Automated Snake Game with Genetic Algorithm

A Report for the Evaluation 3 of Capstone Project 2

Submitted By

Ankit Kumar

(1613114011 / 16SCSE114027)

in partial fulfillment for the award of the degree

of

Bachelor of Technology

IN

Computer Science and Engineering With Specialization of Computer

Networks and Cyber Security

SCHOOL OF COMPUTING SCIENCE AND ENGINEERING

Under the Supervision of

Dr. DILEEP KUMAR YADAV

Associate Professor, Dept. of CSE

APRIL / MAY (2020)

GALGOTIAS UNIVERSITY

(Established under Galgotias University Uttar Pradesh Act No. 14 of 2011)

SCHOOL OF COMPUTING SCIENCE AND ENGINEERING

BONAFIDE CERTIFICATE

Certified that this project report on **"Automated Snake Game with Genetic Algorithm"** is the bonafide work of **"ANKIT KUMAR (1613114011)"** who carried out the project work under my supervision.

SIGNATURE OF HEAD	**SIGNATURE OF SUPERVISOR**
Dr. MUNISH SHABARWAL,	Dr. DILEEP KUMAR YADAV
PhD (Management), PhD (CS)	PhD.
Professor & Dean,	**Associate Professor,**
School of Computing Science and Engineering	**School of Computing Science and Engineering**

ABSTRACT :

This project is based on the **Artificial Neural Network** (ANN) where, the Neural Network plays the snake game on its own. For this, the Neural Network must be trained with the training data, based on the training data, the weights and biases are adjusted to get the optimal solution. But to generate labeled data for training is difficult. So, to overcome this problem we can use the concept of **Genetic Algorithm** for the training of the neural network. In context with the snake game the optimal solution will be the direction in which the snake should be headed so that it can avoid obstacles and eat it's food to grow longer and longer. There can be multiple ways by which we can automate the snake to eat it's food, for eg. We can use some of the path finding algorithms like **BFS, DFS, A* searching,** etc. but these will be hardcoded decisions. They are efficient but very likely to get stuck at some places. So to overcome this we can use an Artificial Neural Network (Genetic Algorithm) based approach to find alternate path when it gets stuck. Python programming language will be used in the coding part as well as for designing the neural network. The snake game will be made with the help of a python library called **PyGame**. The neural network for the snake game will also be made with the help of python.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER - 1

Introduction

1.1) Introduction to the Snake Game

Basically Snake game consists of a environment where there are walls, snake body and food. The snake has to eat the food, and each time snake eat the food, the snake body grows by a constant factor. There are some rules that implied to snake in order to move and eat the food in the environment. The snake has to avoid its body part and the wall while eating its food or moving in the environment. Apart from this the snake can only move in 4 directions : **Left, Right, Up, and Down.** If snake hits its body or walls it will be dead and final score will be the total length of the snake.

1.2) Overall Description

This project shows how we can implement Artificial Intelligence in a classic mobile game (snake). Automated Snake game uses Genetic Algorithm to find the path from the head of the snake to the food while avoiding any such conditions which can make snake dead, i.e., avoiding contact with wall, avoiding contact with its tail. As the Genetic Algorithm is used in optimization problems, this can proved to be a very good approach for teaching an AI. The Neural Network used in this project is a 4 layer network. We can also automate the snake game using path finding algorithms like Breadth First Search, A* (star) search, etc. These algorithms guarantee to find the direct shortest path from the head to the food. If we can optimize them for conditions like when there is not having a direct path available, then the snake will be able to get higher scores.

1.3) Introduction to Machine Learning

Machine Learning is a core sub-area of Artificial Intelligence (AI). ML applications learn from experience (well data) like humans without direct programming. When exposed to new data, these applications learn, grow, change, and develop by themselves. In other words, with Machine Learning, computers find insightful information without being told where to look. Instead, they do this by leveraging algorithms that learn from data in an iterative process.

1.4) Types of Machine Learning

The important 3 types of Machine Learning are :

- SUPERVISED LEARNING
- UNSUPERVISED LEARNING
- REINFORCEMENT LEARNING

1.4.1) Supervised Learning

In supervised learning, we use known or labeled data for the training data. Since the data is known, the learning is, therefore, supervised, i.e., directed into successful execution. The input data goes through the Machine Learning algorithm and is used to train the model. Once the model is trained based on the known data, you can use unknown data into the model and get a new response.

