Orchid International College

Tribhuvan University

Institute of Science and Technology

Project Report on
Game Playing using Neuroevolution

Submitted to
Department of Computer Science and Information Technology
Orchid International College

In partial fulfillment of the requirement for the Bachelor Degree in Computer Science and Information Technology

Submitted By

Animesh Risal (4965/071)

Niroj Shayaju (4985/071)

September, 2018

Acknowledgements

The completion of this project would not have been possible without the support and guidance of many individuals.

We are grateful to Orchid International College for guidance and supervision, as well as providing all the necessary support and friendly environment for the successful completion of the project.

We would like to express our gratitude to our project supervisors **Mr. Nipun Thapa** who took an interest in our project and guided us through the project by providing necessary ideas, information and knowledge for developing an AI Agent and a game system. We would like to thank **Mr. Utsav Koirala** and **Mr. Dhiraj Jha** for their encouragement and guidance towards the making of this report as per the standard.

We are thankful and fortunate enough to get constant support from our colleagues and teaching staff of B.Sc.CSIT department which helped us complete our project. We would also like to extend our regards to all the non teaching staff of B.Sc.CSIT department for their timely support.

Animesh Risal (4965/071)

Niroj Shayaju (4985/071)

Abstract

This report deals with the use of a neuroevolution algorithm called NEAT which combines the use of a Feed Forward Neural Network and the principles of Genetic Algorithm such as fitness evaluation, selection, crossover and mutation to evolve the structure of the neural network. The neural network will use the grid data of a games screen to allow an AI Agent to learn how to play games and learn how to avoid obstacles.

The area of Artificial Intelligence is a growing area in the modern age with it being used in a lot of areas. There are various disciplines in Artificial Intelligence which can be used to solve problems.

This project focuses on creating an AI agent and a 2D game that will read the grid data of a game and be able to learn how to avoid collision without any human intervention. The game will offer two modes where in one mode the Agent will play alongside the Player and see which survives longer and the other mode where the player will generate collision object using mouse click which the AI Agent has to dodge.

List of Tables

List of Figures

List of Abbreviations

AI – Artificial Intelligence

FPS – Frames Per Second

NEAT – Neuroevolution of Augmenting Topologies

NES – Nintendo Entertainment System

SDL - Simple DirectMedia Layer

Table of Contents

Chapter 1 - Introduction

1.1 Background

The field of artificial intelligence is a huge and established research field, and is still growing every day. One of the common techniques, which is applicable to a wide range of problems is the area of neuroevolution, which refers to the evolution of artificial neural networks using genetic/evolutionary algorithms.

Genetic algorithm are adaptive heuristic search algorithms based on evolutionary ideas of natural selection and genetic. They are useful when we want to quickly find a solution in a large state space.

Artificial Neural Networks are computing systems that are inspired by the biological neural networks. The key element of a neural network is its structure. It is composed of a large number of highly interconnected processing elements working in parallel to solve a problem.

Neuroevolution is an important method that has been gaining popularity over the last few years with its use in robot control, music generation and especially game playing.

Game playing is a way in which an agents interact in an environment and make actions based on what the events are occurring in the game. Depending on the game, the rules are different and the agent acts differently based on what they interact with.

The aim is to use Neuroevolution to allow an AI agent in a game to learn to play the game itself. The proposed algorithm for the game playing AI Agent is the NEAT algorithm.

NEAT is a method of neuroevolution where neural network evolve artificially over several generations not only for their weights but for their topology as well. The fitness function acts as a feedback and tries to find the best neural network which can maximize the fitness function.

This type of learning is called reinforcement learning as the AI learns to play better as it is given more rewards for doing the task properly.

1.2 Statement of Problem

The creation of the AI Agent is done to research certain aspects of AI like the ability of using the concept of neural network to aid in the ability of game playing. When designing a neural network, we have to create the structure having already assumed the best structure for the AI Agent. This can sometimes work but can sometimes not work as we can't usually create the best neural network if we start introducing new scenarios the predefined neural network structure is not familiar with.

1.3 Objectives

- To create an AI agent that uses genetic algorithm to evolve the structure of a neural network
- To create an AI agent that makes decision based on grid data provided by the game
- To allow the AI agent to learn to play games without human intervention
- To train the AI agent so it can get better at playing the game and get better high scores
- To create a game where a user can play with or against the AI agent

1.4 Scope

The game will be created based around certain conditions such as

- A grid cell will be of size 60x60 pixels
- The game will have two modes.
    - First Mode - User plays along with the agent and see which survives longer.
    - Second Mode - Other where the user generates collision object for the agent to dodge.

1.5 Limitations

- The gird will be of size 5 x 7
- The AI Agent will move only left or right
- The player will only be able to initialize 2 collision object at one time

1.6 Literature Review

When it comes to using a learning algorithm, there are several ways in which the process can be done. The ability to use neural network with backpropagation is one of the traditional approaches in training a neural network. [1]

The another approach in training a neural network is instead of using backpropagation there is the option of using genetic algorithm[2] to adjust the weights of the connections in the neural network.

While both of these are possible ways in which an AI Agent can be trained there can come the issue of how much weights and layers of neurons are needed.

The algorithm NEAT [3] which clears the issue from the first two process has the ability to generate different neural networks and weights based on the how it is being trained. The algorithms doesn't require us to presuppose the best neural network structure for the AI.

For this project we've decided to go with the use of NEAT algorithm as it allows us to create an AI Agent that can be used for multiple environments.

Chapter 2 – System Analysis

2.1 Overview

It is important to make sure that we get the idea of the process of making the system. The work can be broken down into different phases to allow a good development process.

2.2 System Modeling

For this project we have used Waterfall model for the development process. Although it is an older type of development process, there aren't many changing requirements.

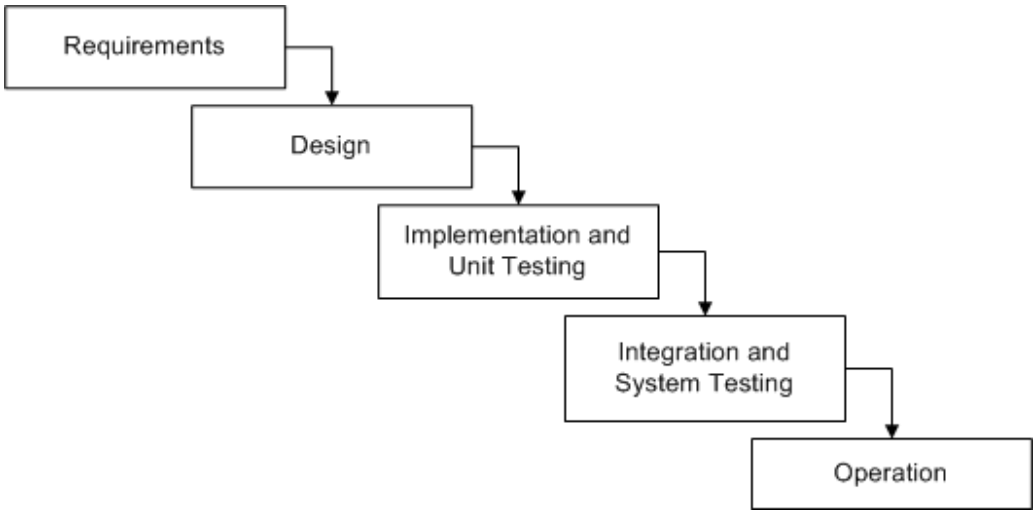


Figure 2.1 - Waterfall Model

2.3 Requirement Analysis

Requirement analysis is a process of analysis and evaluation of a proposed project to determine if it technologically feasible, economically be profitable.

2.3.1 Functional Requirement

Player chooses between game modes – The game offers a choice of two modes where the player can choose to play as a spaceship alongside the AI Agent or play as an enemy that generates collision

2.3.1.1 Player generates collision object

The mode where the player is the enemy, they can click on the top 3 rows to generate new collision objects.

2.3.1.2 Game increases collision object count

In the mode where the player plays along with the AI agent, the number of count of collision object increases as time goes increasing the game's difficulty.

2.3.1.3 Use Case Diagram

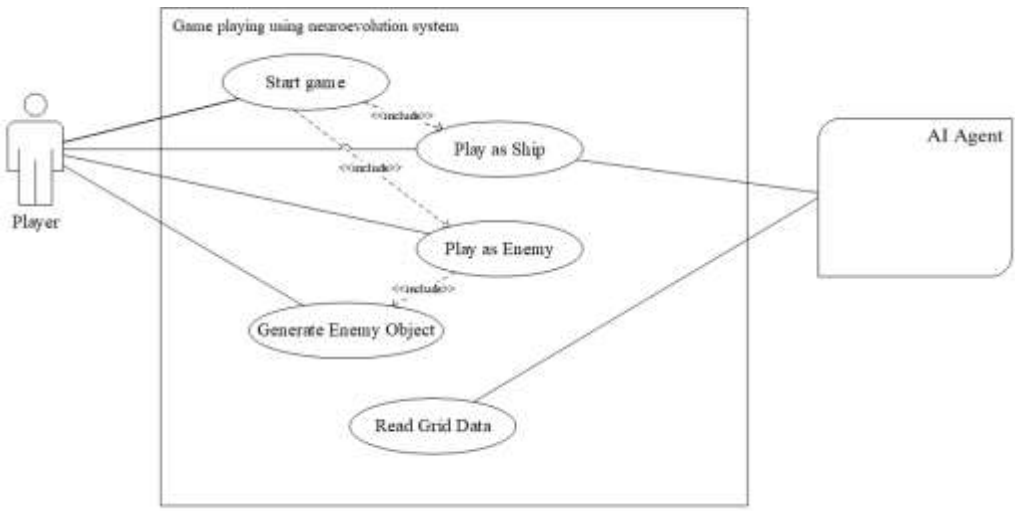


Figure 2.2 - Use Case Diagram

Use Cases Description

Table 2.1 - Start Game

Use Case Number	1
Use Case Name	Start Game
Primary Actors	Player
Description	A player can start the game and can have the choice of either playing as a ship alongside the Agent or the user can choose to play as an enemy that generates collision objects to collide with the Agent's spaceship.

Table 2.2 - Play as ship

Use Case Number	2
Use Case Name	Play as ship
Primary Actors	Player, AI Agent
Description	If the user chooses to play as a ship then the user will control a ship in the game and will play alongside with the Agent and see which ship can last longer in the game.

Table 2.3 - Play as Enemy

Use Case Number	3
Use Case Name	Play as Enemy
Primary Actors	Player
Description	If the user chooses to play as the enemy, the user is able to generate enemy objects.

