EE475 Final Project: Snake Game

Group Members: Zhenguo Mo, Yi Shen, Yihan Ping

1. Topic

Our project focus on using genetic algorithms and neural networks in a snake game to make it a self learner.

The snake game is a simple but classic and challenging game since it was invented. Many game enthusiasts try to use artificial intelligence to make snakes get higher scores. Previously, snake game learned by its self using A* ,BFS and DFS algorithms. However when the snake get longer and longer, the snake will touch itself first before get the food. Therefore, we want to apply a different algorithms to the snake game.With the new algorithms, the snake lives longer and can get at least 90 scores.

2. Design Description

There are several challenges when modeling artificial intelligence methods for autonomous players on games (bots). NEAT is one of the models that, combining genetic algorithms and neural networks, seek to describe a bot behavior more intelligently.

In NEAT, a neural network is used for decision making, taking relevant inputs from the environment and giving real-time decisions. In a more abstract way, a genetic algorithm is applied for the learning step of the neural networks' weights, layers, and parameters.

Part 1: Initialize the Game

Firstly a game itself is written. It will have a 720x480 field, a snake of 3 pieces at the start, one randomly generated food at each moment in time. The score will appear on the top left corner.

In [ ]:

```
import pygame
import sys
import time
import random
import math
import numpy as np
import neat
import os
import pickle

directions = {0: 'UP', 1: 'RIGHT', 2:'LEFT', 3:'DOWN'}
directions_to_num = {'RIGHT': 1, 'DOWN': 2, 'LEFT': 3, 'UP': 0}

black = pygame.Color(0, 0, 0)
white = pygame.Color(255, 255, 255)
red = pygame.Color(255, 0, 0)
green = pygame.Color(0, 255, 0)
blue = pygame.Color(0, 0, 255)
frame_size_x = 720
frame_size_y = 480
difficulty = 25

def setup():
    difficulty = 25

    pygame.init()
    pygame.display.set_caption('Snake')
    game_window = pygame.display.set_mode((frame_size_x, frame_size_y))

    snake_pos = [70, 70]
    snake_body = [[70, 70], [70-10, 70], [70-20, 70]]

    food_pos = [random.randrange(1, (frame_size_x//10)) * 10, random.randrange(1, (frame_size_y/
    food_spawn = True

    score = 0
    return snake_pos, snake_body, food_pos, food_spawn, score, difficulty, game_window

def is_obstacle(x, y, win):
    if x < 0 or x >= frame_size_x or y < 0 or y >= frame_size_y:
        return [1]

    color = win.get_at((x, y)[:3])
    if color == green:
        return [1]
    elif color == black:
        return [0]
    else:
        return [-1]

def dist(x1, y1, x2, y2):
    return math.sqrt((y2-y1)**2 + (x2-x1)**2)

def show_score(score, choice, color, font, size, game_window):
    score_font = pygame.font.SysFont(font, size)
    score_surface = score_font.render('Score : ' + str(score), True, color)
    score_rect = score_surface.get_rect()
    if choice == 1:
        score_rect.midtop = (frame_size_x/10, 15)
    else:
        score_rect.midtop = (frame_size_x/2, frame_size_y/1.25)
    game_window.blit(score_surface, score_rect)
```

Part 2: Game Body

In this part, we use the key events present in the KEYDOWN class of Pygame. The events that are used over here are, K_UP, K_DOWN, K_LEFT, and K_RIGHT to make the snake move up, down, left and right respectively. Moreover, we add some food for the snake. If the snake touches itself, then the game will over and start a new game.

To train the snake, a timer is used which ticked down every time it moved.If the snake ran out of moves, it was penalized. The fitness funtion is set as followed.If the snake died, it lost fitness.If the snake got food, it got +1 fitness for every food. The inputs the snake knows are in front, behind, left, right of its head and the direction of food in x and y.If food is on the right of head, the number would be 1. If the food is on the left of head, the number would be -1. If the food is in the same row with the head, the number would be 0. Same rule is applied to y.