1.4.2) Unsupervised Learning

In unsupervised learning, the training data is unknown and unlabeled - meaning that no one has looked at the data before. Without the aspect of known data, the input cannot be guided to the algorithm, which is where the unsupervised term originates from. This data is fed to the Machine

Learning algorithm and is used to train the model. The trained model tries to search for a pattern and give the desired response.

1.4.3) Reinforcement Learning

Like traditional types of data analysis, here, the algorithm discovers data through a process of trial and error and then decides what action results in higher rewards. Three major components make up reinforcement learning: the **agent**, the **environment**, and the **actions**. The agent is the learner or decision-maker, the environment includes everything that the agent interacts with, and the actions are what the agent does. Reinforcement learning occurs when the agent chooses actions that maximize the expected reward over a given time. This is easiest to achieve when the agent is working within a sound policy framework.

CHAPTER - 2

Literature Review

This project is the implementation of the snake game, that has the ability to find and eat it's food without colliding with any of the obstacles. The snake is implemented using the Artificial Neural Network, that helps the snake to take the automated decisions. The two components of a basic real-time path finding are : (i) heading in the direction of the goal and (ii) avoiding any static and dynamic obstacles that may litter the path to that goal in real-time.

2.1) Different ways of static path finding

Static path finding means the obstacles are not moving. The snake game is based on the grid system, so that we can apply the various path finding algorithms, some of them are : Breadth First Search, Depth First Search, Dijkstra's Algorithm and A* (A star) algorithm, etc. The problem with these algorithms are that they are not smart and cannot take decisions when there does not exist a direct path.

2.2) How to apply path finding algorithms in Snake Game

These algorithms are the graph based algorithms. We can make the snake environment as a grid based system, and each block can be considered as a node and it's adjacent grids and be treated as child nodes. Therefore using the snakes head as the starting grid, these algorithms can be used to find the path up to the food, if a path is available.

2.3) Depth First Search

Depth-first search (DFS) is an algorithm for traversing or searching tree or graph data structures. The algorithm starts at the root node (selecting some arbitrary node as the root node in the case of a graph) and explores as far as

possible along each branch before backtracking. Here the snake head is the root node and the goal node will be the food.

2.4) Breadth First Search

Breadth-first search (BFS) is an algorithm for traversing or searching tree or graph data structures. It starts at the tree root (or some arbitrary node of a graph, sometimes referred to as a 'search key'), and explores all of the neighbor nodes at the present depth prior to moving on to the nodes at the next depth level. It uses the opposite strategy as depth-first search, which instead explores the node branch as far as possible before being forced to backtrack and expand other nodes.

2.5) A*(star) path Search

A* is an informed search algorithm, or a best-first search, meaning that it is formulated in terms of weighted graphs: starting from a specific starting node of a graph, it aims to find a path to the given goal node having the smallest cost (least distance traveled, shortest time, etc.). It does this by maintaining a tree of paths originating at the start node and extending those paths one edge at a time until its termination criterion is satisfied.

At each iteration of its main loop, A* needs to determine which of its paths to extend. It does so based on the cost of the path and an estimate of the cost required to extend the path all the way to the goal. Specifically, A* selects the path that minimizes :

$$f(n) = g(n) + h(n).$$

where n is the next node on the path, g(n) is the cost of the path from the start node to n, and h(n) is a heuristic function that estimates the cost of the cheapest path from n to the goal. A* terminates when the path it chooses to extend is a path from start to goal or if there are no paths eligible to be

extended. The heuristic function is problem-specific. If the heuristic function is admissible, meaning that it never overestimates the actual cost to get to the goal, A* is guaranteed to return a least-cost path from start to goal.

CHAPTER - 3

Proposed Model

From past some years the Artificial Intelligence and Machine Learning have evolved a lot. Now a days these models are in use everywhere from automation to prediction, in software to IoT devices, etc. The AI have been extensively used in gaming industry for different purposes. In this project I have implemented a Genetic Algorithm based optimization, to adjust the weights and biases of the neural network that have been used in Snake Game.