Table 2.4 - Generate Enemy Object

Use Case Number	4
Use Case Name	Generate Enemy Object
Primary Actors	Player
Description	The user clicks the mouse button to generate an enemy object which the Agent will try to avoid getting hit with.

Table 2.5 - Read Grid Data

Use Case Number	5
Use Case Name	Read Grid Data
Primary Actors	AI Agent
Description	The game system generates grid data in a matrix showing where the collision object positions are which are fed to the AI agents input neurons.

2.3.2 Non Functional Requirement

a) Game uses pixel graphics

For the art style of the game it is going to be using pixel graphics giving the game a more retro and arcade feel to it.

b) Game Framerate.

To ensure the game doesn't feel running too fast and the player can play at ease, the game is locked at a steady frames of 12 FPS. This also allows for enough computation to occur without slowdowns.

c) Cross Platform

The game is able to run on any platform that supports Python 3 which includes all of the major platforms like Windows, Mac and Linux.

2.4 Feasibility Study

The feasibility of the project is analyzed into different parts so that it is possible to create the simulation. For feasibility analysis, understanding of some requirements are essential. The feasibility analysis is divided into 3 parts.

2.4.1 Technical Feasibility

The system can be feasible as it can run on any modern day computer that runs on a modern operating system. A minimum of 1 GB ram is recommended to run the program properly. If the player decides to train the AI Agent themselves then a computer with more power will be required to train it.

2.4.2 Economic Feasibility

Since the system is a desktop application the cost is minimized as there will not be any server costs to upkeep and maintain the system.

2.4.3 Operational Feasibility

The game can be easily operated as playing the game doesn’t require a lot of button presses and the menu is easily navigable. The development tools are also free modules that are used to create the game modules.

2.5 Context Diagram

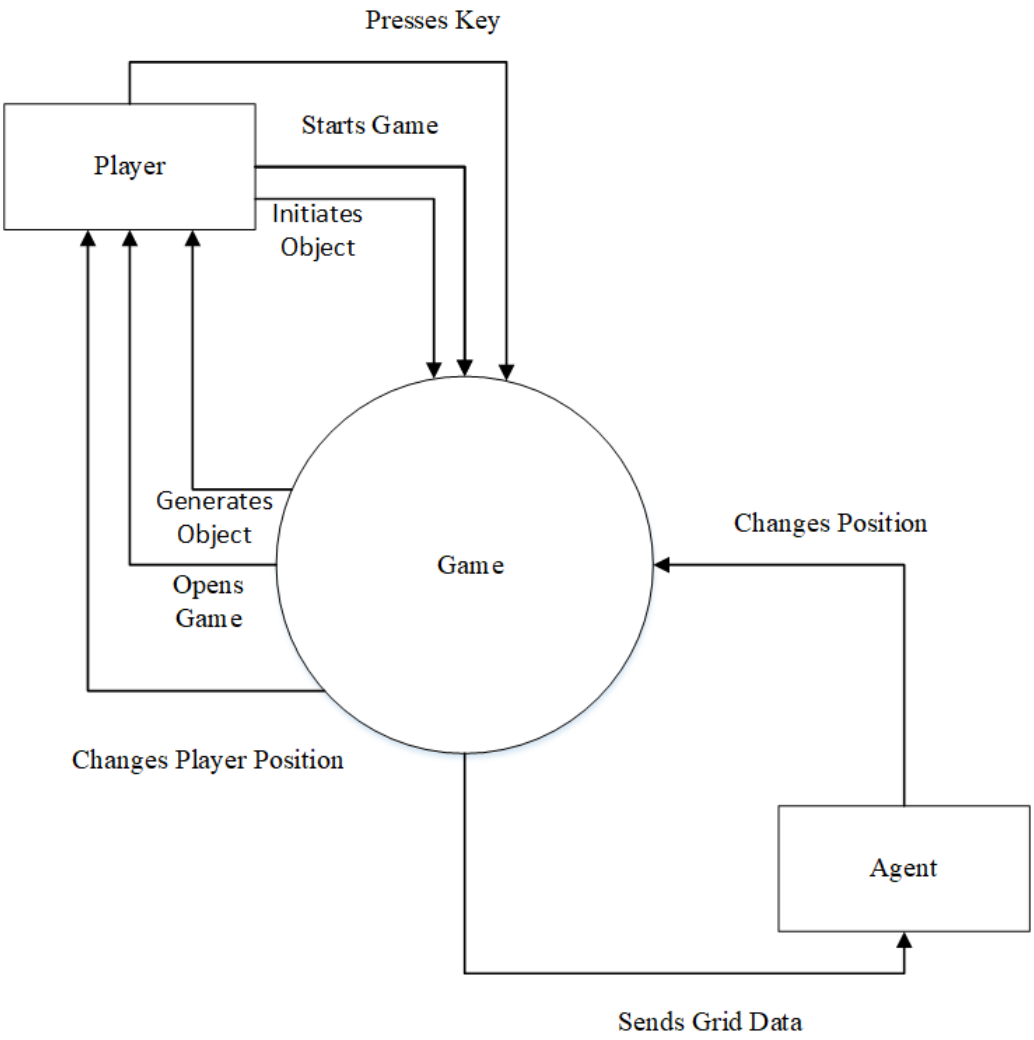


Figure 2.3 - Context Flow Diagram

2.6 Data Flow Diagram

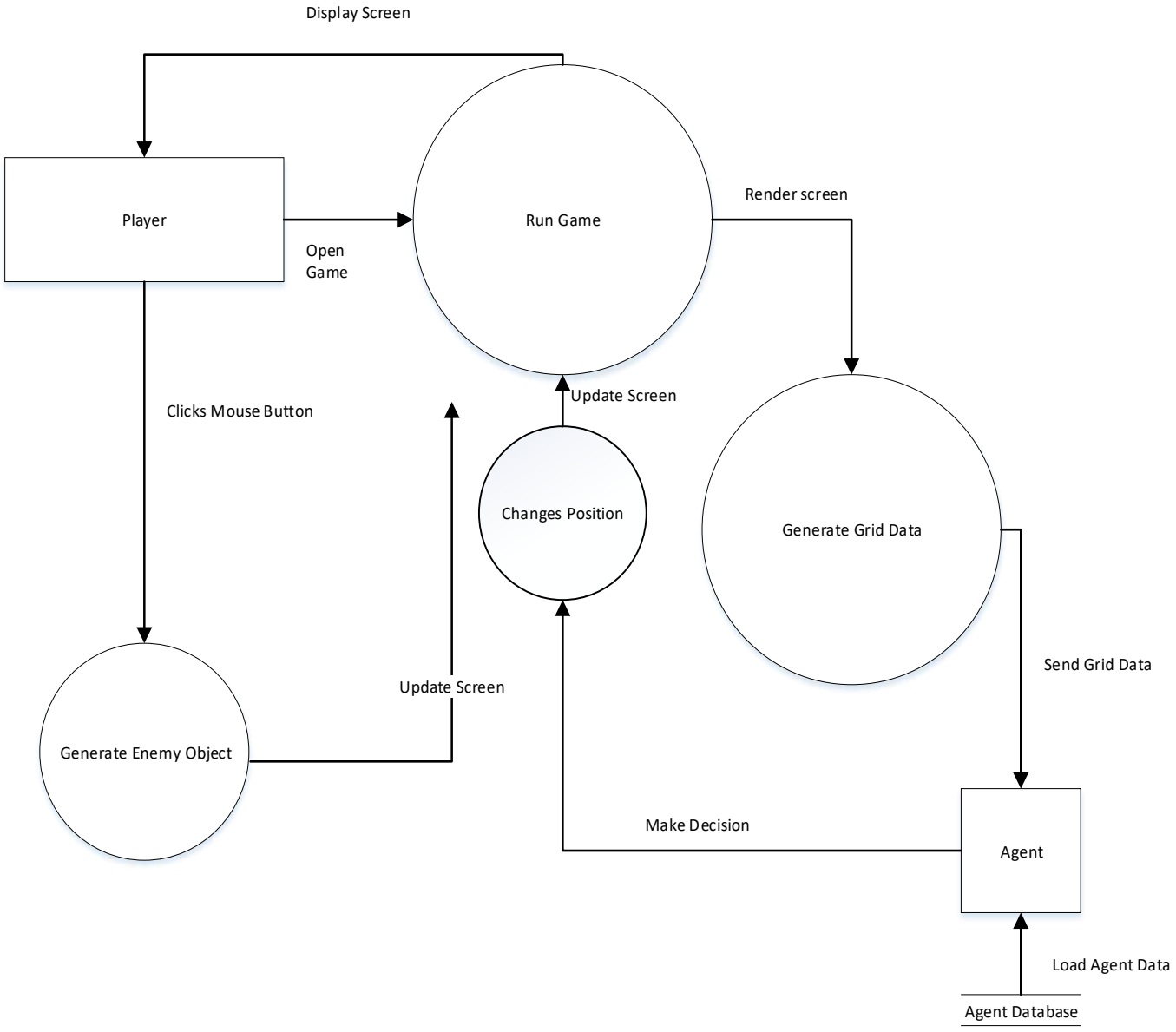


Figure 2.4 - DFD Level 1

Chapter 3 - System Design

3.1 Overview

System design is the process of defining the architecture, diagrams, interfaces, and data for a system to satisfy certain requirements.

3.2 NEAT Algorithm

The algorithm is divided into the parts. The first part is a Feed Forward Neural Network which is used to perform the computation. The second part is the use the principles of genetic algorithm to evolve the structure of the Neural Network.

3.2.1 Feed Forward Neural Network

a) Activation Function

When doing computation with a neural network, the most fundamental part of the neural network is the activation function. Feed forward neural networks can use an activation function called the Sigmoid Function.

