Implementation of Neural Network in Game Engine to Create Smart Bot and Behavior Analysis

Department of Computer Science and Engineering
BRAC University

Supervisor Dr. Md. Ashraful Alam

Authors

Zamshed Khan Shovon, ID: 13101059, shovon421@gmail.com

Kaisar Ahamed, ID: 13301006, kaisarahamed.007@gmail.com

Tanvir Akram Khan, ID: 13101051, emu.khan66@gmail.com

Saad Ziaul Hasan, ID: 13101265, saad0hasan0@gmail.com

Declaration

Thesis is submitted to the Department of Computer Science and Engineering, BRAC University, Dhaka by the authors for the purpose of obtaining the degree of Bachelor of Engineering in Computer Science and Engineering. We, hereby declare that this thesis is based on results we have found ourselves. Materials and resources taken from any research conducted by other researchers are mentioned in references. Our thesis either in whole or in part, has not been previously submitted for any degree.

Signature of Supervisor

Md. Ashraful Alam, Ph.D

Assistant Professor

Department of Computer Science and Engineering

BRAC University

Signature of Authors

Zamshed Khan Shovon, (ID: 13101059)

Kaisar Ahamed, (ID: 13301006)

Tanvir Akram Khan, (ID: 13101051)

Saad Ziaul Hasan, (ID: 13101265)

Acknowledgement

The basic idea of this innovative research is a brain child of Zamshed Khan Shovon, the lead member of our research from the Department of Computer Science and Engineering of BRAC University. We four- Zamshed Khan Shovon, Kaisar Ahamed, Tanvir Akram Khan and Saad Ziaul Hasan teamed up to carry out this research successfully.

We would like to express our cordial thanks and utmost gratitude to our supervisor; Dr. Md. Ashraful Alam, Assistant Professor, School of Computer Science and Engineering of BRAC University for providing his valuable insight, suggestions and guidance during our research. His continuous supervision, motivation and the urge to manage our required devices in time made us capable of undertaking this mammoth tusk.

We would like to remember the contribution of Abdullah Umar Nasib, for his assistance in our research.

We are thankful to Udemy and Unity for providing us tutorials and an efficient environment to develop our game engine respectively. Without Unity developing the game would be much harder.

Lastly, we convey gratitude from the depth of our hearts towards the faculty members of department of Computer Science and Engineering for providing us with all the knowledge throughout our undergraduate life so that we could do this research.

ABSTRACT

Virtual game players always had a desire for playing with an opponent that acts intelligently like a human. That is why MMO (Massive Multiplayer Online) games have gained huge popularity. Programmers have developed and implemented many systems and algorithms overtime, none came out as successful as Neural Network AI. Artificial Neural Networks (ANN) are computing systems inspired by the biological neural networks that constitute animal brains. Such systems learn (progressively improve performance) to do tasks by considering examples, generally without task-specific programming. We mainly implemented Genetic Algorithm, perceptron algorithm and finally neural network with the help of tensorflow in unity game engine. We finally selected neural network for output efficiency. In this paper, our main focus is to analyze the behavior of the Bots to observe the percentage of efficiency achieved, after the implementation of ANN algorithms in game engine and pointing out the evolving behavior properties. Results from the analysis of our findings can also be helpful for automation and AI development while the whole world is running for these. Neural network algorithms are very complex and a quite time inefficient for video games. But, we implemented ANN in game engine to create smart bots, increasing the efficiency of that algorithm will be another challenge for us. With the help of tensor-flow we made the training process easier, thus making ANN easier to implement in common games.

Keywords: Artificial Neural Networks, Genetic Algorithm, Reinforcement Learning Algorithm, Regression Algorithm, Smart Bot, 2D & 3D Games, Unsupervised Learning, Scarcity of Memory

INDEX

CHAPTER 01
INTRODUCTION

1.1 Introduction

Game bots generally do not use AI algorithms for decision making rather follow a specific set of instructions. In order to make these bots intelligent and eligible for making smart decisions we need to use AI algorithms in game engine. Despite having complexities and complications, we implemented neural network AI in game engine to create smart enemy bot. Artificial neural networks are systems with impressive computing capabilities inspired by the biological neural networks that constitute the human brains. Genetic algorithm is a AI algorithm that acts the same way. In a genetic algorithm, a population of candidate solutions (called individuals, creatures, or phenotypes) to an optimization problem is evolved toward better solutions. Each candidate solution has a set of properties (its chromosomes or genotype) which can be mutated and altered. Traditionally, solutions are represented in binary as strings of 0s and 1s, but other encodings are also possible. Genetic algorithms which have been shown to yield good performance for neural network weight optimization are really genetic hill-climbers, with a strong reliance on mutation rather than hyperplane sampling. Neural control problems are more appropriate for these genetic hill-climbers than supervised learning applications because in reinforcement learning applications gradient information is not directly available. Genetic reinforcement learning produces competitive results with the adaptive heuristic critic method, another reinforcement learning paradigm for neural networks that employs temporal difference methods. The genetic hill-climbing algorithm appears to be robust over a wide range of learning conditions. Therefore, focusing on unsupervised learning had a better chance of yielding fruitful results [1]. We developed 3 3D game sample in unity in order to implement the genetic algorithm on it. For the first one it takes the sessions played by the bots as a test set and then consider all possible reactions for all the given circumstances and initialize it as a population. Then it selects reactions based on fitness and go through the crossover process. Outputs of the crossover process are then mutated. After mutation a new population is generated and fitness tests are run on them again. This process goes on until an optimal solution is chosen as reaction. This way the bot will behave like a human and will no more be as clueless as before.