3.1) Use of Genetic Algorithm and Neural Network in Path Finding

An artificial neural network is an information-processing system that has certain performance characteristics in common with biological neural networks. Each input into a neuron has a weight value associated with it these weights are the primary means of storage for neural networks. Learning takes place by changing the value of the weights. The key point is that a trained Neural Network has to ability to generalize on situations that it has never encountered. This is a particularly useful feature that should help considerably with dynamic scenes. There are many different types of neural networks but the one particularly suited to real-time games is the Feed Forward Neural Network due to the speed it can process data . The Artificial Neural Network requires training with large amount of labeled data. But it is very difficult to generate labeled training data for the snake game, therefore a simpler approach of using Genetic Algorithm is used. The population of snakes is first initialized with random Neural Networks, then each of the snakes in the population plays the game, and after all the snakes have played there games. The calculation of

Fitness function for each of the brain took place. Then some of the best performing snakes are used to generate the next population. Then crossover takes place and after that if required mutation takes place. With this process after some number of generation we will be able to fine tune the weights, that in turn will produce best performing snakes. This process can also be visualized as an optimization problem, here a large population of Snake object played there game, and among them the best performing, generates the next generation that should have higher adaptability to the environment.

CHAPTER - 4

Implementation

4.1) Software Requirements

- **Operating System** : Windows, Linux, Mac
- **Programming Language** : Python v 3.6 or above
- **Libraries** : PyGame, Numpy
- **Code Editor** : PyCharm or VS Code

4.2) Hardware Requirements

- **Processor** : Minimum 2GHz
- **Memory (RAM)** : Minimum 2GB,
  Recommended 4 GB
- **ROM** : A minimum of 10GB is
  required to install and run PyCharm smoothly
- **Monitor Resolution** : 1024x768 or higher

4.3) Implementation of Genetic Algorithm in Snake Game

4.3.1) Population Initialization

This is the first step of Genetic Algorithm. In this step a population of about 50 snakes is initialized with there random neural networks as there brain. Each of the snake in the current population plays there game.

4.3.2) Fitness Function Calculation

After each of the snakes in the current population (generation) played there game, then based on there final score a fitness value is calculated for each of the snake in current generation.

4.3.3) Crossover

Selection of Parent:

- Roulette Wheel Selection
- Tournament Selection
- Random Selection

For crossover two parents are selected using Roulette Wheel Selection from the population, and there weights and biases are crossed, so as to get a new brain (neural network). There are multiple ways of doing crossover few of them are:

- One-Point Crossover
- Multi-Point Crossover
- Uniform Crossover

4.3.4) Mutation

Only 10% mutation rate is introduced, so that there can be diversity in the population, diversity in the population will help to produce better child generations, and also to prevent domination of one generation over other generation.

4.3.5) Survivour Selection

The Survivor Selection Policy determines which individuals are to be kicked out and which are to be kept in the next generation. It is crucial as it should ensure that the fitter individuals are not kicked out of the population, while at the same time diversity should be maintained in the population.

There are two most common types used:

1. Age based Selection

2. Fitness based Selection

4.3.6) Termination

The termination condition of a Genetic Algorithm is important in determining when a GA run will end. We usually want a termination condition such that our solution is close to the optimal, at the end of the run.Usually, we keep one of the following termination conditions −

- When there has been no improvement in the population for X iterations.
- When we reach an absolute number of generations.
- When the objective function value has reached a certain pre-defined value.

4.4) Architecture of Neural Network to be used with Genetic Algorithm

*Number of **Layers** in the model* **: 4**

*Number of Neurons in **Input Layer*** **: 24**

*Number of Neurons in **Hidden Layer 1*** **: 9**

*Number of Neurons in **Hidden Layer 2*** **: 9**

*Number of Neurons in **Output layer*** **: 3**

Activation Functions Used **: Sigmoid Activation**

Input layer consists of 8 direction and in each direction the snake is able to see three things :

1. Distance to wall
2. Distance to food
3. Distance to its tail

The output layer consists of three neurons which corresponds to three directions :

1. Left
2. Forward
3. Right

4.5) Code Review

All the code base has been written in python v 3.6 programming language. The main reason of using python language for this project are huge community support, easy to implement, and availablity of tons of libraries which makes the development fast and easy. Object Oriented Programming is used wherever possible. The OOP makes the code looks cleaner and more readable.

Visual Studio Code editor has been used to code in python.

Main Libraries of Python that has been used in project :

- PyGame

  Command : **pip install pygame**

  This command will install pygame module into the system, so that we can use it wherever we want. PyGame module is needed so that we can make the snake game.