The mathematical function of a sigmoid function is

$$f(x) = \frac{1}{(1 + e^{-x})}$$

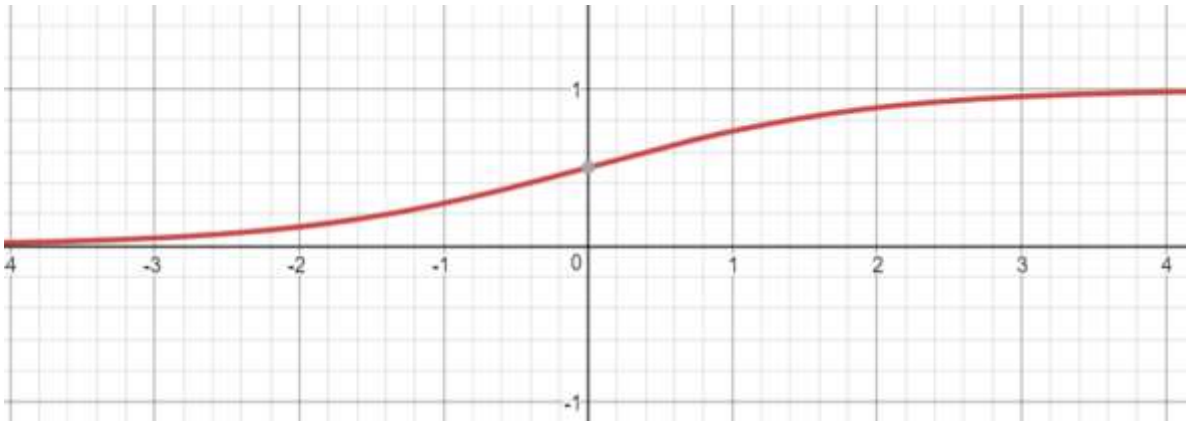


Figure 3.1 - Sigmoid Function Graph

A sigmoid function gives us a nonlinear 'S' curve that normalizes a value between 0 and 1. One change that has been made is to change the constant of x to 4.9x steepening the curve. So the mathematical equation becomes

$$f(x) = \frac{1}{(1 + e^{-4.9x})}$$

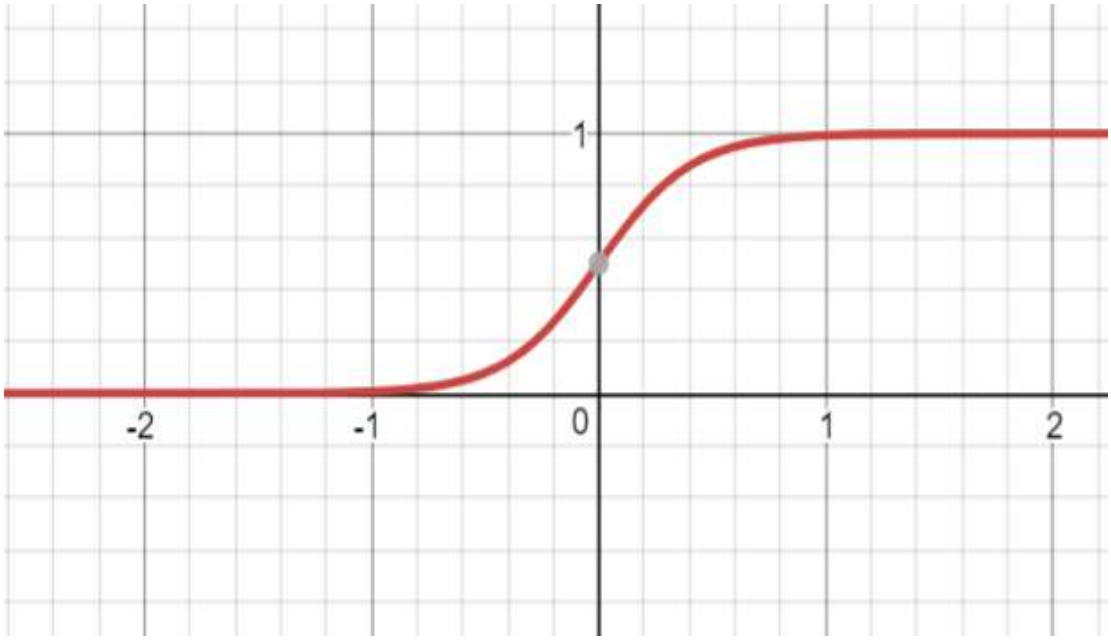


Figure 3.2 – Sigmoid Function with steepened constant

The steepened sigmoid allows more fine tuning at extreme activations. It is optimized to be close to linear during its steepest ascent between activations −0.5 and 0.5

Since the range is required to be in between -1 and 1, the maximum value has been changed from 1 to 2 giving us a range of 0 to 2 and then subtract at the end with 1 giving us a range of between -1 and 1. The mathematical function with the modified values is depicted as

$$f(x) = \frac{2}{(1 + e^{-4.9x})} - 1$$

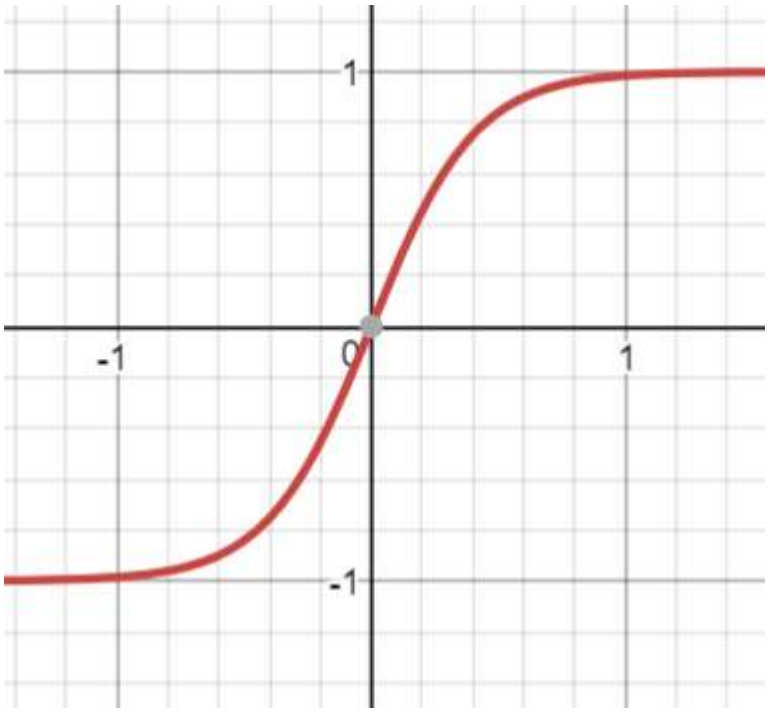


Figure 3.3 – Sigmoid function with steepend constant and modified values

b) Network Structure

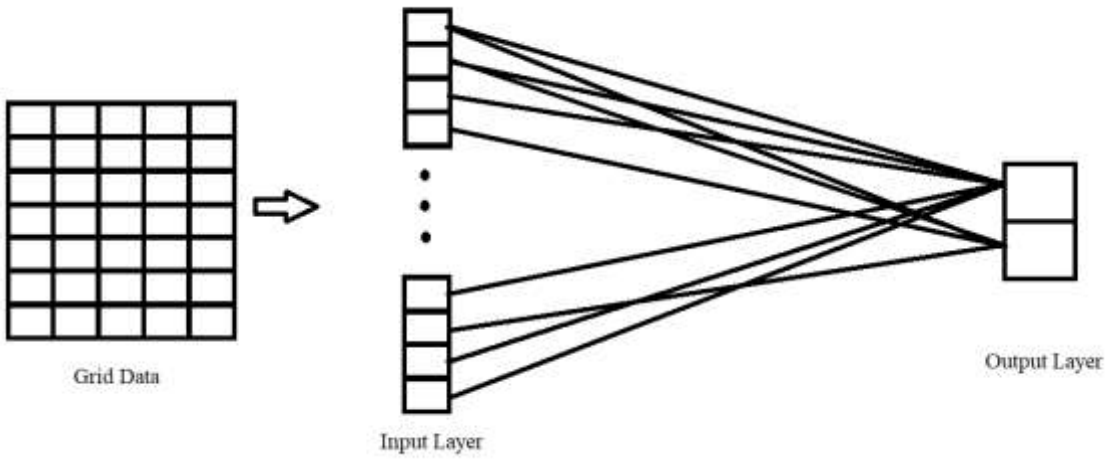


Figure 3.4 - Initial Neural Network Structure

The game environment has a resolution of 420x300. Every 60x60 pixels is equal to 1 grid. Since the games data is read using the grids, the grids act as the input layer of the neural network where each grid equals to 1 input neuron. Since the game consists of a 5 x 7 grids. There's a total number of 35 input neurons. The grids can have a value of 0 or -1. 0 represents there are no collision objects on the grid while -1 represents that there are collision objects on the grid. Since the agent has the ability to move left or right. The neural network consists of two output neurons. One to move left and the other to move right. The agent will move to a direction if the value at the either output neurons is greater than 0.5 at output node and which neuron has the higher value. During the initial phase of the neural network. All the input neurons connect to both output neurons, giving a total number of 70 connections. Since there is a direct connection between the input layer and output layer, there is initially no nodes at the hidden layer. The nodes at the hidden layers is generation by the process of mutation during the genetic algorithm phase.

3.2.2 Genetic Algorithm

While the neural network is used to perform computation. Genetic Algorithm is used in the training program and is used to evolve the structure of the neural network to make it adapt to the game environment.

Genetic Algorithm consists of two things. Genotypes and Phenotypes

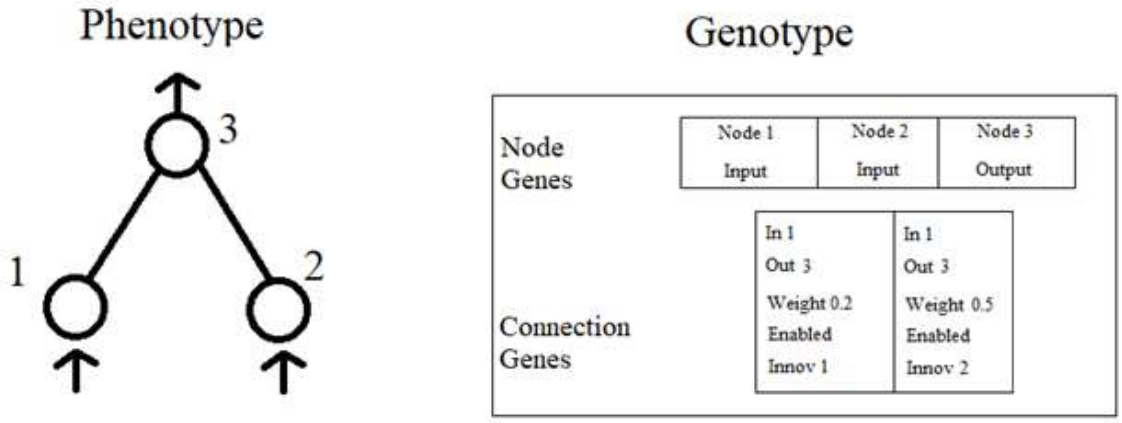


Figure 3.5 - Genotype and Phenotype

The above diagram shows the genotype to phenotype in the NEAT Algorithm. The phenotype is depicted from the shown genotype.

