

R V COLLEGE OF ENGINEERING

(Affiliated to Visvesvaraya Technological University, Belagavi)



An Assignment Report on

“Snake Game with Genetic Algorithm”

Submitted in partial fulfillment for Assignment component of

**Advances in Algorithms
18MCS2D2**

of

**MASTER OF TECHNOLOGY
in
COMPUTER NETWORK ENGINEERING**

Submitted by
**AKASH HEGDE
1RV18SCN01**

Under the guidance of
**PRAVEENA T.
ASSISTANT PROFESSOR
R.V. COLLEGE OF ENGINEERING**

Department of Computer Science and Engineering

R V COLLEGE OF ENGINEERING
R.V.VIDYANIKETAN POST, MYSURU ROAD,
BENGALURU- 560059

2018-19

ACKNOWLEDGEMENT

The satisfaction and euphoria that accompany the successful completion of any task would be incomplete without the mention of the people who made it possible, whose constant guidance and encouragement crowned the efforts with success.

It is indeed a great pleasure for me to present this self-study report on “Snake Game with Genetic Algorithm” as a part of the course Advances in Algorithms in the 2nd semester of Master of Technology in Computer Network Engineering.

I would like to thank Management of R V College of Engineering for providing such a healthy environment for the successful completion of self-study work.

It gives me immense pleasure to thank Dr. Ramakanth Kumar P, Professor and Head of Department for his constant support and encouragement.

Also, I would like to express my gratitude to my course guide Praveena T., Assistant Professor, Department of Computer Science & Engineering and all other teaching and non-teaching staff of Computer Science Department who have directly or indirectly helped me in the completion of the self-study work.

Last, but not the least, I would hereby acknowledge and thank my parents who have been a source of inspiration and also instrumental in the successful completion of the self-study work.

Akash Hegde
Dept. of CNE

List of Figures

Sl.No.	Figure Name	Page No.
1.1	Steps of a genetic algorithm	2
1.2	Example of an initial population	3
1.3	Example of a crossover	4
1.4	Example of a mutation	4
2.1	Snake Game in progress	7
2.2	Display of score board	7
3.1	Neural network architecture	9
3.2	Training Snake Game in progress	10
3.3	Output file with best fitness chromosomes	11

Table of Contents

Acknowledgement	(i)
List of Tables and Figures	(ii)
Chapter 1 1. Introduction 1.1 Introduction to Snake Game 1.2 Introduction to Genetic Algorithm	1-4
Chapter 2 2. Creating a Snake Game	5-7
Chapter 3 3. Training Snake Game with Genetic Algorithm	8-11
Chapter 4 4. Conclusion and Future Scope	12

Chapter 1

Introduction

1.1 Introduction to Snake Game

Snake is the common name for a video game concept where the player manoeuvres a line which grows in length, with the line itself being a primary obstacle. The concept originated in the 1976 arcade game Blockade, and the ease of implementing Snake has led to hundreds of versions (some of which have the word snake or worm in the title) for many platforms. After a variant was preloaded on Nokia mobile phones in 1998, there was a resurgence of interest in the snake concept as it found a larger audience. There are over 300 Snake-like games for iOS alone.

The gameplay of the Snake Game is pretty simple. The player controls a dot, square, or object on a bordered plane. As it moves forward, it leaves a trail behind, resembling a moving snake. In some games, the end of the trail is in a fixed position, so the snake continually gets longer as it moves. In another common scheme, the snake has a specific length, so there is a moving tail a fixed number of units away from the head. The player loses when the snake runs into the screen border, a trail or other obstacle, or itself.

1.2 Introduction to Genetic Algorithm

A genetic algorithm (GA) is a metaheuristic inspired by the process of natural selection that belongs to the larger class of evolutionary algorithms (EA). Genetic algorithms are commonly used to generate high-quality solutions to optimization and search problems by relying on bio-inspired operators such as mutation, crossover and selection. John Holland introduced genetic algorithms in 1960 based on the concept of Darwin's theory of evolution; afterwards, his student David E. Goldberg extended GA in 1989.