- Numpy

  Command : **pip install numpy**

  This command will install numpy module into the system. This module consists of various mathematical tools like matrix multiplication, addition, dot product, random array, etc. These mathematical tools will be very useful in making of the neural network. These are very efficient algorithms and can prove to be very efficient in performing complex calculations.

4.5.1) Food Class

The Food Class constructor generates a Food at any random location into the display. There is only one method in the food class i.e., **show()** method, which shows the food into the display.

4.5.2) Snake Class

The Snake Class constructor generates a new Snake object at a fixed position. The Snake Class consists of various variables and methods some of the important ones are :

1. food : This variable is used to store the food.
2. fitness : This variable is used to store fitness value of the current snake.
3. brain: This variable is the most important variable as it is used to store the neural network that will take the decision in which direction to go for eating food.
4. vision: This variable holds the data that the snakes head sees, it is an array of size 24 that will work as an input to the neural network.
5. move(): This is a method in Snake class that is responsible for moving the snake in the prescribed direction.
6. show(): This method shows the snakes body into the display.
7. calc_fitness(): This method is responsible for calculating the fitness of the snake, based on the final score achived by the snake.
8. look(): This is the most important method, as it is responsible for looking in all the 8 directions for food, walls, and body parts. This method fills in the vision array the corresponding value. So that neural network can make apropriate decision.
9. Other utility methods are also present in the Snake class like saving snake, loading snake, mutate, crossover, also if user wants to play there is a method for that too.

4.5.3) Population Class

Population class consists of various methods that helps the Genetic Algorithm to evolve and make progress as the number of generations grows.

Some of the important methods of the Population class are :

- Its constructor generates a random snake population of size 60, each of the snake in the population has different brain.

- crossover(): This method helps in making the next generation population which will be derived from previous generation and as the generation grows there should be improvement in the scores achieved by new generations.

- selection(): This methods helps in selecting top snakes from previous generation that will be the parent for the next generation.

- run(): Run method runs all the snakes in the current population and show the progress into the display.

4.5.4) Genetic Algorithm Class

This class is responsible for making of the neural network for snake object. The constructor of this class will generate a new random neural network that will act as a brain for snake. The important methods are as follows :

- predict(): Predict method takes in the input array and based on the number of output neurons it outputs values in between 0 to 1. These output values are used by snake class for prediction of the direction to take.

- clone(): This method returns a clone of the neural network.

- crossover(): Crossover method takes in two parents and returns a new brain that is developed with the help of these two parents brain. Crossover makes use of the single point technique for exchanging weights.

- mutation(): Mutation fuction alters some of the value in the brain so that the diversity in the population can be maintained.

CHAPTER - 5

Results and Discussion

Fig. 1: Path Finding by A*(star) Algorithm

Here in the above figure we have used A* path finding algorithm, for finding out a path from bottom right corner to the top left corner.

The results showed that A* algorithm was able to find the shortest path in between two points, in minimal amount of time.

Fig. 2: Breadth First Search path Finding

The BFS algorithm was run under same condition as it was with A* algorithm, and the findings were the BFS algorithm was able to find the shortest path berween two points but the time taken by BFS was more when compared to A* Search Algorithm.

