SNAKE AI GAME

Group Members:

NAME	MATRIC NUMBER
Amrita Nambiar	17173025/1
Eng Kah Hui	17184031/1
Athar Bagunaid	17083073/1
Dong Yirui	17102165/1

TABLE OF CONTENTS

Problem Definition

1. Simple strategies may keep the snake alive, however, without it making efficient decisions to move toward the apples, it would not be able to perform well.

2. The application of genetic algorithm and neural network in developing a controller with artificial intelligence.
    a. Both of the algorithms are effective for AI gaming

3. The achievement of highest score possible for game by AI gaming techniques.
    a. E.g. In this game, it is to achieve as high score as possible while keeping the snake alive.

Related Work

- Best First Search (Sharma et al., 2019)

  This Greedy Best-First Search algorithm has a one-move horizon and only considers moving the snake to the position on the board that appears to be closest to the goal, i.e. apple. They use Manhattan distance to define how close the snake head is to the apple.

  This method has almost guaranteed that the snake will be able to eat in an optimal (shortest) way at least the first four apples. However, the one-step horizon also makes it easy to get stuck on local minima and plateaus.

Related Work

- A* search (Sharma et al., 2019)

  A* incorporates a heuristic in a multiple move horizon. Before taking action, it considers not only where the goal is and how far it is, but also the current state it has searched so far.

  This A* algorithm uses the Manhattan distance from the head to the apple as a heuristic and the number of steps as the “cost so far”. Each iteration of the algorithm lasts until a path is found that leads the snake to eat an apple. It improves the Best First Search algorithm by finding a full path to the apple and not stopping at the first move, this has the advantage of not getting stuck at a dead end on the way to the apple.

  The algorithm is guaranteed to find an optimal path to the apple if one exists, only when there is no memory or time restrictions.

Related Work

- Reinforcement Learning (Deep Q-Learning) (Almalki and Wocjan, 2019)

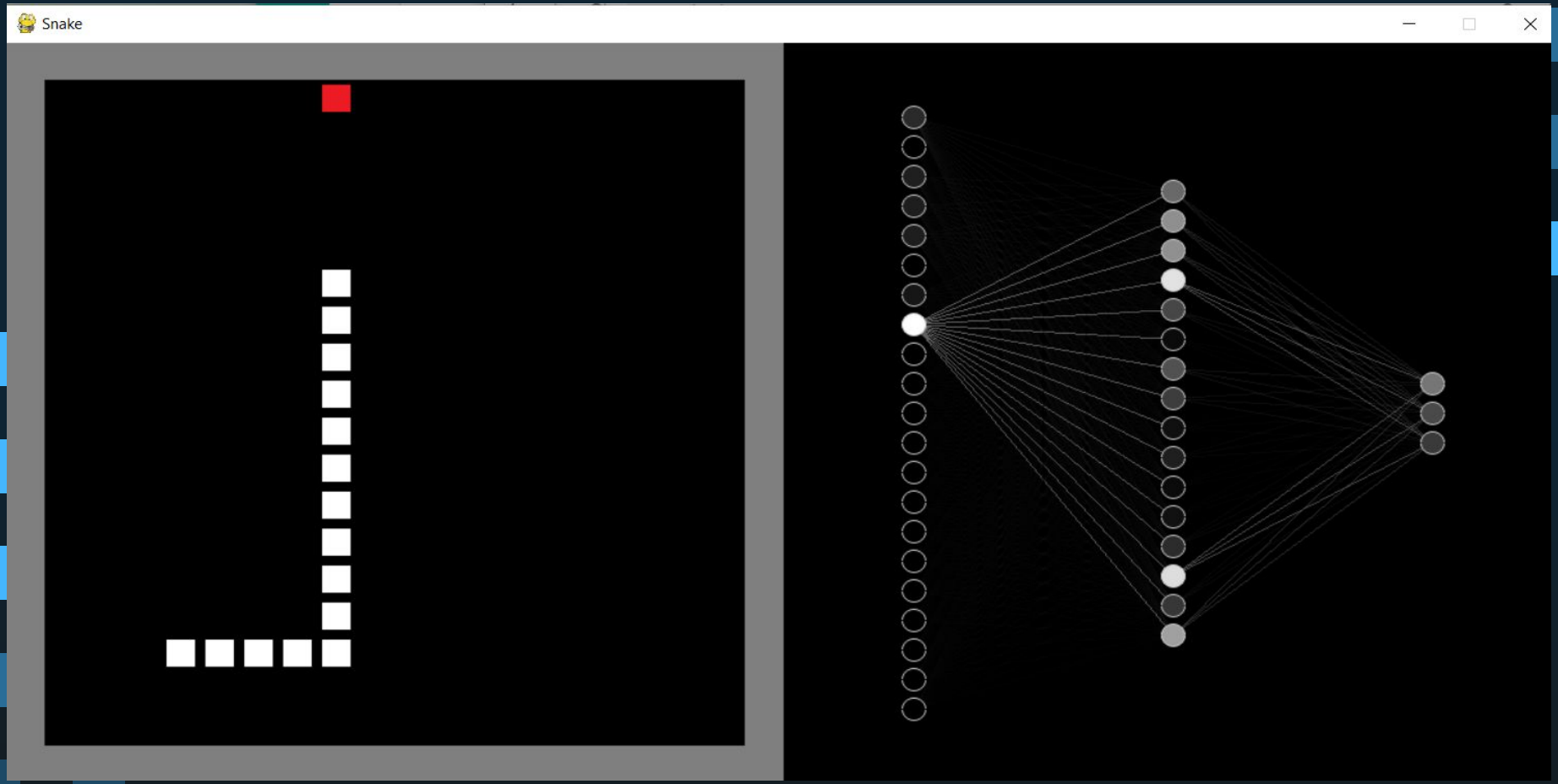
  The Deep Reinforcement Learning model enables the autonomous agent to play the Snake Game. In Reinforcement Learning, there are two main components: the environment(game) and the agent(snake). Every time the agent performs an action, the environment gives a reward to the agent, which can be positive or negative depending on how good the action was from that specific state. The goal of the agent is to learn what actions maximize the reward, given every possible state.