A typical genetic algorithm requires some population in the solution domain and a fitness function to find the fittest individual. To evolve individuals in the population genetic algorithm uses some operations like crossover, mutation, and selection.

Genetic algorithm starts with some random initial population. It then tries to produce offspring from the best individuals in the population. The concept is that, if the fittest individuals are selected then the chances of producing a better offspring are more. This process keeps on iterating until the target is not achieved. Each iteration is known as a *generation*. Figure 1.1 represents the general steps of a genetic algorithm.

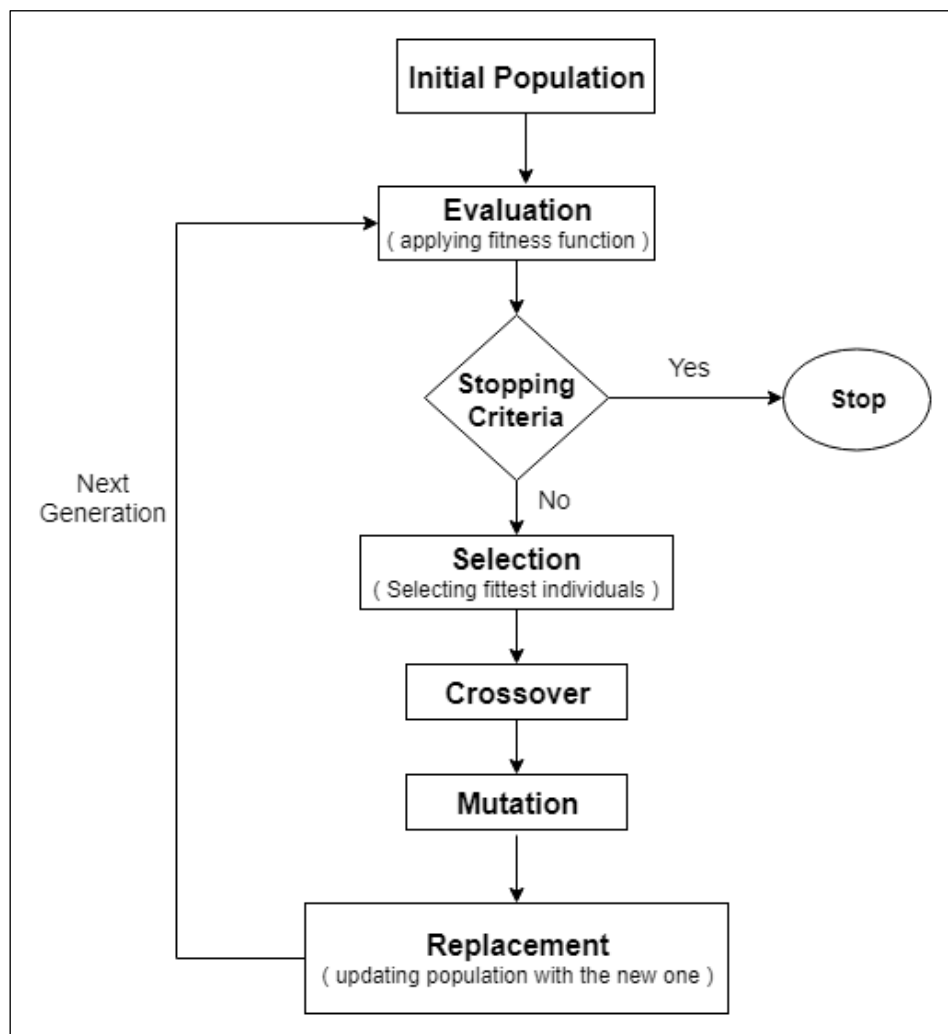


Figure 1.1 Steps of a genetic algorithm

Initial Population –

Initial Population refers to a set of possible solutions. Each member (individual) of the population is usually known as the chromosome (phenotypes) and represents a solution for the problem to be investigated. The chromosome is represented as a set of parameters (features or genes or weights) that defines the individual. Size of the population depends on the problem. Random selection of initial population makes sure that it covers a wide range of possible solution. Figure 1.2 represents an example of an initial population in genetic algorithm.