NEAT consists of two types of genes. The first is the Node Gene which has the information on its node number and what kind of node it is (input, hidden or output). The other is the connection gene which consists of a lot more information such as its input neuron, output neuron and its weight. It is has the information on whether it is disabled or not. Disabling signifies if the connection can send data or not. The most important element of the connection gene is the innovation number. The innovation numbers are historical markers that identify the original historical ancestor of each gene. New genes are assigned increasingly higher numbers. When adding a connection, a single new connection gene is added to the end of the neural network and given the next available innovation number.

The following process takes place during the neural network training

a) Population Creation

While the algorithm can create different species of the neural network, for this version only a single species is used during the training with a population of 40 neural networks is created which is fully connected.

b) Fitness Evaluation

The fitness evaluation of the neural networks is done by the score they receive during training. The higher the value, the higher the fitness. The neural networks are then sorted according to their fitness value.

c) Selection

After the evaluation process is done. The top 20 neural networks are selected for crossover process. The bottom 20 neural network are removed from the population. To fill up the population again the process of crossover takes place and the next generations of neural networks are created.

d) Crossover

Crossover happens between two neural networks to produce an offspring neural network. When crossing over, the genes in both neural network with the same innovation numbers are kept in the child node. These genes are called matching genes. Genes that do not match are either disjoint (non-matching genes appearing inside the range of the fit parents genes) or excess (non-matching genes appearing outside the range of the fit parents genes), depending on whether they occur within or outside the range of the other parent's innovation numbers. They represent structure that is not present in the other neural network. When creating the offspring, genes are randomly chosen from either parent at matching genes, whereas all excess or disjoint genes are always included from the fit parent. So the child neural network will have the same structure as the fit parent. There is a random chance the matching gene in the child will have a weight from either the fit parent or the unfit parent and if either one of the connection genes is disabled in the parents, there is a random chance the child can have a disabled connection gene as well.

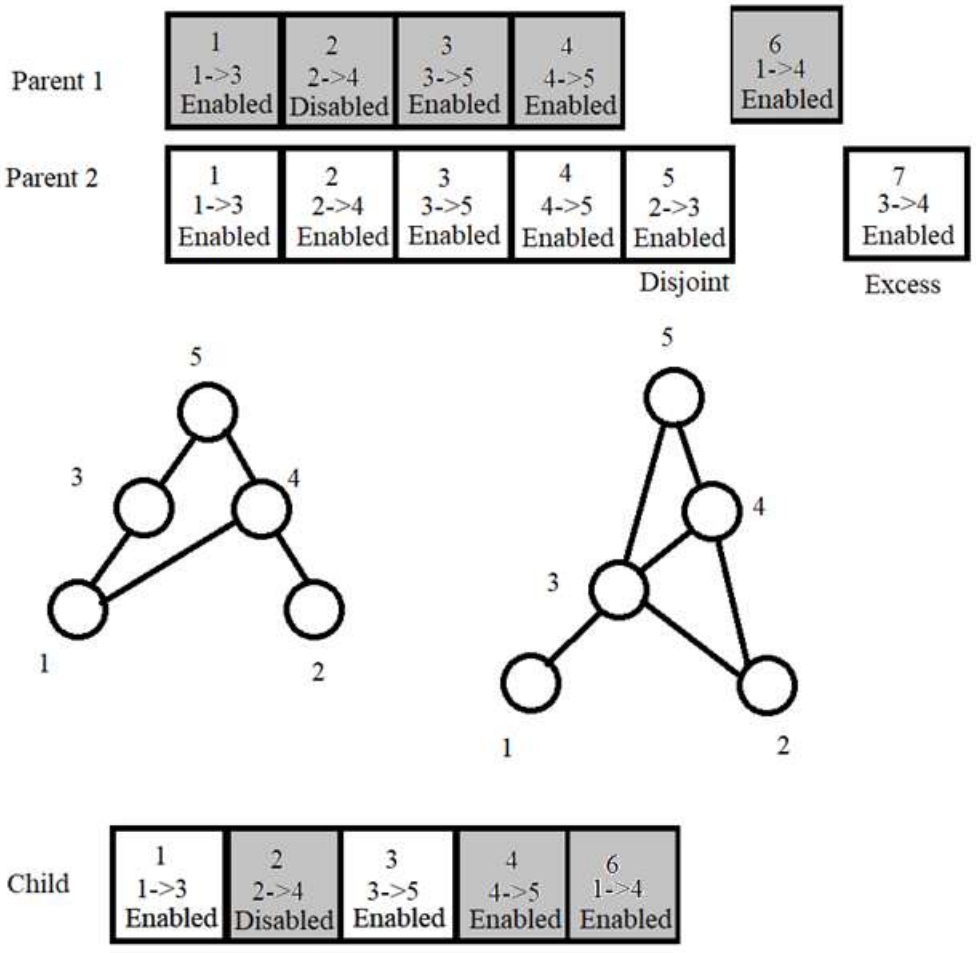


Figure 3.6 - Crossover

e) Mutation

i. Node Mutation

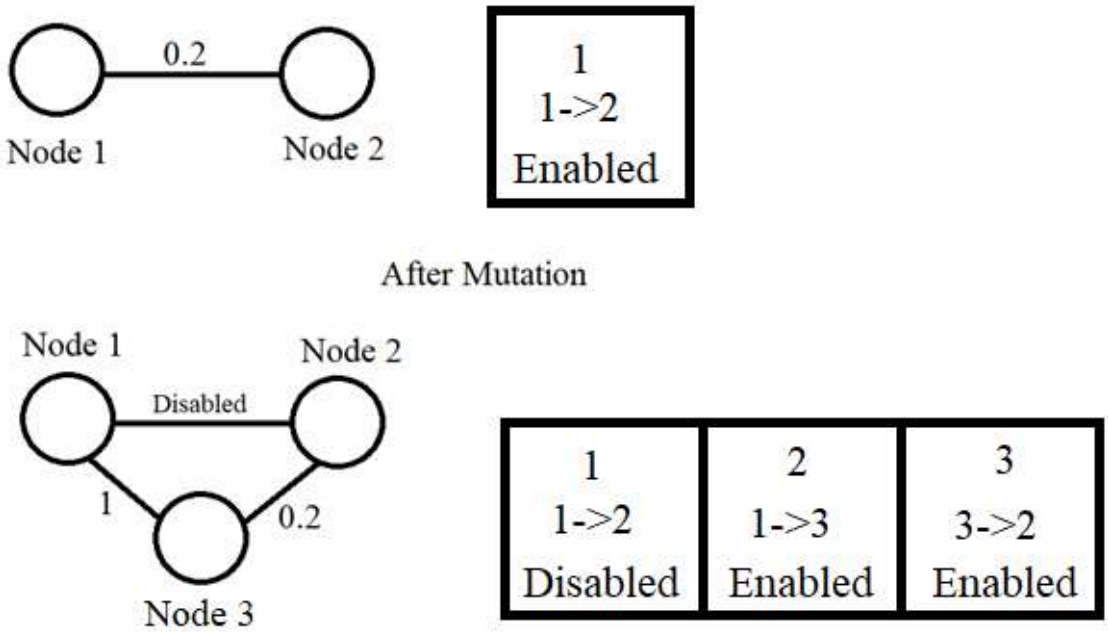


Figure 3.7 - Node Mutation

If a node mutation occurs, a new node is added between any two neurons in the neural network. The old connection between the two neurons is disabled and two new connections are added to the neural network. The weight of the connection leading to the new node is 1 while the weight from the new node leading out has the same weight as before.

ii. Connection Mutation

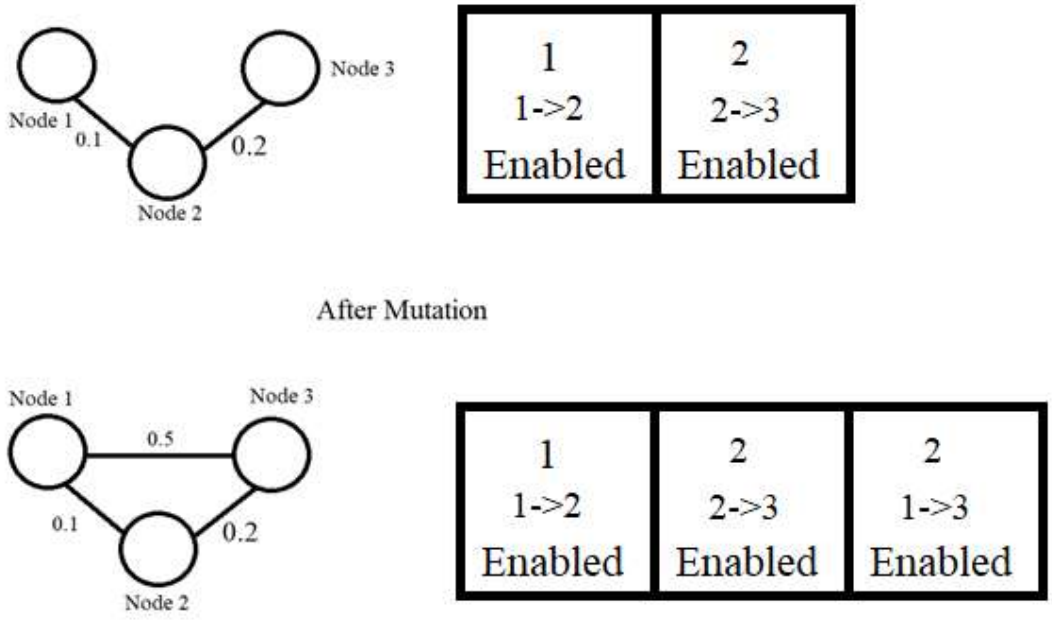


Figure 3.8 - Connection Mutation

If a connection mutation occurs, a new connection is made between any two existing neurons and is given a random weight.

iii. Weight Mutation

If a weight mutation occurs, then there's a chance that the weight of a connection can change to a random value.