1.2 Motivation

As we like to play PC games and we were playing for quite a long time, we realized the urge for smart enemy. So we decided to discover the challenges behind the development of a game integrated with the environment that inhabits a smart enemy. Although MMO games have emerged as an alternative where real person plays against real person, it's not very effective for countries with lower internet speed, higher internet charge and inadequate infrastructure to set up servers locally. Moreover continuous transfer of data for gaming can be a reason for network traffic overload. So the solution can be found with the help of smart game bots. These scenarios motivated us to contribute in finding a solution for not only the gaming community but also for all the internet users. A hope that our country might take a baby step through our research toward the generation when we will be able to develop innovative games those are competitive to the global market also motivated us in the process.

1.3 Thesis Contribution

This thesis has an objective of pioneering the era of intelligent PC game development in our country. We aim to assist in overcoming the challenges our developers are facing currently by shedding some light into the rigorous development process. As we analyze the behavior of the bot based on certain parameters, we will be able to point out significant evolution in reactions. Even if the idea of Neural Network AI is not a new one, we go beyond the beginning state in several ways:

We identify appropriate parameters for the gaming environment to retain a standard output from our test and training.

We study the characteristics of the bot's behavior and list out significant improvement through evolution while trained.

1.4 Problem Statement

Neural Network AI is a completely new field of research for us. As there are very few people involved in Neural Network research in our university, it was tough to take guidance. Developing the game engine also requires high performance PC and software. As we started sorting out algorithms based on their efficiency, we were faced with the uncertainty of their

performance as there is no fixed standard. Making the test linear enough to assess the output reactions was a challenge too.

1.5 Solutions

We went through online tutorials and courses. We also tested multiple algorithms to establish a custom scale of performance during the research although not mentioning all of them to reduce redundancy. The game inhabits a 3D environment yet game rules are kept simple to help us in assessing the output.

1.6 Methodology

We analyzed the algorithms and sorted them based on efficiency and practicality of this research. We developed the game in Unity and implemented Neural Network AI algorithms in the game engine. Later we let the bot play to determine and analyze its reactions upon challenging circumstances. As we let it play more and more, the bot got trained and its efficiency and fitness for survival was increased.

1.7 Data

For our research, there was no data available. That is why we prepared our own dataset through the tests. We gave parameters as input and collected output reactions as data for our research.

1.8 Thesis outline

Chapter 1 gives a brief overview of our research, our estimated goal and what we have gained.

Chapter 2 discusses the literature review and background study of our thesis, what properties we considered and how they work and more.

Chapter 3 discusses the chosen methodology, and the reasons behind choosing it.

Chapter 4 discusses comparative study and analysis of the resulting output we found. The training and evolution stages are also briefly discussed in this chapter.

Chapter 5 ends our paper with conclusion and proposed future work for the system.

CHAPTER 02

LITERATURE REVIEW

2.1 3D Game Theory

Three Dimensional games follow coordinate system of basic mathematics. These models are all about the representations of shapes in a 3D space, with a coordinate system used to calculate their position. WebGL uses the right-hand coordinate system — the x axis points to the right, the y axis points up, and the z axis points out of the screen, as seen in the diagram below.

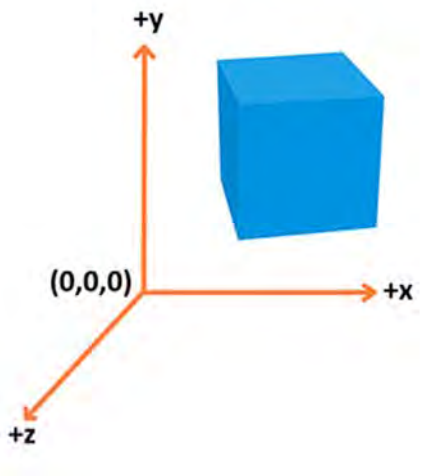

Figure 1: Coordinate System

2.1.1 Objects

Diverse kinds of Articles are manufactured utilizing vertices. A Vertex is a point in space having its own particular 3D position in the facilitate framework and typically some extra data that characterizes it. Every vertex is described by these attributes:

- **Position**: Identifies it in a 3D space (x, y, z).
- **Color**: Holds an RGBA value (R, G and B for the red, green, and blue channels, alpha for transparency — all values range from 0.0 to 1.0).
- **Normal:** A way to describe the direction the vertex is facing.

- **Texture**: A 2D image that the vertex can use to decorate the surface it is part of instead of a simple color.

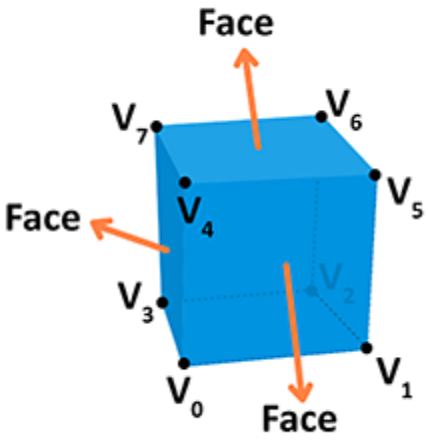

Figure 2: 3D Object

Here, for instance a face of the given shape is a plane between vertices. For instance, a 3D square has 8 diverse vertices (focuses in space) and 6 unique faces, each built out of 4 vertices. An ordinary characterizes which way the face is coordinated in. Likewise, by associating the focuses we're making the edges of the solid shape. The geometry is worked from a vertex and the face, while material is a surface, which utilizes shading or a picture. On the off chance that we associate the geometry with the material we will get a work [2].