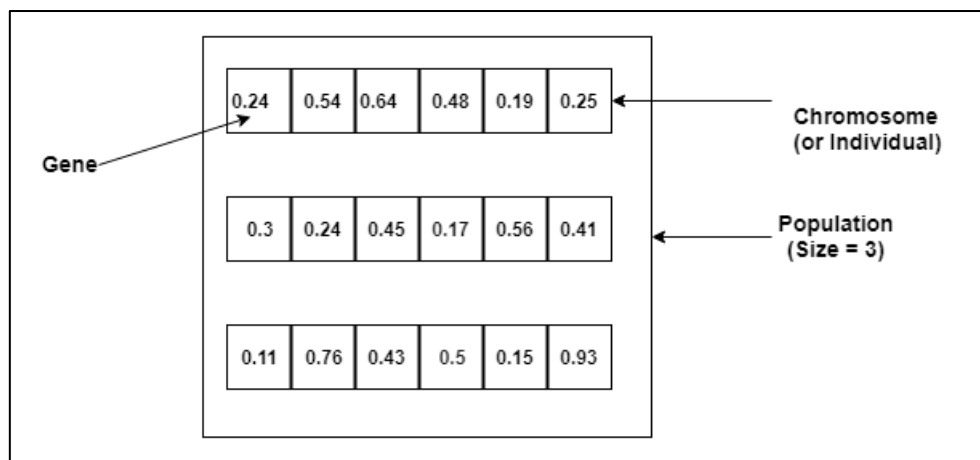


Figure 1.2 Example of an initial population

Evaluation and Fitness Function –

Once a random initial population is generated, the next thing is to evaluate the fitness of these individuals. To evaluate the fitness of these individuals, some fitness function has to be defined. The fitness function has to be chosen according to the problem. Fitness function measures the quality of each individual.

Selection –

Some best individuals are selected from the evaluated population. These selected individuals are mated to produce some new offspring.

Crossover –

Each individual selected in the previous step has some quality. The objective is to produce better offspring so that the algorithm can evolve and find a better solution to the problem. To do that, two individuals from the best population are selected and a

new child (offspring) is produced with features of both as shown in Figure 1.3. This is known as Crossover.

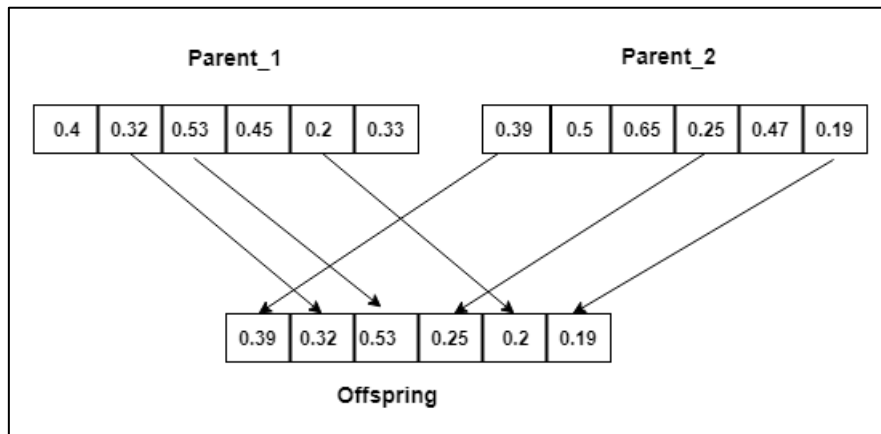


Figure 1.3 Example of a crossover

Mutation –

Mutation is applied to maintain the diversity within the population and inhibit premature convergence. With some low probability, a portion of the new individual is subjected to mutation as shown in Figure 1.4.

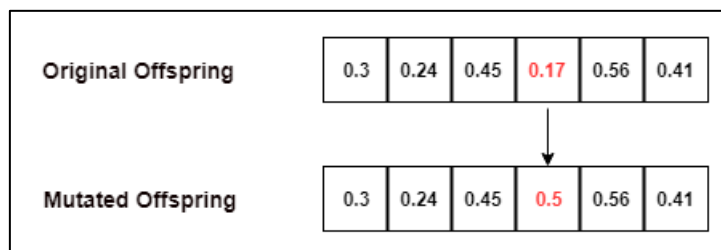


Figure 1.4 Example of a mutation

Replacement –

New population replaces a previous one for the next generation. This process keeps on iterating until a certain target is not achieved.