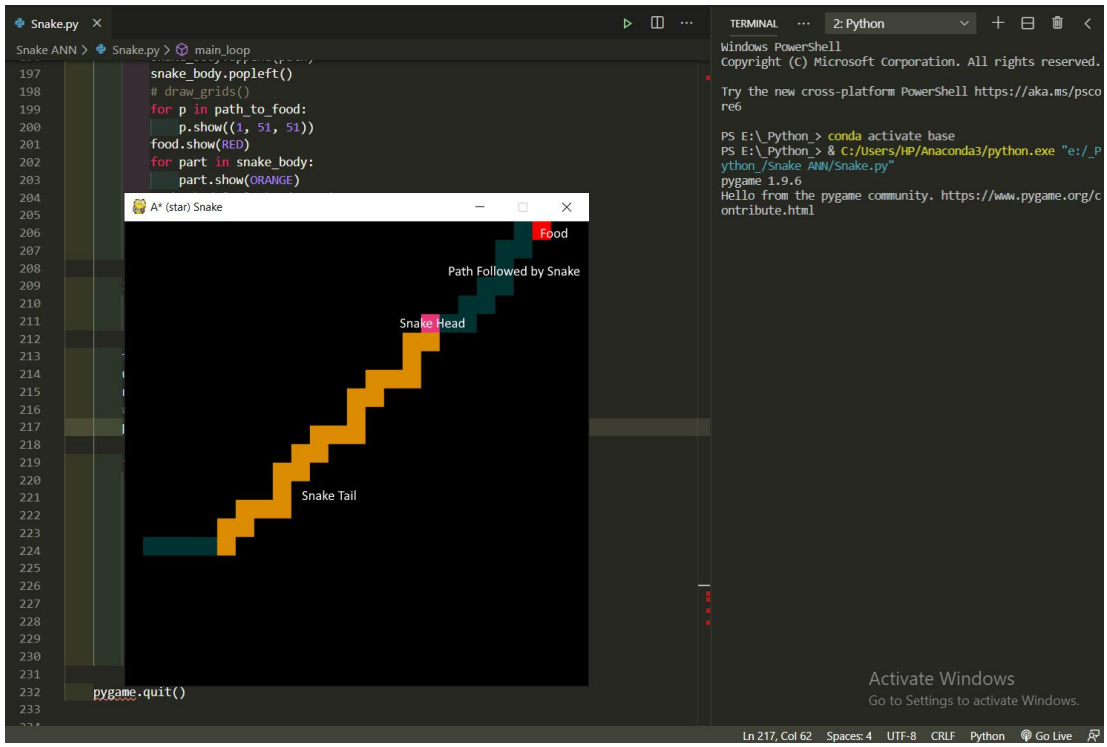

Fig.3 : The Snake game run with A* Algorithm

Here the Snake Game was run with the use of A* path finding algorithm, The algorithm was able to find the path from snake head to the food.

Food is shown in RED color

Head is shown in PINK color

The path that the snake has to follow is shown with GREY color

And the rest of the body is shown with orange color.

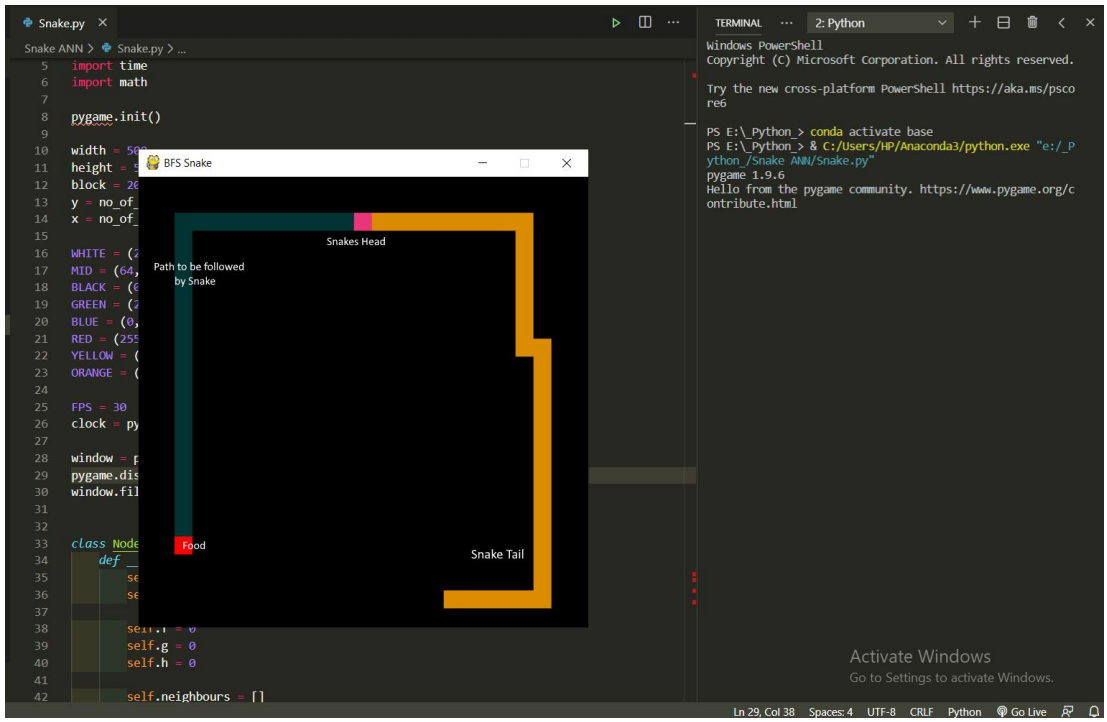

Fig.4 : The Snake game run with BFS algorithm

The BFS algorithm was also able to find the path from snake head to the food.