3.3 Class Diagram

3.3.1 Agent Class Diagram

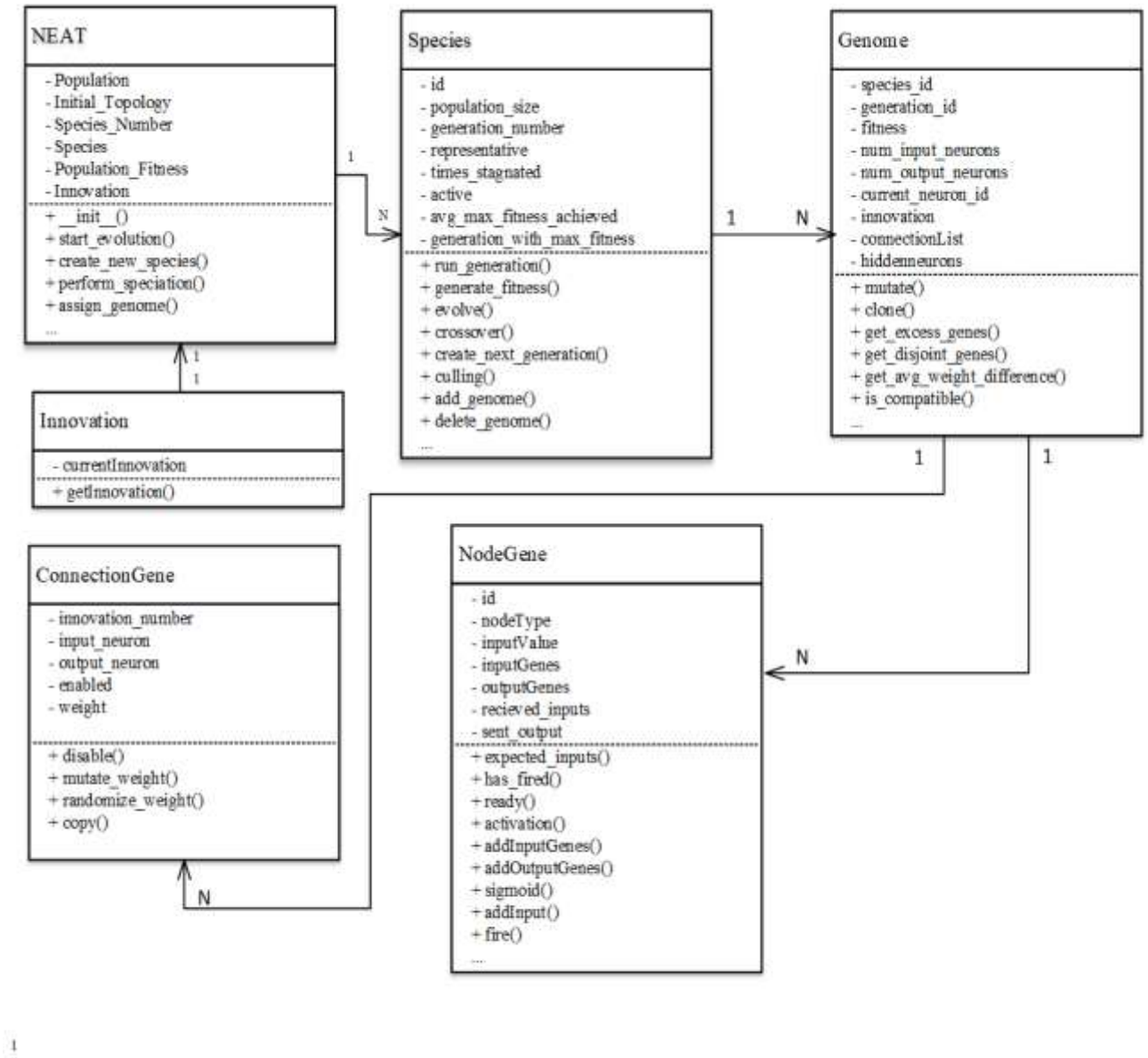


Figure 3.9 - Agent Class Diagram

3.3.2 Game System Class Diagram

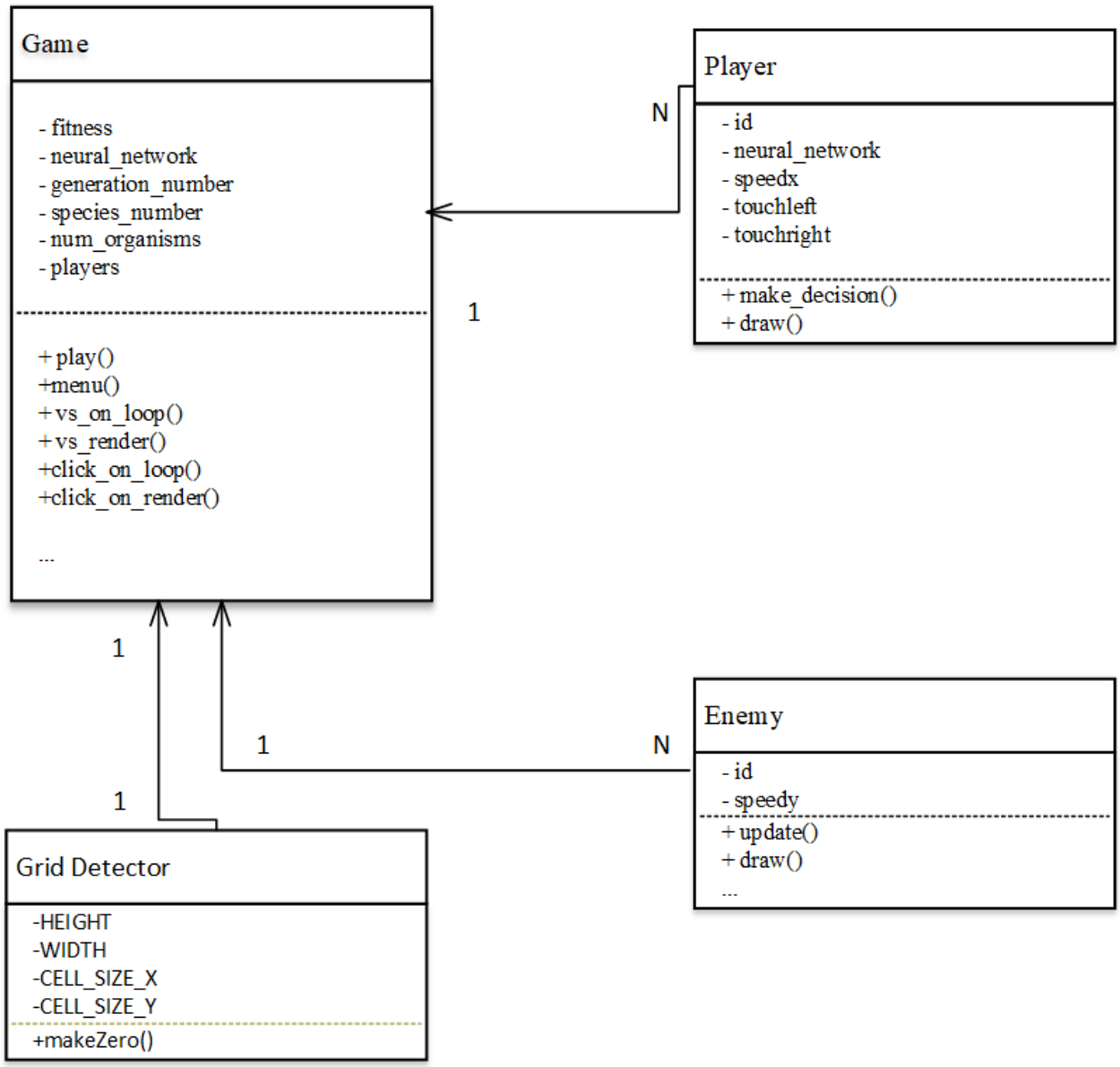


Figure 3.10 - Game System Class Diagram

3.4 Activity Diagram

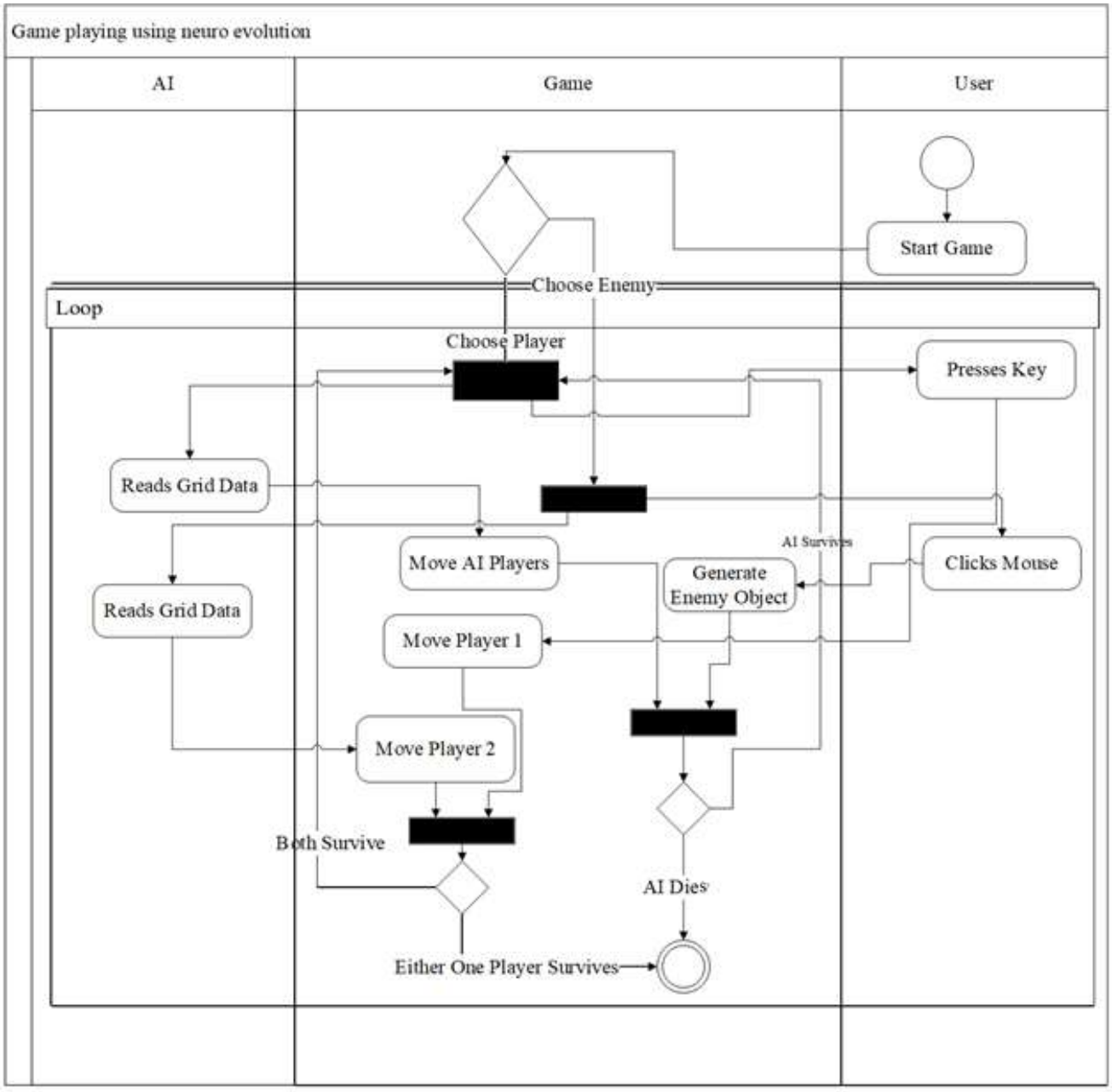


Figure 3.11 - Activity Diagram

3.5 State Diagram

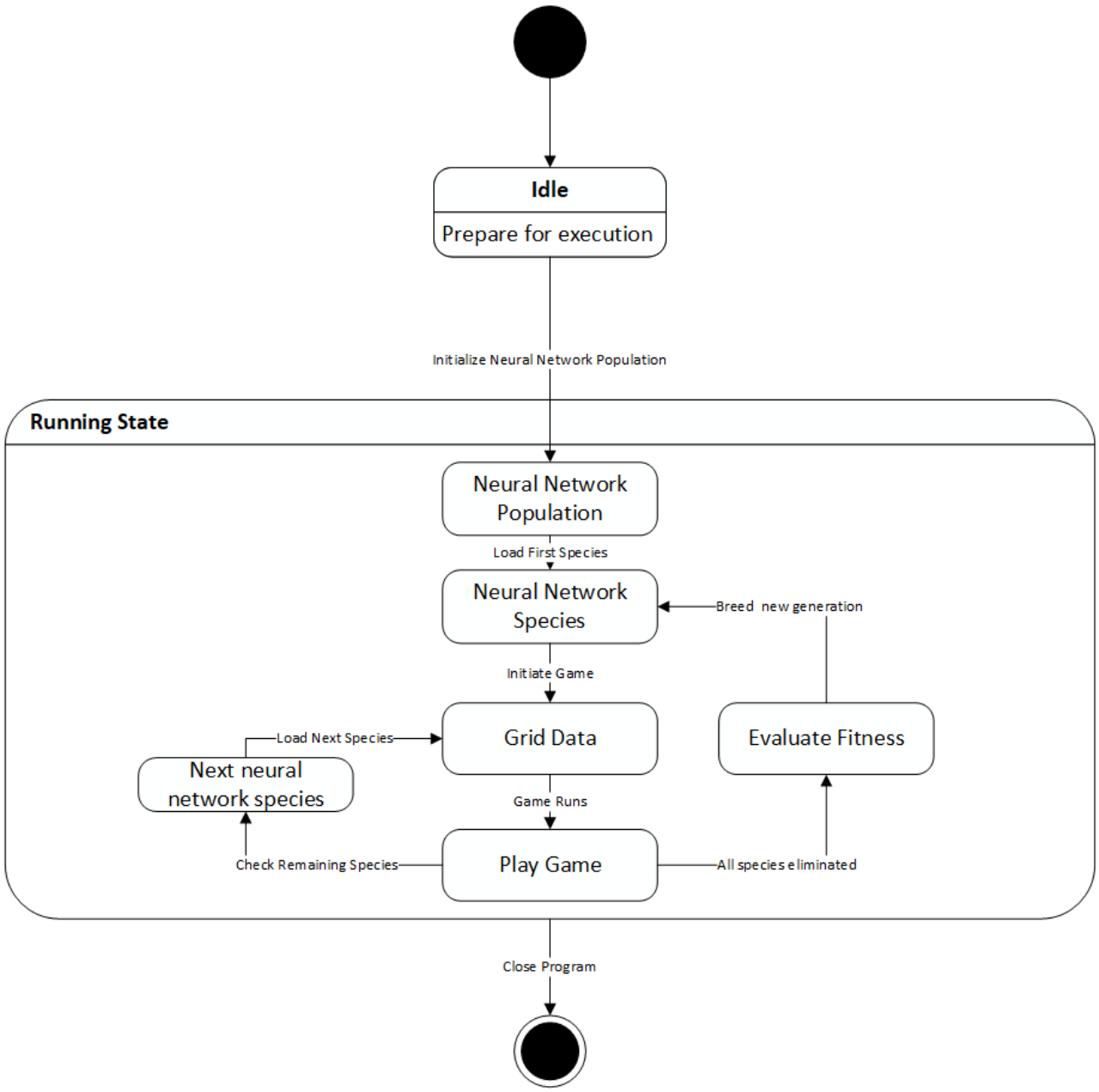


Figure 3.12 - State Diagram

3.6 Sequence Diagram

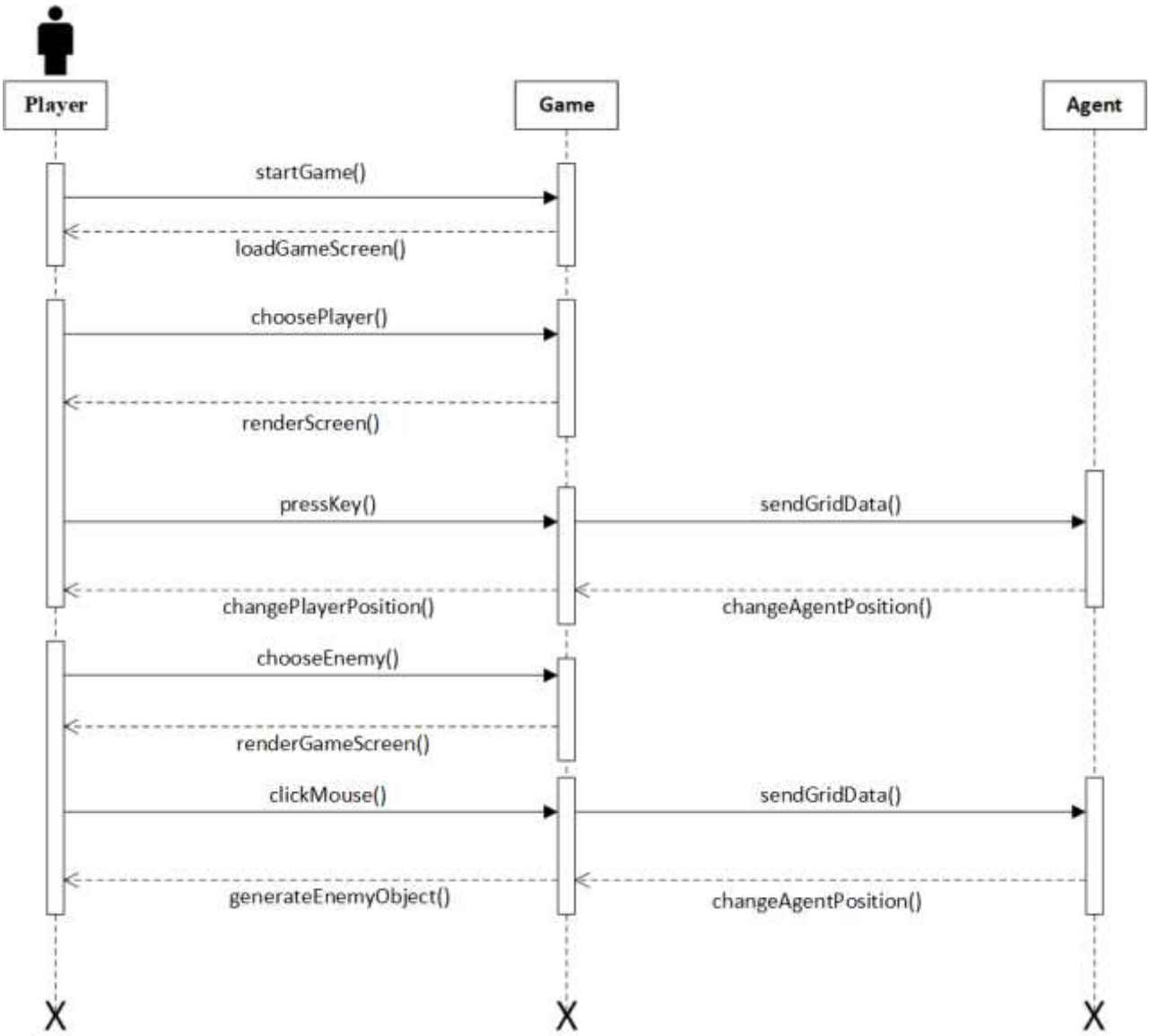


Figure 3.13 - Sequence Diagram

Chapter 4 - System Development and Testing

4.1 Overview

Software Development is the process of developing software in phases. This involves writing code based on the designs and meeting the requirements of the objective.

4.2 Programming Tools

a) Python 3

Python is a high level, interpreted, programming language. It is a language that has a design philosophy that emphasizes code readability, and syntax that allows programmers to write more logic in less lines of code. Python is meant to be an easily readable language. The syntax of Python resembles a lot like English with the use of some English keywords as syntax. Unlike other language it does not have blocks, and the use of semicolon is optional. Python uses indentation to create code blocks.

Python 3 is the latest version of Python that has more features even though there is still support for Python 2.

b) Pygame

Pygame is a Python module designed for writing video games that works on all platforms. It includes ability of producing computer graphics and sound libraries designed to be used with the Python programming language. It is built over the SDL library, with the intention of allowing real-time computer game development without having to use the low-level mechanics of the C programming language.

c) Visual Studio Code

Visual Studio Code is a code editor developed by Microsoft for all major platforms including Windows, Linux and macOS. It includes support for debugging, Git control, syntax highlighting, intelligent code completion, snippets, and code refactoring. It is also customizable, so users can change the editor's theme, keyboard shortcuts, and preferences.

d) Git

Git is a distributed version control management system designed by Linus Torvalds that allows for tracking changes in computer files and allows working with other people much easier. It is used by software developers for managing source code during software development. Since it is distributed As Git is a distributed system, developers can have a repository of a source code, with full history of the software, tracking abilities that is independent from a central server.

4.3 Multimedia Tools

a) Aseprite

Aseprite is a software program that is used to create animated sprites & pixel art. Sprites are little images that can be used in your website or in a video game. It allows a user to draw characters with movement, intros, textures, patterns, backgrounds, logos, color palettes, isometric levels, etc.

b) Famitracker

FamiTracker is a windows tracker for producing music for the NES/Famicom-systems. It allows the user to create chiptune music.

c) Bfxr

Bfxr is a software that allows the creation of sound effects for a game.