Genetic algorithms are applied to train neural networks instead of techniques like gradient descent or Adam. They are also used to select the neural network architecture with respect to the hyperparameters.

Chapter 2

Creating a Snake Game

The Snake Game is built using pygame. Pygame is a free and open source Python programming language library used for making multimedia applications like games. It is great for beginners as it is simple and easy to use. Pygame can be installed to an existing Python version either by using pip or using a wheel from the Python extension packages.

First, the pygame modules have to be initialized using a `pygame.init()` command. `pygame.init()` will attempt to initialize all the pygame modules. Not all pygame modules need to be initialized, but this will automatically initialize the ones that do.

After initializing pygame modules, the game window itself has to be displayed. For that, its dimensions (width and height) have to be initialized. They are set as 500 x 500 pixels.

The function `display.fill(window_color)` fills white color into game window and `pygame.display.update()` allows only a portion of the screen to be updated, instead of the entire area. If no argument is passed, it updates the entire surface area. This creates a game window.

Then, the snake and apple have to be displayed in the game window. Green colour has been used for the snake and a sprite image for the apple. At the start of each game, the starting position of the snake has to be fixed while the apple can take any random location. Starting length of the snake is 3 units, where each unit is a 10×10 block.

The function `pygame.draw.rect()` will draw a rectangle corresponding to given arguments which will represent the snake and `display.blit()` will show the image of an apple.

The next thing is to decide at what frame rate the game has to be played. This is done using `pygame.time.Clock()` that creates a “clock” object. Whenever `clock.tick()` is called and passed with some number, it will limit the run time speed of the game. As an example, if `clock.tick(30)` is called, then the program will never run at more than 30 frames per second.

The logic of the Snake Game defines some gameplay rules –

- If the snake collapses with itself or with boundaries, then it is “game over.”
- In the game, the snake will continue to move in one direction until a button is pressed. To move the snake, one unit has to be added to the head and one unit has to be removed from the tail.
- Another case is when a button is pressed to move in a direction (left, right, up or down). Also, the snake cannot move backward.
- After seeing which direction button is pressed, the head position of the snake is updated.
- If the snake eats an apple, the apple moves to a new position and snake length increases.
- When the snake eats an apple, the length of the snake is increased, thereby improving the score. To increase snake size, one unit is added at the head of the snake. Also, another apple has to be created at a random location to proceed the game.
- Once the game is over, the total score must be displayed.

Figures 2.1 and 2.2 depict the Snake Game in progress and the final score board respectively.

