10)
        self.dog_x_pos = self.length1 * math.cos(self.angle1) + self.length2 * math.cos(self.angle1 + self.angle2)
        self.dog_y_pos = self.length1 * math.sin(self.angle1) + self.length2 * math.sin(self.angle1 + self.angle2)
        self.mouth_rect = pygame.Rect(
            self.base_pos_x + (
                    self.length1 * math.cos(self.angle1) + self.length2 * math.cos(self.angle1 + self.angle2)) - 5,
            -1 * (self.length1 * math.sin(self.angle1) + self.length2 * math.sin(
                self.angle1 + self.angle2)) + SCREEN_HEIGHT - 35, 10, 10)
        pygame.draw.rect(screen, (0, 0, 0), self.mouth_rect, width=1)

        self.lines = [
            line_head,
            line_body
        ]

    4 usages (4 dynamic)
    def reset(self):
        self.angle1 = dog_angle1
        self.angle2 = dog_angle2
        self.length1 = dog_length1
        self.length2 = dog_length2
        self.base_pos_x = dog_base_pos_x  # Relative to Cord System
        self.base_pos_y = dog_base_pos_y  # Relative to Cord System
        self.delta_angle = dog_delta_angle
```

Code Part 2

```python
        self.mouth_rect = dog_mouth_rect
        self.lines = dog_lines

        self.distance = 100000000
        self.min_distance = 1000000000000
        self.dog_x_pos = 0
        self.dog_y_pos = 0
        self.caught = True


1 usage
class Ball:
    def __init__(self, ):
        self.ball_x_pos = ball_x_pos
        self.ball_y_pos = ball_y_pos
        self.ball_angle = ball_angle
        self.ball_velocity = ball_velocity
        self.rect_ball = ball_rect_ball
        self.x_vel = 0
        self.y_vel = 0

    def draw(self, time):
        self.ball_x_pos -= self.ball_velocity * math.cos(self.ball_angle) * time
        self.ball_y_pos += self.ball_velocity * math.sin(self.ball_angle) * time - 4.9 * time ** 2
        self.x_vel = self.ball_velocity * math.cos(self.ball_angle)
        self.y_vel = self.ball_velocity * math.sin(self.ball_angle) - 9.8 * time
        pygame.draw.circle(screen, (255, 0, 0), to_pygame(self.ball_x_pos, self.ball_y_pos), radius=10)
        self.rect_ball = pygame.Rect(self.ball_x_pos - 5, -self.ball_y_pos + SCREEN_HEIGHT - 35, 10, 10)
        pygame.draw.rect(screen, (0, 0, 0), self.rect_ball, width=1)

    4 usages (4 dynamic)
    def reset(self):
        self.ball_x_pos = ball_x_pos
        self.ball_y_pos = ball_y_pos
        self.ball_angle = ball_angle
        self.ball_velocity = ball_velocity
        self.rect_ball = ball_rect_ball
        self.x_vel = 0
        self.y_vel = 0
```

```python
def main(genomes, config):
    pygame.time.wait(5000)
    pygame.display.set_caption("SCUDEM")
    pygame.init()
    run = True
    time = 0
    round = 0
    catches = 0
    ball = Ball()
    dogs = []
    ge = []
    nets = []
    outputs = []
    for _, g in genomes:
        net = neat.nn.FeedForwardNetwork.create(g, config)
        nets.append(net)
        dogs.append(Dog((randint(0, 255), randint(0, 255), randint(0, 255))))
        g.fitness = 0
        ge.append(g)

    # Just for manual play

    while run:
        time += 1 / 60
        screen.fill((255, 255, 255))
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                run = False

                pygame.quit()
                quit()
        pygame.display.set_caption(str(catches))
        if len(dogs) == 0:
            run = False
            print(catches)
            break
        ball.draw(time)
        for x, dog in enumerate(dogs):
            dog.draw()
```

```python
            dog.distance = distance_calc(dog, ball)
            if distance_calc(dog, ball) < dog.min_distance:
                dog.min_distance = distance_calc(dog, ball)
            output = nets[x].activate(
                (dog.angle1, dog.angle2, dog.distance, ball.ball_x_pos, ball.ball_y_pos, ball.x_vel, ball.y_vel,
                 dog.dog_x_pos, dog.dog_y_pos, time))  #