4.4 Source Code

4.4.1 Collision Object Initialization

```
self.mouse = pygame.mouse.get_pos()

self.click = pygame.mouse.get_pressed()

if self.mouse[0] < 300 and self.mouse[1] < 180:

    x = self.mouse[0] - self.mouse[0] % 60

    y = self.mouse[1] - self.mouse[1] % 60

    if self.click[0] == 1 and self.mouse_limit == 0:

        if len(self.enemy) < 2:

            self.mouse_limit = 1

            self.enemy.append(Enemy(x , y))

    else:

        self.mouse_limit = 0
```

4.4.2 Grid Detector

```
from color import *


WIDTH = 700

HEIGHT = 420


class Detector:

  def __init__(self,HEIGHT, WIDTH, CELL_SIZE):

    self.HEIGHT = HEIGHT

    self.WIDTH = WIDTH

    self.CELL_SIZE_X = int(WIDTH / CELL_SIZE)
```

```
    self.CELL_SIZE_Y = int(HEIGHT / CELL_SIZE)
    self.makeZero()

def makeZero(self):
    self.matrix = [[0] * self.CELL_SIZE_X for i1 in range(self.CELL_SIZE_Y) ]

def fillMatrix(self, game):
    for enemy in game.enemy:
        x = int(enemy.rect.top / 60)
        y = int(enemy.rect.left / 60)

        if((x >= 0 and y >= 0) and (x < HEIGHT / 60)):
            game.detector.matrix[x][y] = -1
```

4.5 Testing

A few test scenarios have been created to evaluate the performance of the AI Agent and the Game.

4.5.1 Fitness Testing

The following fitness evaluation was done with 40 neural networks trying to avoid 3 randomly generating collision object which are at least 1 grid apart. The test is done for the first 25 generations. All three types of sigmoid functions have been tested to show the results on how they perform. Max Fitness the maximum fitness of each generation and Total fitness is the sum of all the fitness of all the neural network in each generation.

a) Normal Sigmoid Function

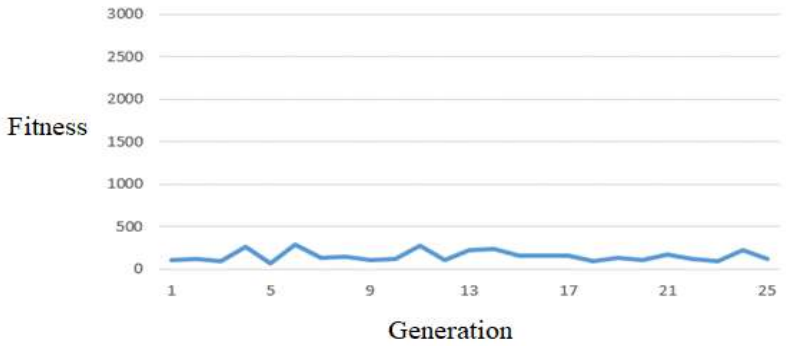


Figure 4.1 - Max Fitness using normal sigmoid function

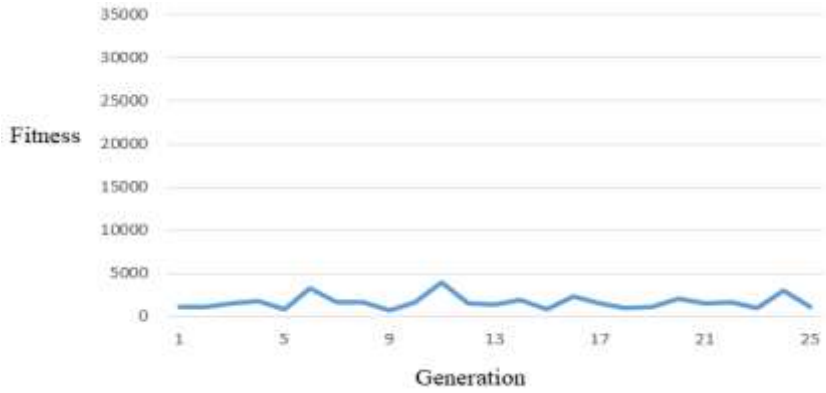


Figure 4.2 - Total Fitness using normal sigmoid function

b) Sigmoid Function with steepened curve

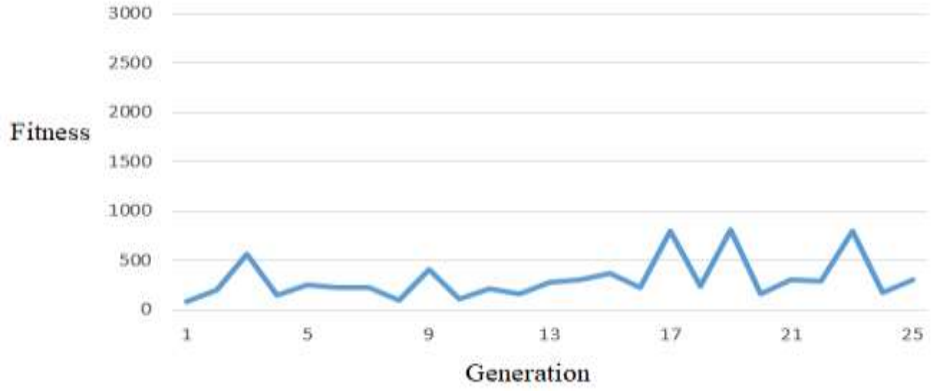


Figure 4.3 - Max Fitness using a steepened curve

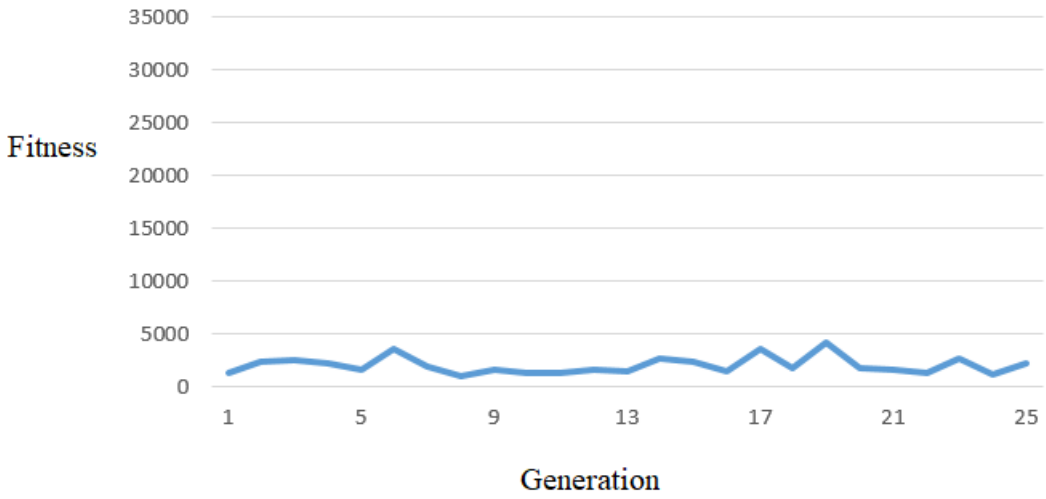


Figure 4.4 - Total Fitness using a steepened curve

c) Sigmoid Function with steepened curve and modified values

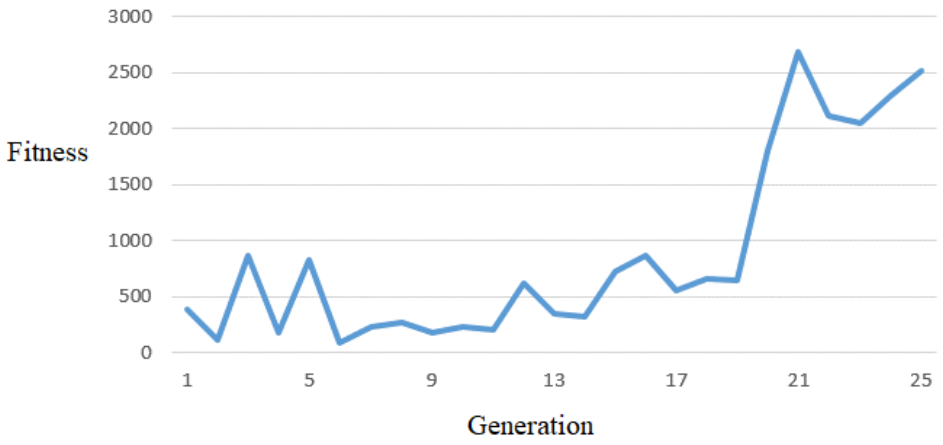


Figure 4.5 - Max Fitness using a steepened curve and modified value

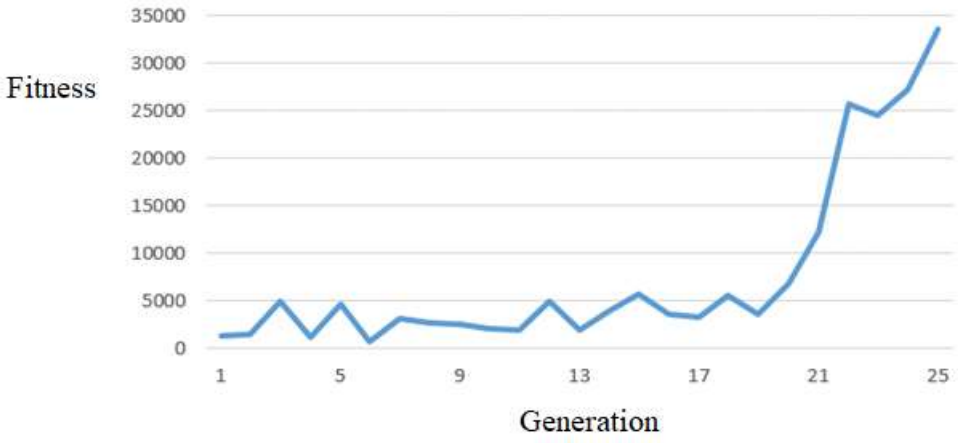


Figure 4.6 - Total Fitness using steepened curve and modified values

4.5.2 Initializing Object Test

The following test is done to generate an enemy object in the game. When the user clicks the mouse a collision object is generated.

Figure 4.7 - Collison Object Generation Test

4.5.3 Grid Detector Testing

The following test is done to check if the Grid data is being read and is being inserted into the Grid matrix where the collision object is represented by -1.

```
[[0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0]]
[[0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0]]
[[0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0]]
[[0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0]]
[[0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0]]
[[0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0]]
[[0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0]]
[[0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0]]
[[0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0]]
[[0, -1, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0]]
[[0, 0, 0, 0, 0], [0, -1, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0]]
[[0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, -1, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0]]
[[0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, -1, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0], [0, 0, 0, 0, 0]]
```

Figure 4.8 - Grid Data Tests

4.5.4 Agent Collision Test

The collision test is done with the one of the trained AI Agents. The game generates enemy objects at random positions and after adds a new collision object after every 200 frames. The objective of this test is to see the percentage and how many collisions avoidance that occur with the AI Agent up to 300 collision objects.