2.1.2 Rendering Pipeline

In this very approach images are prepared and processed to be visible on the screen. This methodology takes three dimensional objects and calculates the fragments and renders them into two dimensional pixels. The process can be shown as followings:

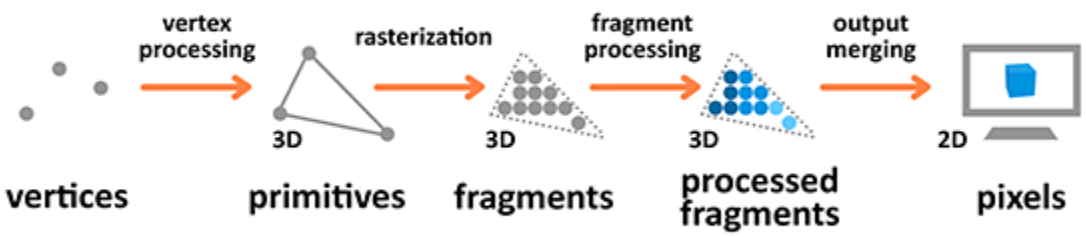

Figure 3: Rendering Pipeline

Terminology used in the diagram above is as follows:

- A **Primitive**: An input to the pipeline — it's built from vertices and can be a triangle, point or line.
- A **Fragment**: A 3D projection of a pixel, which has all the same attributes as a pixel.
- A **Pixel**: A point on the screen arranged in the 2D grid, which holds an RGBA color.

2.2 Reinforcement Learning

Reinforcement Learning (RL) refers to a kind of Machine Learning method in which the agent receives a delayed reward in the next time step to evaluate its previous action. It was mostly used in games (e.g. Atari, Mario), with performance analysis on per to per or even with human. Recently, as the algorithm evolves with the combination of Neural Networks, it is capable of solving more complex tasks [3]. Some of the earlier researchers of AI used to believe that RL might be usefully reproduced in machines. The beginning of the era started in 1951, when a Harvard graduate, M Minsky built a machine that used a simple form of reinforcement learning to mimic a rat learning to navigate a maze. We have evidenced a few successes over the next decades too.

However, there would be large portions tests to the current Reinforcement Learning research. Firstly, it is excessively memory consuming with store qualities in each of the states, since those issues could be pretty unpredictable. This includes looking under quality close estimation techniques, for example, Decision Trees or Neural Networks. There is needed a

significant number outcome of presenting, and investigate tries will minimize their effect on the nature of the result. [4]

2.3 Unity Game Engine

Unity Game Engine is known as a third party game making platform. According to the Unity press release, over 45% of today's developers are somehow involved using this platform where Unreal Engine 4, another third party game making platform with huge developer appeals, being used by 17% of market share. With the manta of 'democratize game development', the unity game engine has already succeeded with flying colors all over the world.

The Unity platform is free of cost and simple to install and the features are supported regardless the OS of the user. Even while installing, the user might choose the customized features based on his requirements of work. Along with the documentation files in their website, a number of tutorials in available in the net to get started with Unity and building your own game. The Unity API is known as one of the most well documented online documentation and provides you insight into operations and features works here. So, the problems someone might face working with Unity will not waste much time to solve.

Although the Unity Game Engine framework was developed using one of the most used programming language, C++; the end users can interact with both C++ & JavaScript. For this very reason, we have incorporated unity to work on and using as the game developing platform. That gives us easy access and much freedom to design and making our game, the way we want.

2.4 Existing Works

Neural Networking is such a wonderful thing to work on for smarting any game enemy bot. The outcomes of this implementation is noticeable and surely be further workable. There have been few works with the game bot to make them more efficient and also with the neural networking. However, only few researchers had focused on implementing neural networking in game engines to make them smarter. Previously a group of researchers proposed an approach to briefly describe the scope and background of AAIA'17 Data Mining Challenge: Helping AI to

Play Hearthstone in the context of a more general project related to the development of an AI engine for video games, called Grail [05].

CHAPTER 03

ALGORITHM AND WORKING PRINCIPLES

3.1 System Design

In our proposed technique, we can divide the whole system into two different segments named ‘Developing’ and ‘Training’. Firstly, in the developing segment, we build our systems based on the model we proposed from zero. We designed our models based on the parameters and rules we choose to work on. To be done with that we had to go through various steps and levels of development. Primarily, we selected a suitable and fruitful algorithm initially which was decision tree algorithm, can be considerable as a predictive modeling approach. Decision trees where the target variable can take continuous values (typically real numbers) are called regression trees. First step of this process is initialization. The second segment which was training the system involves creating multiple section sets, crossing over them and mutating as well as re-evaluating the section sets for fitness checking with the help of the algorithm [6].

3.2 Parameter Selection

We needed to set some parameters to make decisions on different situations. Based on the parameters we developed the games and made playable. By taking decisions of these situation, the intelligent bot will learn and be better after every fitness testing.

3.2.1 Survival Time

The intelligence development of proposed games will rely on the measurement of the survival time of the bot in every try. It will keep track of the survival timing in both games to use the improved time in every fitness testing and accepting the developed methodology of better survival. How much time the bot can stay before get eliminated, will be measured here which will be resultant reward and to proceed in next stage also.

3.2.2 Distance Covered

The total distance covered by the bot by walking in segments will be added. The bot cannot stop walking permanently as it will eliminate the chance of falling over the edge. The best timing will improve the efficiency level of the bot with the techniques used after every fitness testing. The walking time is more vital for the first game as the points will be adding points in the game and the bot will be able to cover more distance if it doesn't fall from the edge too early.