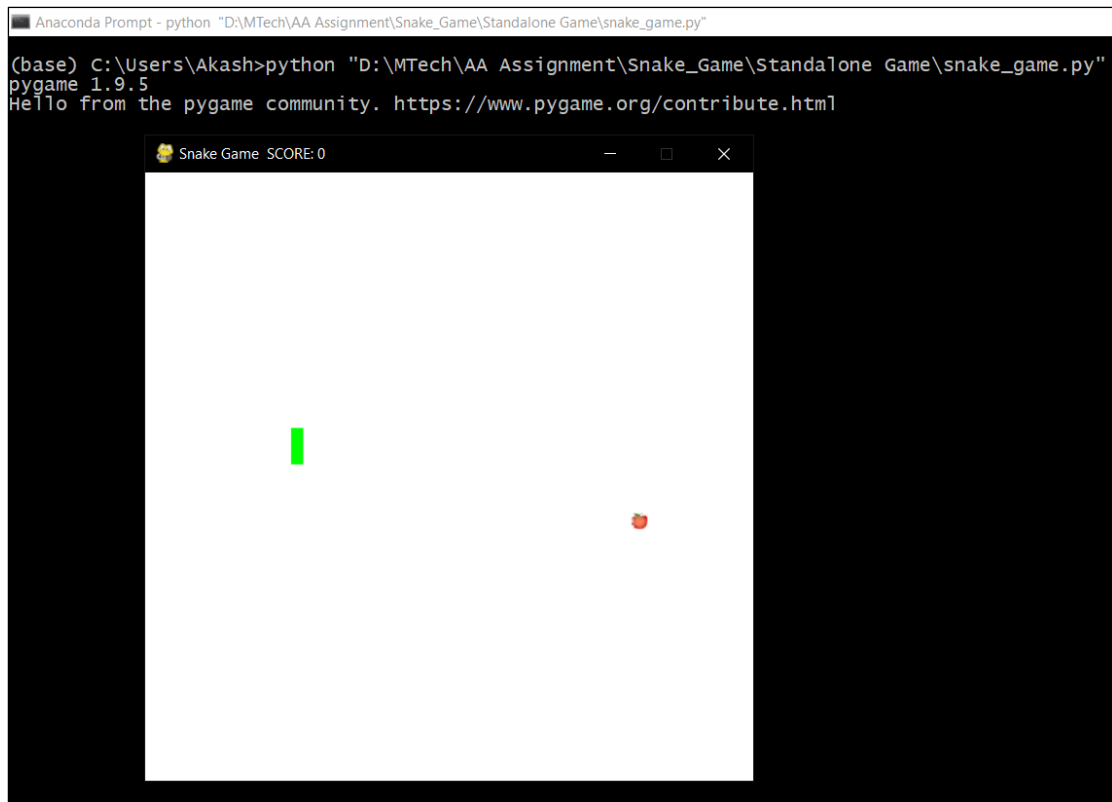


Figure 2.1 Snake Game in progress

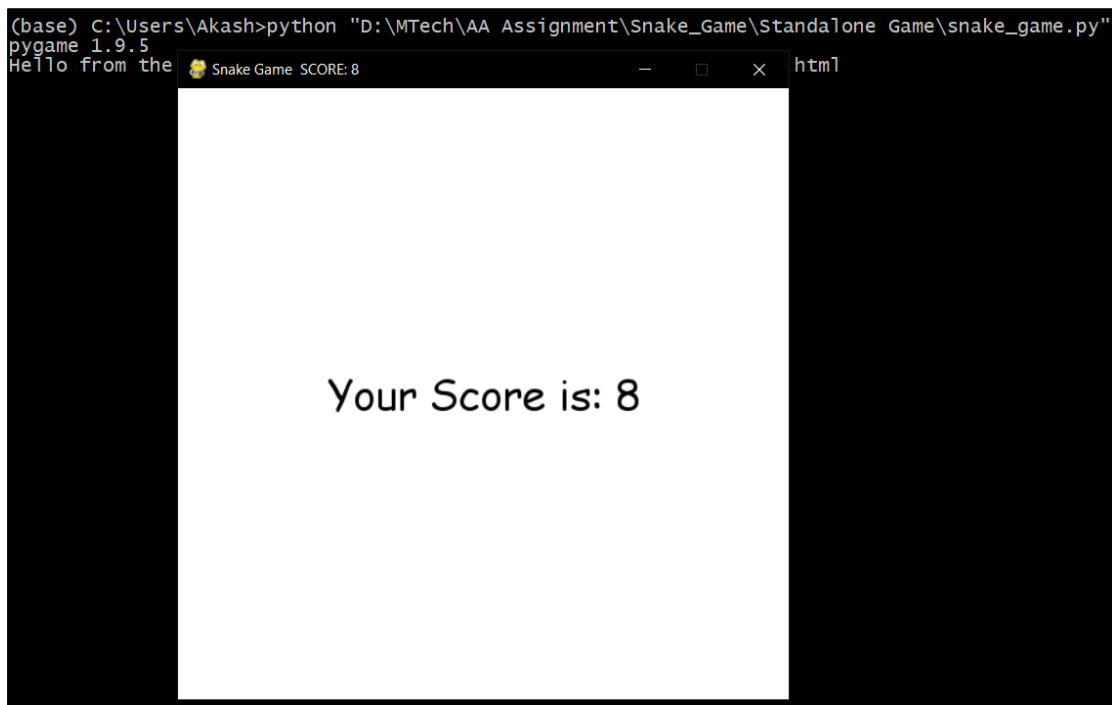


Figure 2.2 Display of score board

Chapter 3

Training Snake Game with Genetic Algorithm

The following section describes how the Snake Game is trained with a genetic algorithm (GA) and a neural network.