            if output[0] > 0.5:
                if dog.angle1 >= 0 and dog.length1 * math.sin(dog.angle1 - dog.delta_angle) + dog.length2 * math.sin(
                        dog.angle1 + dog.angle2) > 0:
                    dog.angle1 -= dog.delta_angle
            elif output[0] < -0.5:
                if dog.angle1 <= math.pi and dog.length1 * math.sin(
                        dog.angle1 + dog.delta_angle) + dog.length2 * math.sin(dog.angle1 + dog.angle2) > 0:
                    dog.angle1 += dog.delta_angle
            else:
                pass
            if output[1] > 0.5:
                if dog.angle2 >= -math.pi/2 and dog.length1 * math.sin(dog.angle1) + dog.length2 * math.sin(
                        dog.angle1 + dog.angle2 - dog.delta_angle) > 0:
                    dog.angle2 -= dog.delta_angle
            elif output[1] < -0.5:
                if dog.angle2 <= math.pi/2 and dog.length1 * math.sin(dog.angle1) + dog.length2 * math.sin(
                        dog.angle1 + dog.angle2 + dog.delta_angle) > 0:
                    dog.angle2 += dog.delta_angle

        for x, dog in enumerate(dogs):
            if ball.ball_y_pos > -50:
                if ball.rect_ball.clip(dog.mouth_rect):
                    ge[x].fitness += 10000
                    dog.caught = True
                    catches += 1
                    dogs.pop(x)
                    nets.pop(x)
                    ge.pop(x)
                elif ball.rect_ball.collideobjects(dog.lines) and dog.caught == False:
                    ge[x].fitness -= 100 + dog.min_distance/100
                    dogs.pop(x)
                    nets.pop(x)
```

AI Model

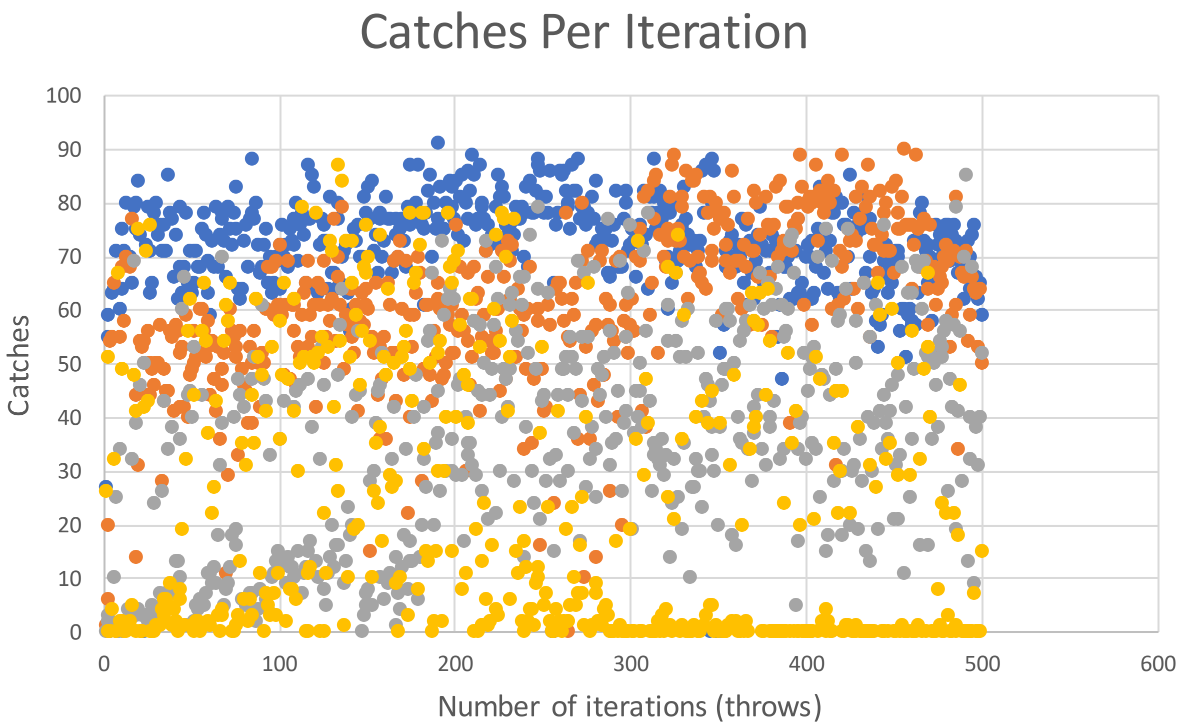
AI Manipulation

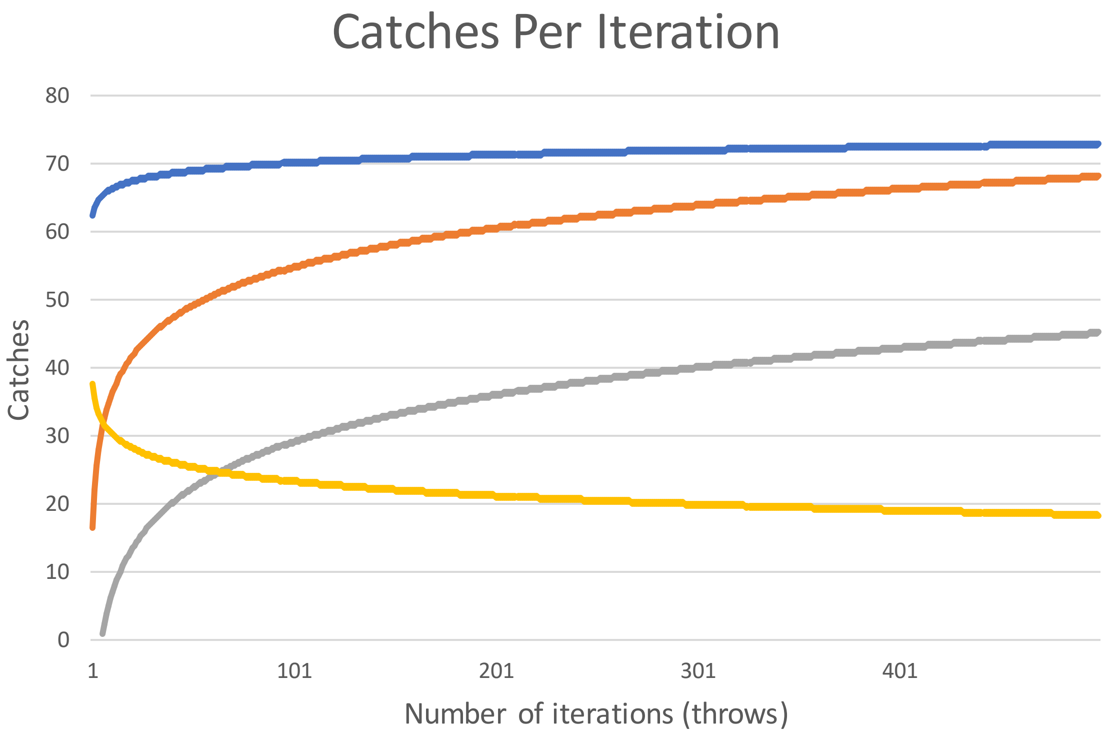
- Variables Modified for Food:
  - Velocity
  - Starting Position of Food (Distance and Height)

- Variable Modified for Fritz:
  - Iterations
  - Number of Fritz's per iteration

- Additional Assumptions
  - Food will always cross through domain of Fritz's movement
  - Once food touches ground or Fritz, food is marked as not caught

Results