3.2.3 Threat Identification

In the second game we will measure how fast the bot can separately identify the lethal object being thrown at it from the harmless ones. Faster identification will also result in longer survival time. As the bot will not have any previous given instruction about the shape of the objects, the color of it or even any hint about being harmless or harmful of the objects for the bot, it's one of the major identifier to justify the performance of the bot in second game.

3.2.4 Dodging Accuracy

There will be multiple shapes of objects, being thrown at the bot in the second game. As the bot identifies the lethal object, how accurately it can dodge it, is another parameter that we have taken into consideration. Better dodging will result in better points and failing will result in poor points.

3.2.4 Tackling accuracy

In the final game we implemented a bot which will avoid enemy stream coming randomly from a direction and it will come up with a strategy of its own. The strategy totally depends in which environment the bot is trained and how many times the steps are executed for the training. The training strategy significantly varies if we max out the max steps as we did in our game.

3.3 Test Environment

The game environments of the proposed models are simple. In the first game, bots will spawn in a large playfield. They are allowed to walk any direction they want. If a bot reaches the edge of the playfield and does not stop, it will fall over [Figure 4]. So the bots have to decide whether to walk straight or turn left / right or stop. In the next phase, next generation of bots will respawn and restart.

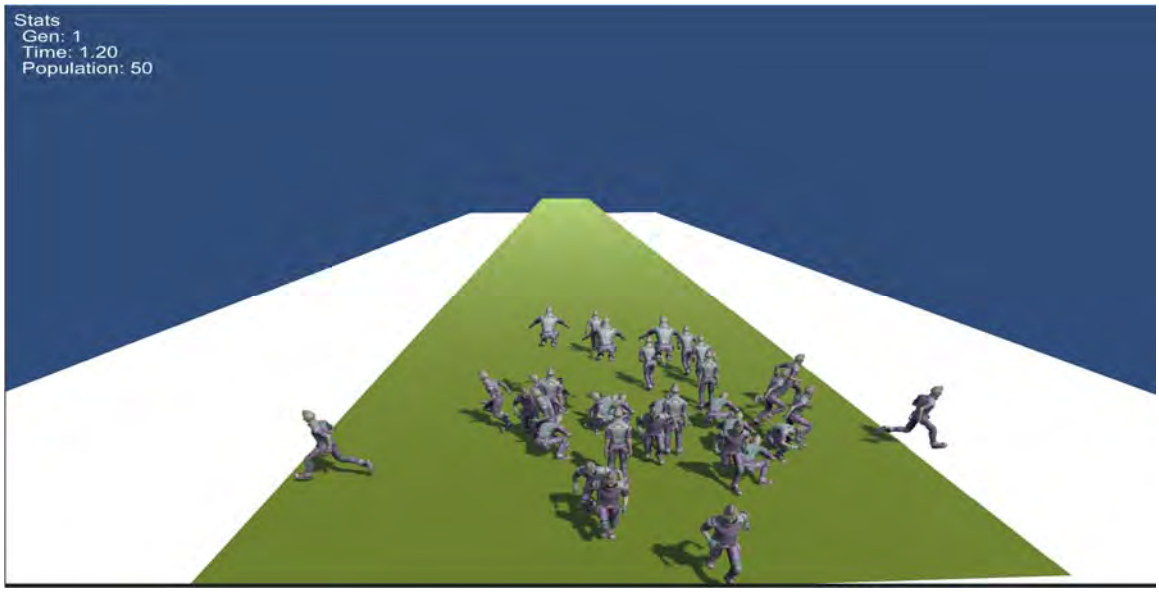

Figure 4: Game Environment

In the second game, different shapes of object will be thrown toward a bot. Only one type of object is lethal and the rest are harmless. The bot only has to dodge the lethal objects, not the harmless ones. Getting hit by the lethal object will result in lost points and dodging them will result in gaining points. The set of rules has been discussed briefly in the test game rules section.