The main advantage of using a combination of genetic algorithm and neural network for training the Snake Game is that there is no need of any training data, unlike backpropagation where training data is necessary. Weights are updated using genetic algorithm instead of backpropagation.

The entire process of training the Snake Game with GA is given as –

- Creating a snake game and deciding neural network architecture.
- Creating an initial population.
- Deciding the fitness function.
- Play a game for each individual in the population and sort each individual in the population based on the fitness function score.
- Select a few top individuals from the population and create the remaining population from these top selected individuals using Crossover and mutation.
- The new population is created (meaning the next generation).
- Repeat until the stopping criteria are not satisfied.

Initially, the Snake Game is created with pygame and network architecture consists of 7 units in the input layer, 3 units in the output layer with ‘softmax’ and uses 2 hidden layers, one of 9 units and other of 15 units with ‘relu’ as shown in Figure 3.1.

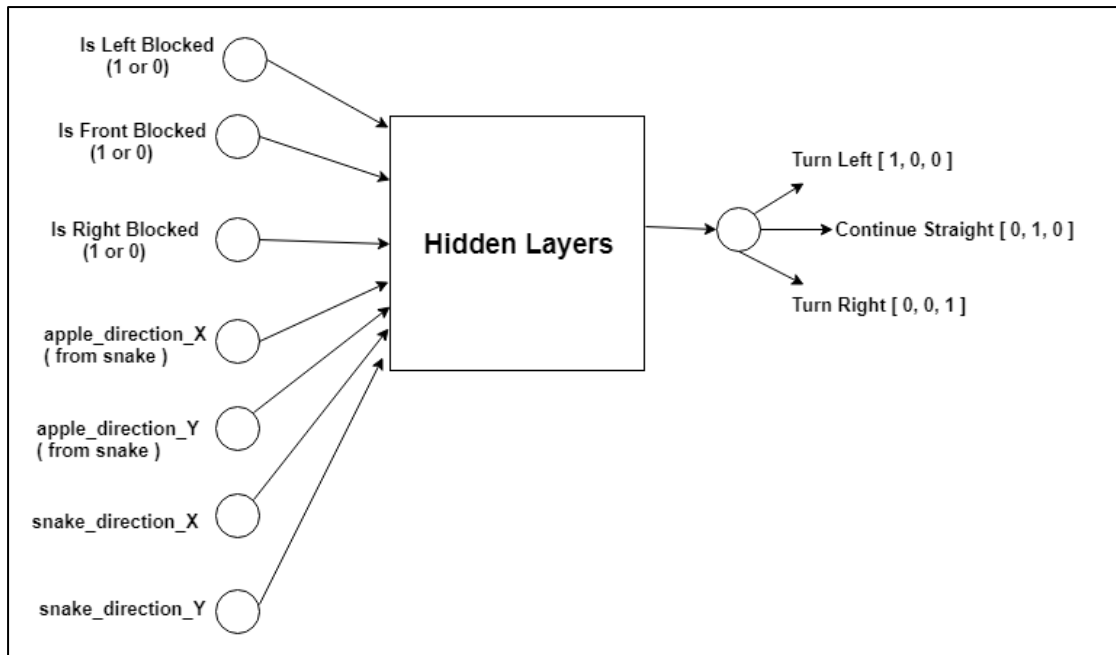


Figure 3.1 Neural network architecture

Creating Initial Population –

30 individuals are chosen in the initial population and each individual is an array of weights of the neural network. These individuals are randomly initialized.

Deciding the Fitness Function –

Any fitness function can be used. Here, the following fitness function has been used – “On every grasp of food, 5000 reward points are given and if the snake collides with the boundary or itself, a penalty of 150 points is awarded.”

Evaluating the population –

For each individual, a game is played and the fitness function is calculated which is then appended in a list.