  However, the naive reward mechanism of DQN only produces sparse and delayed rewards that may lead to ineffective learning of correct policies.

Environment & Simulation

- The game environment is a 20x20 map
- There are boundaries with left, right, top and bottom
  - The snake has to move up and down from right to the left without touching the boundaries
  - When the snake is close to the leftmost boundary, it will go back to rightmost boundary along last row
  - The process repeats until it is out of options
- The snake has to avoid hitting its own body or the wall
- The snake has to eat apples
- The length of snake increases as it eats

Simulation



Proposed Approach

- Using a neural network & genetic algorithm, each population of snakes will slowly learn how to play the game
- The neural network has:

  - 1 input layer: Inputs for the network (Snake's vision)

  - 1 hidden layer: Learning layer

  - 1 output layer: Moving direction of snake (Straight, Left, Right)
- Takes an input vector and calculate the output by propagation through the network
- Purpose of Genetic Algorithm: Train the network

Proposed Approach

- Steps at each generation (*generation_number=100*):

  1- Parents selection

  2- Offsprings production

  3- Mutated individuals production

  4- Evaluation of whole population (old population + offsprings + mutated individuals)

  5- Additional mutations on random individuals

  6- Keeping only *population_size=1000* individuals, throwing bad performers

Initial Results

```
Best Fitness gen 1  :  38900
Pop size =  1000
Average top 6 =  22383
Average last 6 =  65

Best Fitness gen 2  :  110677
Pop size =  1000
Average top 6 =  44579
Average last 6 =  92

Best Fitness gen 3  :  121633
Pop size =  1000
Average top 6 =  56572
Average last 6 =  262

Best Fitness gen 4  :  225313
Pop size =  1000
Average top 6 =  132219
Average last 6 =  4068

Best Fitness gen 5  :  210881
Pop size =  1000
Average top 6 =  150841
Average last 6 =  6336
```

```
Best Fitness gen 6  :  767132
Pop size =  1000
Average top 6 =  332720
Average last 6 =  8000

Best Fitness gen 7  :  395558
Pop size =  1000
Average top 6 =  307593
Average last 6 =  8553

Best Fitness gen 8  :  598707
Pop size =  1000
Average top 6 =  537867
Average last 6 =  9800
```

```
Best Fitness gen 9  :  5610903
Pop size =  1000
Average top 6 =  1599111
Average last 6 =  11456

Best Fitness gen 10  :  3911038
Pop size =  1000
Average top 6 =  1922209
Average last 6 =  12962
```

Possible Experiments

- We try to play with the parameter values in order to achieve different results. However, due to the long training time, we decided to focus on the changing the crossover and mutation rates.
- Initially, the generations were trained using very low crossover rate 0.3 and very high mutation rate 0.7. However, this result in more random generation instead of consistent development from one generation to another.
- As a result we decided to train our generations with crossover rate of 0.7 and mutation rate of 0.01.
- The following slides will show the result of fitness of our generations