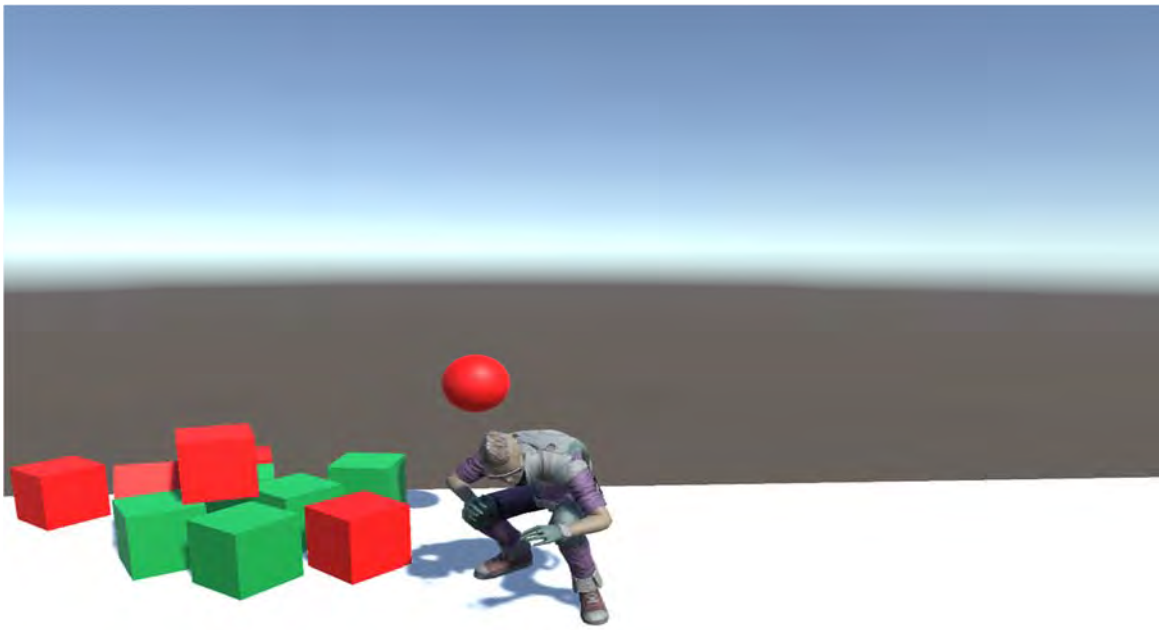

Figure 5: Game Environment

3.4 Test Game Rules

We had to set up few predefined rules to play the game. Based on those conditions, the game will move forward.

I. In the first game, the bots will be given 5 seconds for each generation.
II. Bots are allowed to act anyway they want (they had 6 options/input options).
III. In the second game, the bot have no hint beforehand about the lethal object.
IV. There is no pattern in which the objects will be thrown, balls will be thrown manually.
V. In the third game bit will have to avoid hitting on top and bottom bar as well as the running stream of enemies, it will have to find a safe place for itself

3.5 Algorithm

We used three algorithms in our system which are genetic algorithm, a search algorithm that iteratively transforms a set of strings, each with an associated fitness value, called a population into a new population of offspring objects using the Darwinian principle of natural selection and using operations such as crossover and mutation, secondly we used perceptron algorithm which works like a single neuron iterating data that has similarities with reinforcement learning and thirdly we used neural network algorithm with the help of tensorflow with 6 state input and 4 output.

3.5.1 Genetic Algorithm

I. Defining the fitness function f(x) according to the problem definition.
II. Generating the random population of n chromosomes – each chromosome being the potential solution.
III. Evaluating the fitness f(x) of each chromosome x in the population.
IV. Repeating the following steps to create the new population of chromosomes using loops:
    A. Select some parent chromosomes from a population according to their fitness to form mating pool.
    B. Mate the selected chromosomes as per given crossover probability to form new offsprings.
    C. Mutate new chromosomes as per given mutation probability.
    D. Replace the old population of chromosomes with the new population.
V. Checking if the maximum number of generations is reached, then stop, and return the best solution.
VI. Go to step 3

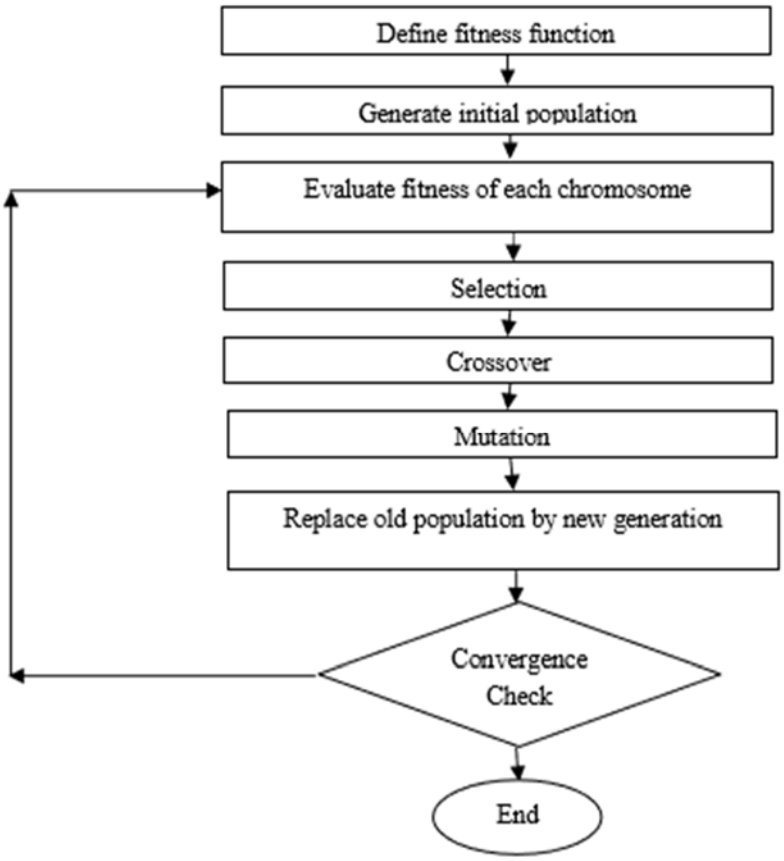

Figure 6: Basic flowchart of Genetic Algorithm

The used genetic algorithm works step by step in the above mentioned way [7]. In the section, the process has been described in details. First of all, the fitness function has to be defined for the algorithm. Right after that the population for the game environment has to be defined. For our first game, the population is fifty.

After setting the population and specify all the parameters, the algorithm start testing the fitness of the bot in after every single generation. Based on the fitness testing the algorithm decides the life time remaining for the bot and related fitness stuffs.

a. In the fitness evaluation section, first stage is to select the bot and to find out the specific parameters to test the fitness. A GA uses fitness as a discriminator of the quality of solutions represented by the chromosomes in a GA population. The selection component of a GA is designed to use fitness to guide the evolution of chromosomes by selective pressure.