Selection, Crossover, and Mutation –

Selection: According to fitness value, some best individuals will be selected from the population and are stored in the ‘parents’ array.

Crossover: The crossover is used to produce children for the next generation. First, two individuals are randomly selected from the best, then some values are randomly

chosen from the first and some from the second individual to produce new offspring. This process is then repeated until the total population size is not achieved.

Mutation: Then, some variations are being added to the newly formed offspring. Here, for each child, 25 weights are randomly selected and mutated by adding some random value.

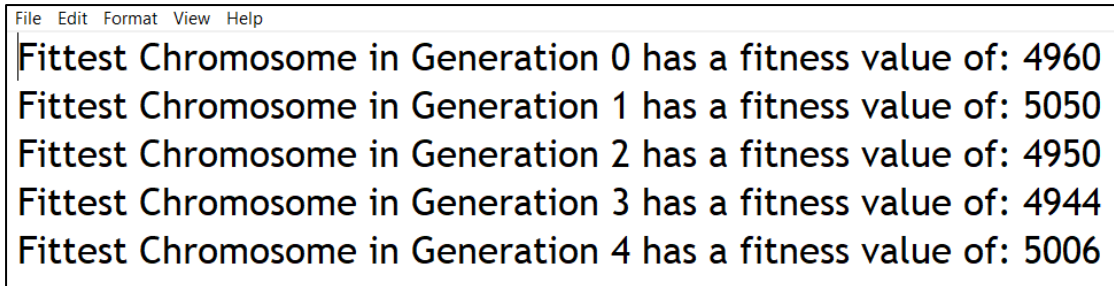
New Population Created –

With the help of fitness function, crossover and mutation, new population for the next generation is created. The previous population is then replaced with this newly formed population.

This process is repeated until a target for certain game score is not achieved. 5 generations are used for training and 2500 steps are run in each game. The best fit chromosomes are found out for each generation and stored in a text file. Figures 3.2 and 3.3 depict the training in progress and the output text file with the fitness chromosomes which have the highest value for that particular generation.



Figure 3.2 Training Snake Game in progress



```
File Edit Format View Help
Fittest Chromosome in Generation 0 has a fitness value of: 4960
Fittest Chromosome in Generation 1 has a fitness value of: 5050
Fittest Chromosome in Generation 2 has a fitness value of: 4950
Fittest Chromosome in Generation 3 has a fitness value of: 4944
Fittest Chromosome in Generation 4 has a fitness value of: 5006
```

Figure 3.3 Output file with best fitness chromosomes

The code written consists of 5 Python scripts which are given as follows –

- main.py – to start training snake game using genetic algorithm.
- Snake_Game.py – contains logic of creating snake game using pygame.
- Run_Game.py – play snake game using predicted directions from genetic algorithm.
- Genetic_Algorithm.py – contains genetic algorithm functions like crossover, mutation.
- Feed_Forward_Neural_Network.py – contains the functions for calculating the output from feed forward neural network.

Chapter 4

Conclusion and Future Scope

Snake Game is a simple game in which the player controls a dot, square, or object on a bordered plane, which is the snake itself. As it moves forward, it leaves a trail behind, resembling a moving snake. The snake grows in length whenever it comes in contact with any food and this also increases the score of the player. The player loses when the snake runs into the screen border, a trail or other obstacle, or itself.

A genetic algorithm (GA) is a metaheuristic inspired by the process of natural selection, that is used to generate high-quality solutions to optimization and search problems by relying on bio-inspired operators such as mutation, crossover and selection.

The Snake Game is trained using a combination of genetic algorithm and neural network in order to achieve a better score. The main advantage of using a combination of genetic algorithm and neural network for training the Snake Game is that there is no need of any training data. 5 generations are used to train the data and each game consists of 2500 steps, with a population of 30 individuals. The best fitness chromosomes are identified for each generation and listed down in a text file.

Future enhancements could improve the score of the Snake Game by using deep learning techniques and real-time speech recognition where in the snake could be moved using voice commands. Furthermore, the Snake Game could be created using different technologies such as OpenCV, Python Curses and Tensorflow Object Detection API.