The problem with both of these implementation was if there does not exists a direct path from head to the food, then in that case both the algorithm was not able to find any path, and the game halts.

To overcome this problem an optimized version was implemented that is shown in next page.

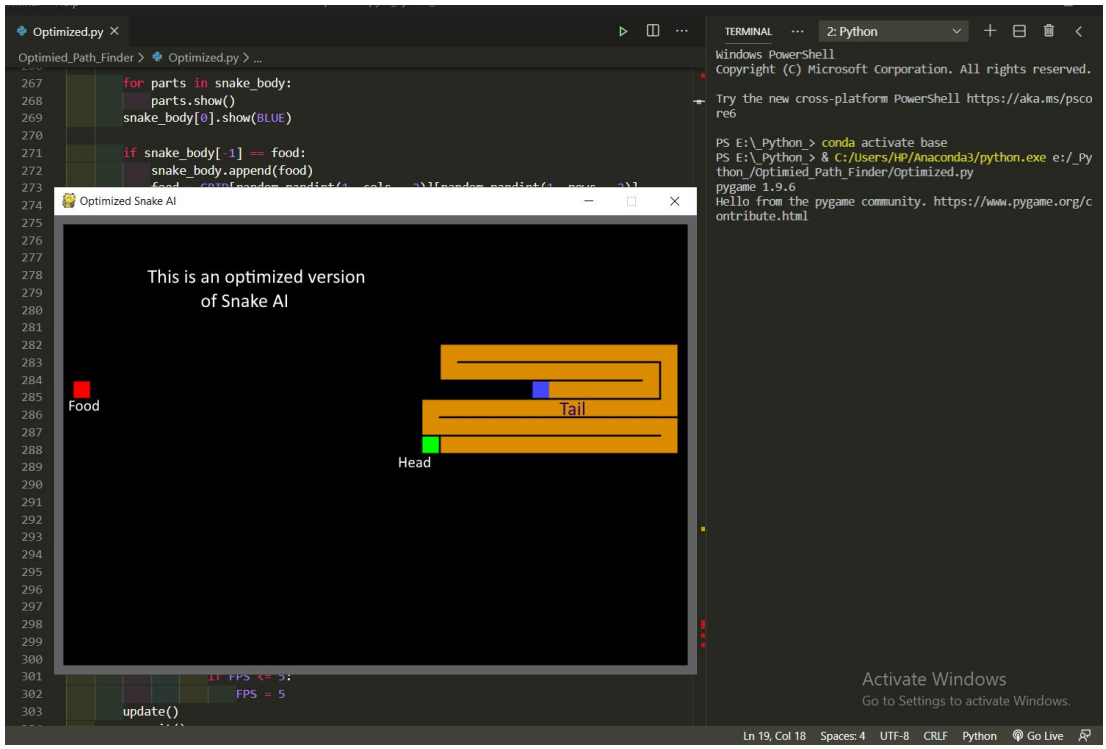

Fig. 5: Optimized Snake AI

This is an optimal Snake AI, this snake has ability to even find a path, if there does not having a direct path from head to food.

CHAPTER - 6

Conclusion and Future Work

At first the Snake game was implemented using path finding algorithm like Breadth First search and A Star(*) search algorithm. Both of these algorithm performed very well but in case of occurrence of loop or when there does not exists a direct path from Snakes head to food, these algorithm were not able to find any path, and they crash the snake to the wall or to its tail.

Then a optimized version of these algorithm was implemented in snake game, this optimized version was able to perform very well then unoptimized version. The average score achieved by this version was very high. But on extreme cases this algorithm was also stuck and was not able to find the direction to take place.

Then a Genetic Algorithm based version is implemented. The GA based algorithm has high potential to achieve very high score as, these algorithms are Optimization Algorithm and they will be able to find complex mathematical equations and there solutions that might be very useful in snake game.

The Genetic Algorithm can be more optimized, which might found some good values for weights and biases used in game. This can be done by trying out different vision systems for snake. Another thing we can do is to implement Artificial Neural Network version of the game, and can see different result.

CHAPTER - 7

References

[1] pygame documentation

[2] Youtube.com

[3] Medium articles

[4] Book - Make your own neural network