Our Results

```
Best Fitness gen 1  :  443989
Pop size =  1000
Average top 6 =  92310
Average last 6 =  64

Best Fitness gen 2  :  330152
Pop size =  1000
Average top 6 =  93567
Average last 6 =  116

Best Fitness gen 3  :  484716
Pop size =  1000
Average top 6 =  203691
Average last 6 =  2062

Best Fitness gen 4  :  380012
Pop size =  1000
Average top 6 =  255152
Average last 6 =  6009
```

```
Best Fitness gen 5  :  500843
Pop size =  1000
Average top 6 =  308847
Average last 6 =  7270

Best Fitness gen 6  :  1335253
Pop size =  1000
Average top 6 =  657421
Average last 6 =  8303

Best Fitness gen 7  :  1621408
Pop size =  1000
Average top 6 =  970619
Average last 6 =  10326

Best Fitness gen 8  :  1205009
Pop size =  1000
Average top 6 =  924500
Average last 6 =  15872

Best Fitness gen 9  :  2846454
Pop size =  1000
Average top 6 =  2126558
Average last 6 =  32173
```

Our Results

```
Best Fitness gen 10  :  2989713
Pop size =  1000
Average top 6 =  2489287
Average last 6 =  51198

Best Fitness gen 11  :  3905988
Pop size =  1000
Average top 6 =  3432127
Average last 6 =  96054

Best Fitness gen 12  :  3910242
Pop size =  1000
Average top 6 =  3676865
Average last 6 =  183462

Best Fitness gen 13  :  5348483
Pop size =  1000
Average top 6 =  4040871
Average last 6 =  289165

Best Fitness gen 14  :  5828977
Pop size =  1000
Average top 6 =  4401891
Average last 6 =  397007
```

```
Best Fitness gen 15  :  4775270
Pop size =  1000
Average top 6 =  4113808
Average last 6 =  545912

Best Fitness gen 16  :  4745472
Pop size =  1000
Average top 6 =  4304272
Average last 6 =  943951

Best Fitness gen 17  :  6715071
Pop size =  1000
Average top 6 =  4939508
Average last 6 =  1487621
pygame 2.0.1 (SDL 2.0.14, Python 3.6.9)
Hello from the pygame community. https:/
pygame 2.0.1 (SDL 2.0.14, Python 3.6.9)
Hello from the pygame community. https:/
```

Our Results

```
Best Fitness gen 18  :  5127162
Pop size =  1000
Average top 6 =  4623961
Average last 6 =  1884860
pygame 2.0.1 (SDL 2.0.14, Python 3.6.9)
Hello from the pygame community. https:/
pygame 2.0.1 (SDL 2.0.14, Python 3.6.9)
Hello from the pygame community. https:/

Best Fitness gen 19  :  5693596
Pop size =  1000
Average top 6 =  4504226
Average last 6 =  2258151
pygame 2.0.1 (SDL 2.0.14, Python 3.6.9)
Hello from the pygame community. https:/
pygame 2.0.1 (SDL 2.0.14, Python 3.6.9)
Hello from the pygame community. https:/

Best Fitness gen 20  :  5285272
Pop size =  1000
Average top 6 =  4759569
Average last 6 =  2495822
```

```
Best Fitness gen 21  :  4839123
Pop size =  1000
Average top 6 =  4544186
Average last 6 =  2620212
pygame 2.0.1 (SDL 2.0.14, Python 3.6.9)
Hello from the pygame community. https://w
pygame 2.0.1 (SDL 2.0.14, Python 3.6.9)
Hello from the pygame community. https://w

Best Fitness gen 22  :  5350754
Pop size =  1000
Average top 6 =  4735811
Average last 6 =  2702371
pygame 2.0.1 (SDL 2.0.14, Python 3.6.9)
Hello from the pygame community. https://w
pygame 2.0.1 (SDL 2.0.14, Python 3.6.9)
Hello from the pygame community. https://w

Best Fitness gen 23  :  5060487
Pop size =  1000
Average top 6 =  4744628
Average last 6 =  2766833
pygame 2.0.1 (SDL 2.0.14, Python 3.6.9)
Hello from the pygame community. https://w
pygame 2.0.1 (SDL 2.0.14, Python 3.6.9)
Hello from the pygame community. https://w

Best Fitness gen 24  :  7002003
Pop size =  1000
Average top 6 =  5194164
Average last 6 =  2787661
pygame 2.0.1 (SDL 2.0.14, Python 3.6.9)
```