Chromosomes are therefore selected for recombination on the basis of fitness. Those with higher fitness should have a greater chance of selection than those with lower fitness, thus creating a selective pressure towards more highly fit solutions. Selection is usually with replacement, meaning that highly fit chromosomes have a chance of being selected more than once or even recombined with themselves. The traditional selection method used is Roulette Wheel (or fitness proportional) selection. This allocates each chromosome a probability of being selected proportional to its relative fitness, which is its fitness as a proportion of the sum of fitness of all chromosomes in the population. There are many different selection schemes. Random Stochastic Selection explicitly selects each chromosome a number of times equal to its expectation of being selected under the fitness proportional method. Tournament Selection first selects two chromosomes with uniform probability and then chooses the one with the highest fitness. Truncation Selection simply selects at random from the population having first eliminated a fixed number of the least fit chromosomes. A full account of the various selection schemes used in the literature and guidance on when it is most appropriate to use them.

b. Then the specified parameters cross matches the values of the bot currently. Based on the testing, the algorithm decides the lifestyle of the bot.

c. In the next stage the algorithm updates the old population with new population as it may lose number of population during the crossover fitness testing.

d. This procedure continues unless the algorithm does get the expected performance for it and itself update the best possible output after every single stage.

3.5.2 Perceptron algorithm

I. Starting the algorithm
II. Identifying the current state
III. Choosing an action
    A. Finding error in the first step
    B. Updating weights and bias value based on output
    C. Going back to step 3.
IV. Determining the next step
    A. Checking the Reach goal
    B. Program ends if reached goal or go back to step 2 if false.

3.6 Workflow

The following working flowchart in Figure 2 we have followed for developing the proposed model.

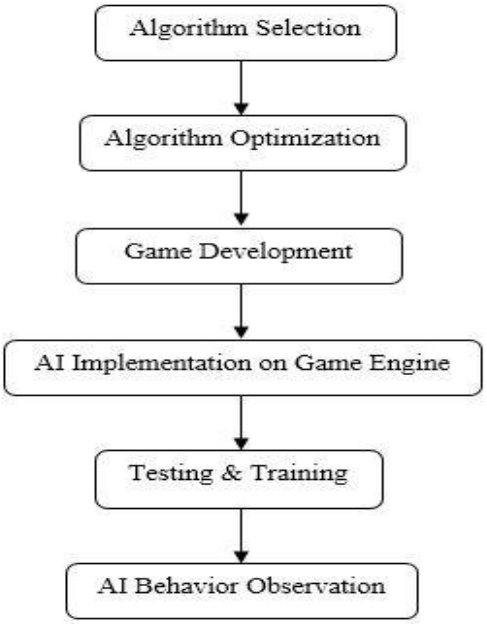

Figure 7: The working flowchart

In the used methodology which is already shown in the above Figure 2, we have designed the system in the following systematic ways. Where we started with selecting the algorithm and followed by the algorithm selection, Algorithm Optimization and designing the game was the most crucial phrases in the early stages of the research. After being done with the game development, we implemented Artificial Intelligence on the game and following that Testing and Training phases has been done. And finally, the behavior of the bot has been analyzed. The steps has been discussed below.

1. **Algorithm Selection:** At the very early stage our first challenge was to decide which algorithm to use for our program. From many of the existing algorithms in Neural Network, we choose genetic algorithm based on the scenario for our game after

measuring the possible outcomes and benefits, constraint can be occur using the algorithms.

2. **Algorithm Optimization:** A Genetic Algorithm is constructed from a number of distinct components. This is a particular strength because it means that standard components can be re-used, with trivial adaptation in many different Genetic Algorithms, thus easing implementation. The main components are the chromosome encoding, the fitness function, selection, recombination and the evolution scheme.
3. **Game Development:** We have used unity game engine for developing the game for our research. Starting from the very beginning, the environment and the bots were designed according to the need of our project.

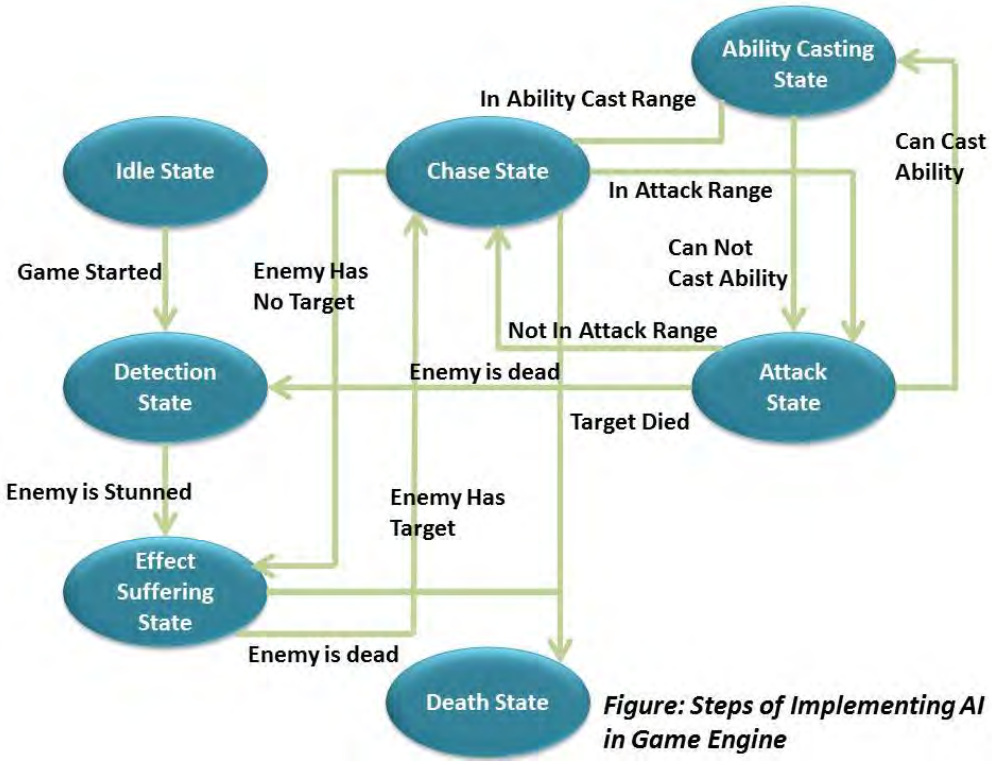

Figure 8: Steps of implementing AI in game engine.