Constant initial velocity and initial food position

Varied initial velocity, constant initial food position

Constant initial velocity, varied initial food position

Varied initial velocity and initial food position

Conclusion

- How precisely does the dog's brain have to estimate the orientation of his head and eyes in order to best predict the location of the item?
- How much time will the dog have to orient its mouth to the correct position?
- Are certain foods more difficult to catch due to different effects of air friction?
- Does the height at which the object is thrown matter?
- What are the most important aspects of the dog's perceptions and behaviors for the dog's success?
- Use your analysis of the situation to decide if Fritz is just clumsy or if his owner is being mean to the dog on the Internet.

Future Directions

- Improve model for multiple variable changes
- Delay reaction time
- Introduce noise for different-sized foods
- Change dog dimensions and mouth size
- Add air friction inside the model
- Expand model to 3D

Citations

- Benson, Tom. "The Drag Equation." *NASA*, 13 May 2021, www.grc.nasa.gov/www/k-12/rocket/drageq.html.
- Corke, Peter. "Inverse Kinematics for a 2-Joint Robot Arm Using Geometry." *Robot Academy*, robotacademy.net.au/lesson/inverse-kinematics-for-a-2-joint-robot-arm-using-geometry/.
- Corke, Peter. "Velocity of 2-Joint Planar Robot Arm." *Robot Academy*, robotacademy.net.au/lesson/velocity-of-2-joint-planar-robot-arm/.
- "Drag Coefficient." *Wikipedia, the Free Encyclopedia*, Wikimedia Foundation, Inc, 12 Nov. 2023, en.wikipedia.org/wiki/Drag_coefficient. Accessed 12 Nov. 2023.
- "Golden Retriever." *Dimensions.com*, https://www.dimensions.com/element/golden-retriever-dog#:~:text=Golden%20Retrievers.
- "Python Flappy Bird AI Tutorial (with NEAT) - NEAT Configuration and Explanation." *YouTube*, www.youtube.com/watchv=MPFWsRjDmnU&list=PLzMcBGfZo4-lwGZWXz5Qgta_YNX3_vLS2&index=5.
- Stanley, K.O., and R. Miikkulainen. "Efficient evolution of neural network topologies." *Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No.02TH8600)*, 2002.