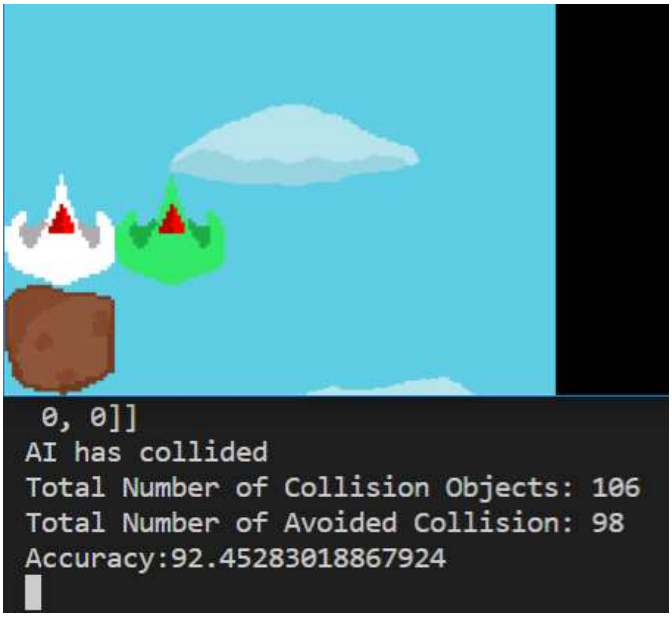


Figure 4.9 - Collison Detection Accuracy Test

Table 4.1 - Accuracy Test Data

Test Number	Total Collisions Avoided
1	287
2	279
3	272
4	276
5	275

After testing the accuracy five times we get an average accuracy of 92.58 and the root mean square error of the avoided collision is 22.7

Chapter 5 - Conclusion and Future Enhancement

5.1 Conclusion

For the current progress, the AI agent has managed to learn to avoid most of the collisions in a game environment using the algorithm NEAT which combines the concept of genetic algorithm to evolve the structure of a feed forward neural networks. The sigmoid function was modified and tested along with the normal sigmoid function that showed the improvement during fitness evaluation. A game environment also has been made that consists of two modes where the Agent can play along with the Player Spaceship and can dodge the 2 collision objects for the most part for an accuracy of around 90% and the other mode where the user can click on the mouse to generate collision objects which the AI Agent will have to dodge.

5.2 Future enhancement

Regardless of the limitations, some minor enhancements can be carried out to further enhance this project. Some key future enhancements to the project are listed below:

- Add support for OpenCV so that we don’t have to hard code the reading of the grid data.
- Dynamic Resolution rather than using a static resolution.
- Make Agent a separate library which can be further used on other systems.
- Use GPU Acceleration to compute the neural network faster.

References

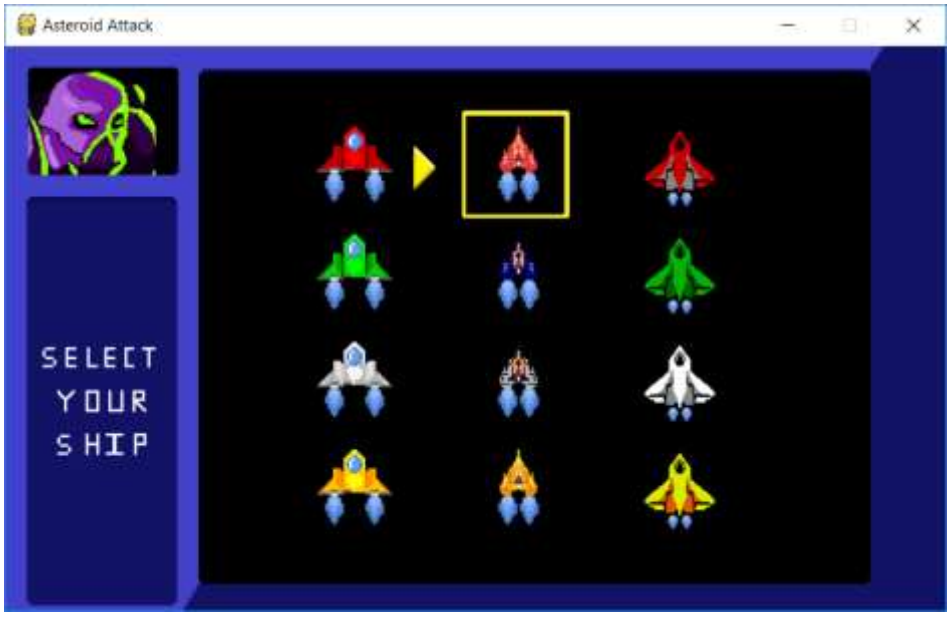
[1] Dr. Rama Kishore, Taranjit Kaur , "*Backpropagation Algorithm: An Artificial Neural Network Approach for Pattern Recognition*", International Journal of Scientific & Engineering Research, Volume 3, Issue 6, June-2012

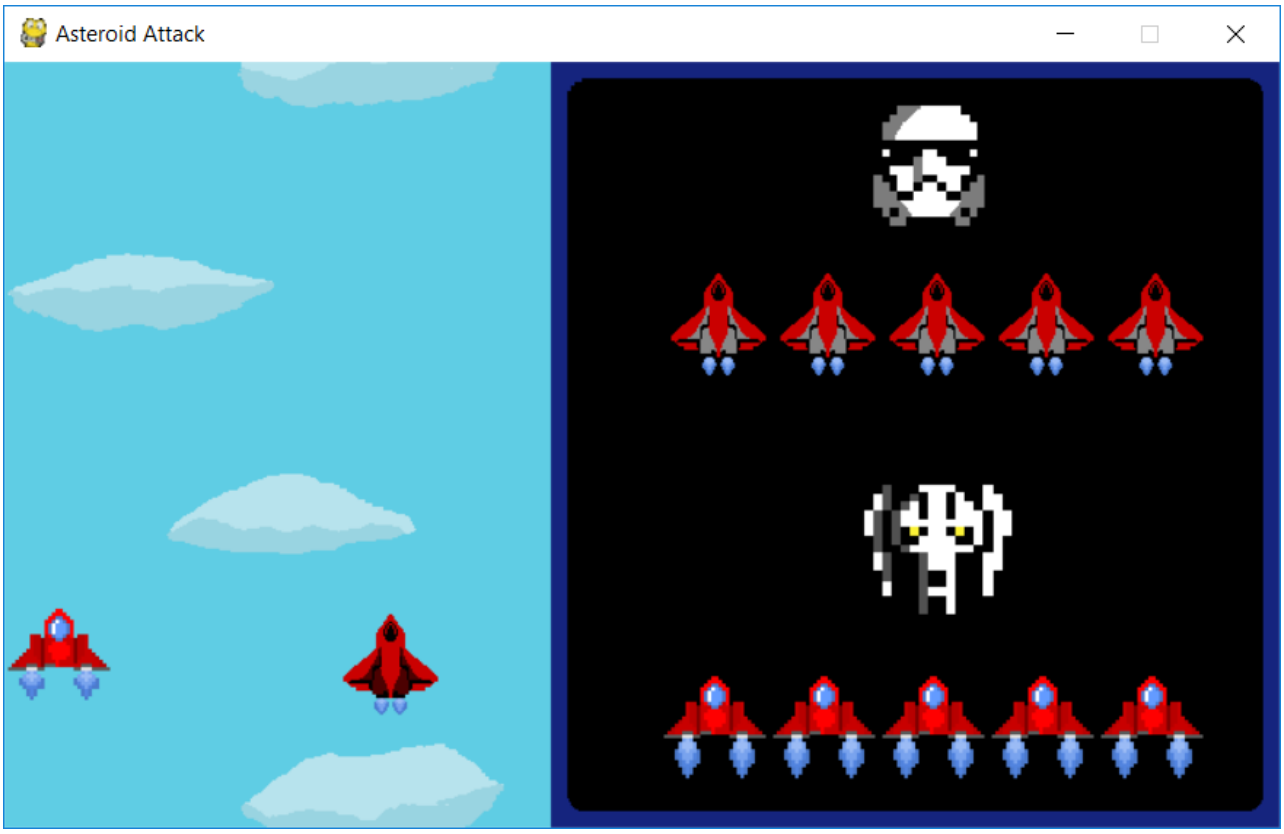
[2] Ms.Dharmistha D.Vishwakarma , "*Genetic Algorithm based Weights Optimization of Artificial Neural Network*", International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering Vol. 1, Issue 3, August 2012.

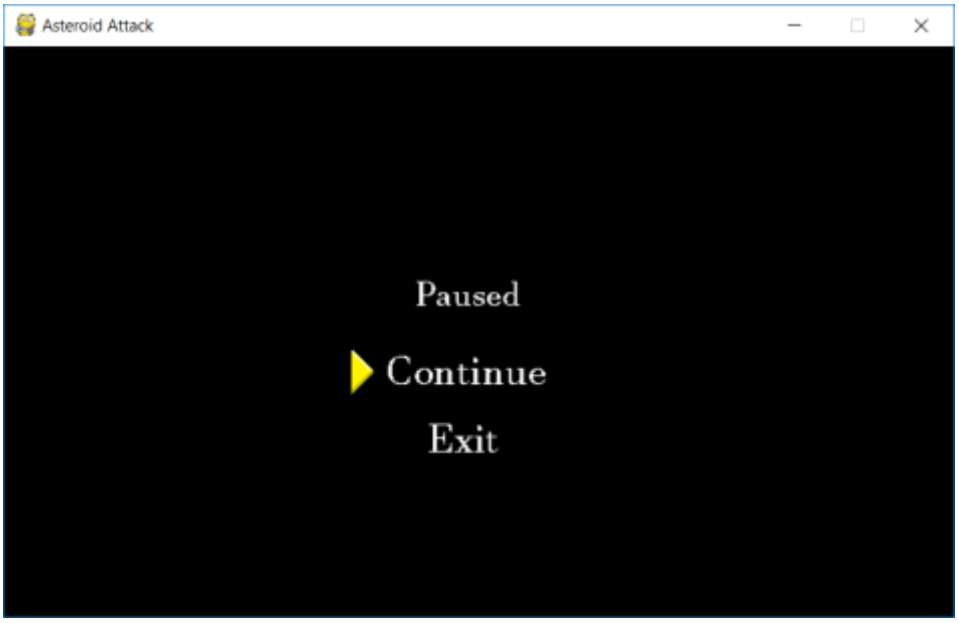
[3]Kenneth O. Stanley and Risto Miikkulainen, "*Evolving Neural Network through Augmenting Topologies*", Mit Press, 2002.

Appendix


Asteroid Attack
Asteroid Attack
Verses Mode
Click Mode
Play along with the AI and see who survives longer


Asteroid Attack
SELECT YOUR SHIP


Asteroid Attack


Asteroid Attack
Paused
Continue
Exit