4. **AI Implementation on game Engine:** In this stage, the game was integrated with the AI algorithm following the steps mentioned above. Few others algorithms were in our plan to implement. However, using Genetic Algorithm gave the best accuracy for the performance and helped in self-learn of the bot.

5. **Testing and Training**: In this very stage, for the first bot we tested the walking distances covered by the bot and the time spent on the pitch. And for the second game, the harmful object avoidance accuracy and the gained point was kept in the record.
6. **AI Behavior Observation**: With no previous given instruction, how the bot learn to walk in safe directions and to stay on the pitch more time was the first segment to observe to justify whether the bot is learning in every generation or not. However, for the second game, it was crucial for the bot to know which object of what shape or color to avoid and which one to collect to gain point. We observed how the bots learn from the progress of the game.

CHAPTER 04
RESULT ANALYSIS

4.1 Reaction analysis

In this stage we have taken the analysis of the reaction of bots in every stages of the game to see how it reacts in stages [8]. In first stage, with the increase of number of times of attempts, how the bots reacted has been noted and analyzed the progress with the span of time. At the stage, the bot has been dropped in the walking segment for the first time and the bots starts running regardless the direction. As the bots are walking first time, they do not have any previous data and so far best path in Figure 9. According to the parameter the time the bots can survive before dropping from any edge is important. Therefore, at the early stages, the parameter's values was not satisfactory and maximum of the bots were getting fallen from any of the edges. As a result, they are scattered and going in different direction in generation 1 with the population of 50.

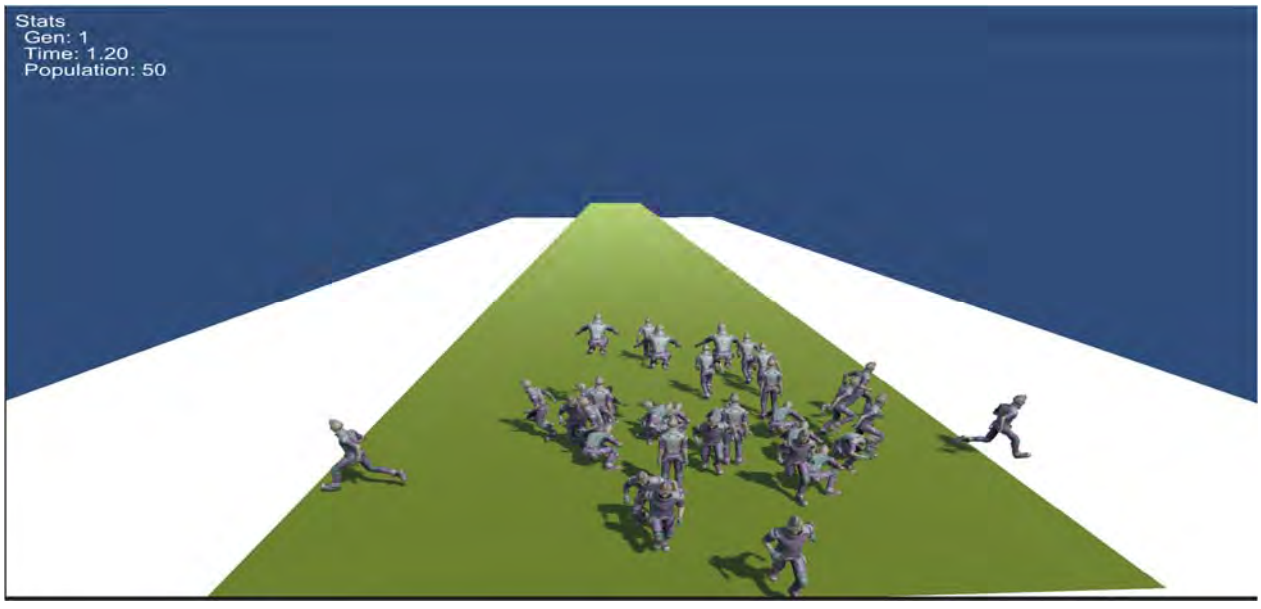

Figure 9: Bots movement without any previous knowledge

However, in Figure 10 the movement of the bots are more likely group wise. In the generation 03, with the same number of population the bots reacts differently. Unlikely,

generation 01, this time the bots makes decision based on the previous walk experience and makes groups and some of the bots stays more time in the walkway then previous generation.