In [2]:

```
def play_game(snake_pos, snake_body, food_pos, food_spawn, score, diff, game_window, genomes, co
    best_score = 0
    best_net = None
    global difficulty
    for _, g in genomes:
        snake_pos, snake_body, food_pos, food_spawn, score, _, game_window = setup()

        g.fitness = 0
        net = neat.nn.FeedForwardNetwork.create(g, config)

        direction = 'RIGHT'
        change_to = direction
        fps_controller = pygame.time.Clock()
        while True:
            board_state = []
            alive = True
            for event in pygame.event.get():
                if event.type == pygame.QUIT:
                    pygame.quit()
                    sys.exit()
                # Whenever a key is pressed down
                elif event.type == pygame.KEYDOWN:
                    # W -> Up; S -> Down; A -> Left; D -> Right
                    if event.key == pygame.K_UP or event.key == ord('w'):
                        difficulty = 10000
                    if event.key == pygame.K_DOWN or event.key == ord('s'):
                        difficulty = 120
                    if event.key == pygame.K_LEFT or event.key == ord('a'):
                        difficulty = 40
                    if event.key == pygame.K_RIGHT or event.key == ord('d'):
                        difficulty = 5000
                    # Esc -> Create event to quit the game
                    if event.key == pygame.K_ESCAPE:
                        pygame.event.post(pygame.event.Event(pygame.QUIT))

            # Snake body growing mechanism
            snake_body.insert(0, list(snake_pos))
            if snake_pos[0] == food_pos[0] and snake_pos[1] == food_pos[1]:
                score += 1
                food_spawn = False
            else:
                snake_body.pop()

            # Spawning food on the screen
            if not food_spawn:
                food_pos = [random.randrange(1, (frame_size_x//10)) * 10, random.randrange(1, (fi
            food_spawn = True

            # GFX
            game_window.fill(black)
            for pos in snake_body:
                pygame.draw.rect(game_window, green, pygame.Rect(pos[0], pos[1], 10, 10))

            # Snake food
            pygame.draw.rect(game_window, white, pygame.Rect(food_pos[0], food_pos[1], 10, 10))


            board_info = []
            direction_one_hot = [0 for i in range(len(directions_to_num))]
            direction_one_hot[directions_to_num[direction]] = 1

            board_info += is_obstacle(snake_pos[0] - 10, snake_pos[1], game_window)
            board_info += is_obstacle(snake_pos[0], snake_pos[1] - 10, game_window)
```

```
            board_info += is_obstacle(snake_pos[0] + 10, snake_pos[1], game_window)
            board_info += is_obstacle(snake_pos[0], snake_pos[1] + 10, game_window)

            angle = math.atan2((food_pos[1] - snake_pos[1]), (food_pos[0] - snake_pos[0]))
            food_x_dir = food_pos[0] - snake_pos[0]
            if food_x_dir > 0:
                inputx_dir = [1]
            elif food_x_dir < 0:
                inputx_dir = [-1]
            else:
                inputx_dir = [0]
            food_y_dir = food_pos[1] - snake_pos[1]
            if food_y_dir > 0:
                inputy_dir = [1]
            elif food_y_dir < 0:
                inputy_dir = [-1]
            else:
                inputy_dir = [0]

            inputs = []
            inputs += board_info # direction one hot vector
            inputs += inputx_dir
            inputs += inputy_dir
            inputs = tuple(inputs)
            outputs = net.activate(inputs)

            if max(outputs) > 0.5:
                change_to = directions[outputs.index(max(outputs))]

            # Making sure the snake cannot move in the opposite direction instantaneously
            if change_to == 'UP' and direction != 'DOWN':
                direction = 'UP'
            if change_to == 'DOWN' and direction != 'UP':
                direction = 'DOWN'
            if change_to == 'LEFT' and direction != 'RIGHT':
                direction = 'LEFT'
            if change_to == 'RIGHT' and direction != 'LEFT':
                direction = 'RIGHT'

            # Moving the snake
            if direction == 'UP':
                snake_pos[1] -= 10
            if direction == 'DOWN':
                snake_pos[1] += 10
            if direction == 'LEFT':
                snake_pos[0] -= 10
            if direction == 'RIGHT':
                snake_pos[0] += 10

            if snake_pos[0] < 0 or snake_pos[0] > frame_size_x-10:
                alive = False
            elif snake_pos[1] < 0 or snake_pos[1] > frame_size_y-10:
                alive = False

            # Touching the snake body
            for block in snake_body[1:]:
                if snake_pos[0] == block[0] and snake_pos[1] == block[1]:
                    alive = False

            if not alive:
                break
            show_score(score, 1, white, 'consolas', 20, game_window)
            # Refresh game screen
            pygame.display.update()
```

```
            # Refresh rate
            fps_controller.tick(difficulty)

def run(config_path):
    config = neat.config.Config(neat.DefaultGenome, neat.DefaultReproduction,
                                neat.DefaultSpeciesSet, neat.DefaultStagnation, config_path)
    p = neat.Population(config)
    p.add_reporter(neat.StdOutReporter(True))
    p.add_reporter(neat.StatisticsReporter())

    winner = p.run(main, 300)
    with open("best.pickle", "wb") as f:
        pickle.dump(winner, f)

def main(genomes, config):
    play_game(*setup(), genomes, config)

def load_pickle(pickle_filename, config_path):
    if os.path.getsize(pickle_filename) > 0:
        with open(pickle_filename, "rb") as f:
            genome = pickle.load(f)
    else:
        sys.exit(1)
    config = neat.config.Config(neat.DefaultGenome, neat.DefaultReproduction, neat.DefaultSpecies
    genomes = [(1, genome)]
    play_game(*setup(), genomes, config)
```

Part 3: Run the Game

In [ ]:

```
if __name__ == "__main__":
    local_dir = os.path.abspath('')
    config_path = os.path.join(local_dir, "config-feedforward.txt")
    load_pickle("best_89.pickle", config_path)
```

3. Conclusion

The link of the demostration video is as followed.YouTube link: https://youtu.be/0gxr4NYpmso (https://youtu.be/0gxr4NYpmso)

Although the performance is pretty well so far, we could further improve the model. One method would be to add checkpoints to training. Currently to speed up training, only 15x15 square grid (150 px by 150 px) is tested by now. We also need rigorous mathematical design for the fitness function to make the snake perform better.