Our Results

```
Best Fitness gen 25  :  5137412
Pop size =  1000
Average top 6 =  4864607
Average last 6 =  2856654
pygame 2.0.1 (SDL 2.0.14, Python 3.6.9)
Hello from the pygame community. https://
pygame 2.0.1 (SDL 2.0.14, Python 3.6.9)
Hello from the pygame community. https://

Best Fitness gen 26  :  4752597
Pop size =  1000
Average top 6 =  4679838
Average last 6 =  2865873
pygame 2.0.1 (SDL 2.0.14, Python 3.6.9)
Hello from the pygame community. https://
pygame 2.0.1 (SDL 2.0.14, Python 3.6.9)
Hello from the pygame community. https://

Best Fitness gen 27  :  5212924
Pop size =  1000
Average top 6 =  4833892
Average last 6 =  2905010
pygame 2.0.1 (SDL 2.0.14, Python 3.6.9)
Hello from the pygame community. https://
pygame 2.0.1 (SDL 2.0.14, Python 3.6.9)
Hello from the pygame community. https://

Best Fitness gen 28  :  5475419
Pop size =  1000
Average top 6 =  5169465
Average last 6 =  2940077
```

```
Best Fitness gen 29  :  5173476
Pop size =  1000
Average top 6 =  4863881
Average last 6 =  2957973
pygame 2.0.1 (SDL 2.0.14, Python 3.
Hello from the pygame community. ht
pygame 2.0.1 (SDL 2.0.14, Python 3.
Hello from the pygame community. ht

Best Fitness gen 30  :  5520826
Pop size =  1000
Average top 6 =  5168935
Average last 6 =  3015475
pygame 2.0.1 (SDL 2.0.14, Python 3.
Hello from the pygame community. ht
pygame 2.0.1 (SDL 2.0.14, Python 3.
Hello from the pygame community. ht

Best Fitness gen 31  :  5680236
Pop size =  1000
Average top 6 =  5233893
Average last 6 =  3007817
pygame 2.0.1 (SDL 2.0.14, Python 3.
Hello from the pygame community. ht
pygame 2.0.1 (SDL 2.0.14, Python 3.
Hello from the pygame community. ht

Best Fitness gen 32  :  5567974
Pop size =  1000
Average top 6 =  5266513
Average last 6 =  3066903
```

Our Results

```
Best Fitness gen 33  :  5434724
Pop size =  1000
Average top 6 =  5021136
Average last 6 =  3084705
pygame 2.0.1 (SDL 2.0.14, Python 3.6.9
Hello from the pygame community. https
pygame 2.0.1 (SDL 2.0.14, Python 3.6.9
Hello from the pygame community. https

Best Fitness gen 34  :  6798797
Pop size =  1000
Average top 6 =  5456648
Average last 6 =  3101663
pygame 2.0.1 (SDL 2.0.14, Python 3.6.9
Hello from the pygame community. https
pygame 2.0.1 (SDL 2.0.14, Python 3.6.9
Hello from the pygame community. https

Best Fitness gen 35  :  5114702
Pop size =  1000
Average top 6 =  4954652
Average last 6 =  3121305
pygame 2.0.1 (SDL 2.0.14, Python 3.6.9
Hello from the pygame community. https
pygame 2.0.1 (SDL 2.0.14, Python 3.6.9
Hello from the pygame community. https

Best Fitness gen 36  :  5886957
Pop size =  1000
Average top 6 =  5533286
Average last 6 =  3102758
```

```
Best Fitness gen 35  :  5114702
Pop size =  1000
Average top 6 =  4954652
Average last 6 =  3121305
pygame 2.0.1 (SDL 2.0.14, Python 3.6.9)
Hello from the pygame community. https://www.py
pygame 2.0.1 (SDL 2.0.14, Python 3.6.9)
Hello from the pygame community. https://www.py

Best Fitness gen 36  :  5886957
Pop size =  1000
Average top 6 =  5533286
Average last 6 =  3102758
pygame 2.0.1 (SDL 2.0.14, Python 3.6.9)
Hello from the pygame community. https://www.py
pygame 2.0.1 (SDL 2.0.14, Python 3.6.9)
Hello from the pygame community. https://www.py

Best Fitness gen 37  :  6955310
Pop size =  1000
Average top 6 =  5460108
Average last 6 =  3079386
pygame 2.0.1 (SDL 2.0.14, Python 3.6.9)
Hello from the pygame community. https://www.py
pygame 2.0.1 (SDL 2.0.14, Python 3.6.9)
Hello from the pygame community. https://www.py
```

THANKS

CREDITS: This presentation template was created by **Slidesgo**, including icons by **Flaticon**, and infographics & images by **Freepik** and illustrations by **Stories**