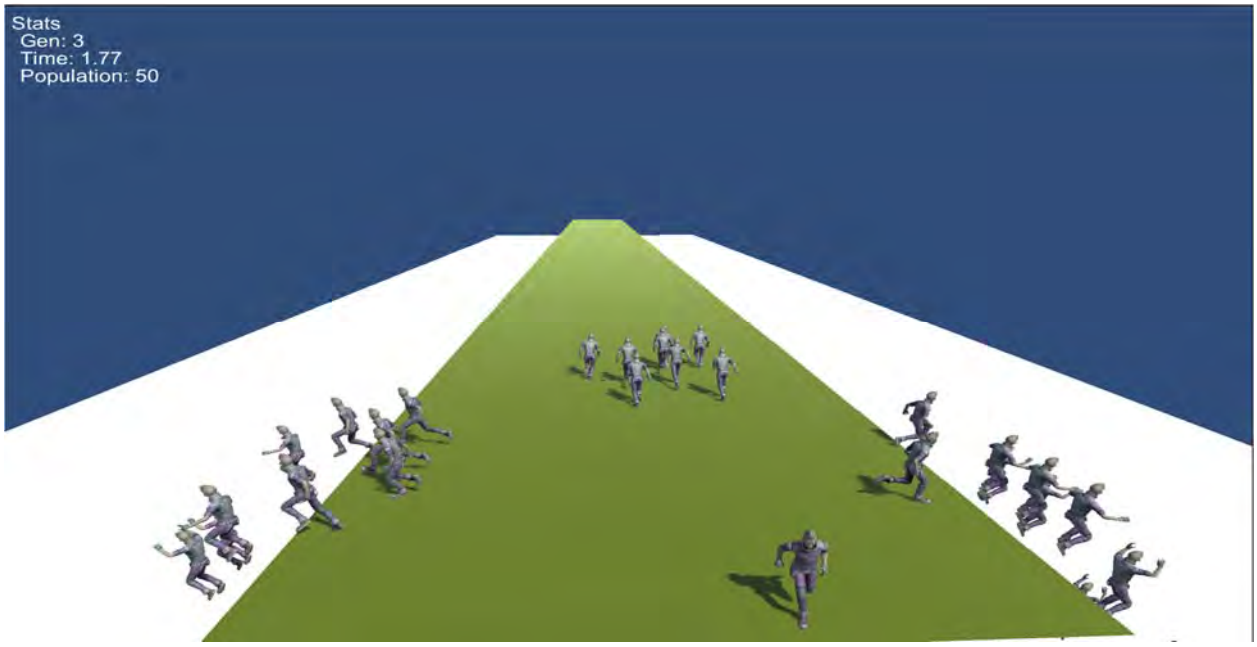

Figure 10: Bots movement in generation 03 with previous learn.

After few more generation, the bots learnt by themselves from the previous runs and keeping the best path in learn in Figure 11. Therefore, this time the bots goes in same direction and stays more time in the walkway comparing the previous generation. As we see in the figure 05, the bots are going in one direction that's because the current best path shows the bots the path. Following this path, they are surviving more time. Therefore, we can come to the conclusion that these bots are learning in generation to generation.

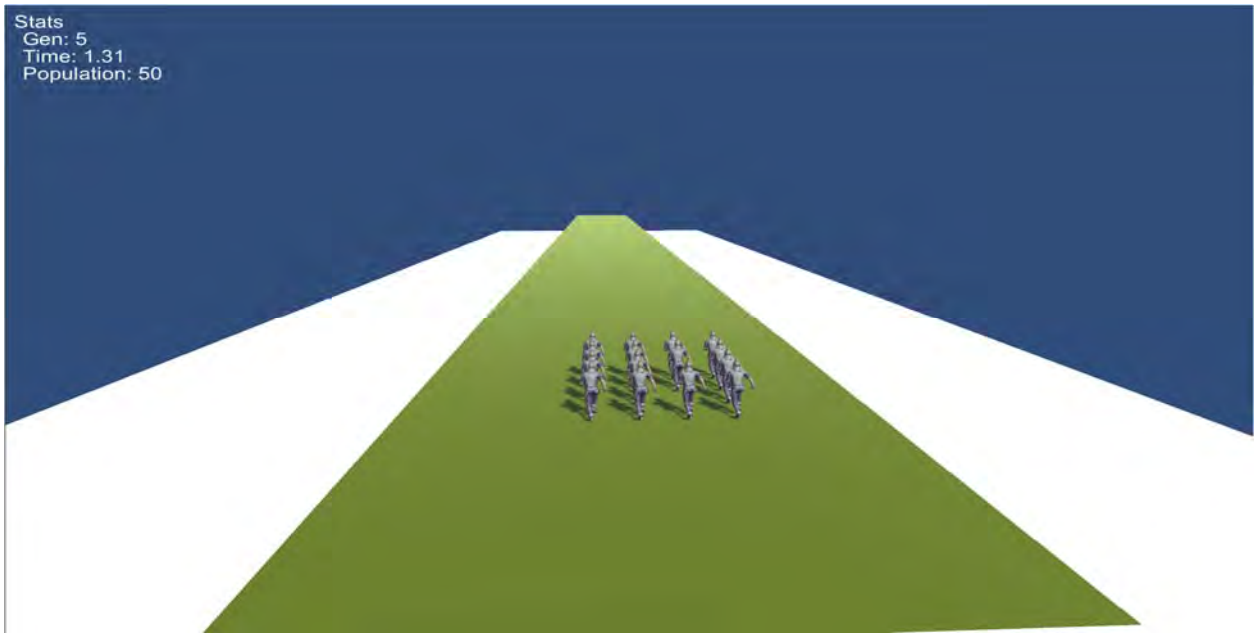

Figure 11: After few generations the bots find a direction to walk to survive more.

In the second round of the game, where the bots had no previous clues about the shapes of the objects been thrown towards them. All the objects will not be harmless. Some of the objects will be harmful and will cause decrease the life capacity of the bot whereas rest of the bots will be adding extra game point for the bot. With the span of time and stages, the bot will learn which object to avoid and to take.

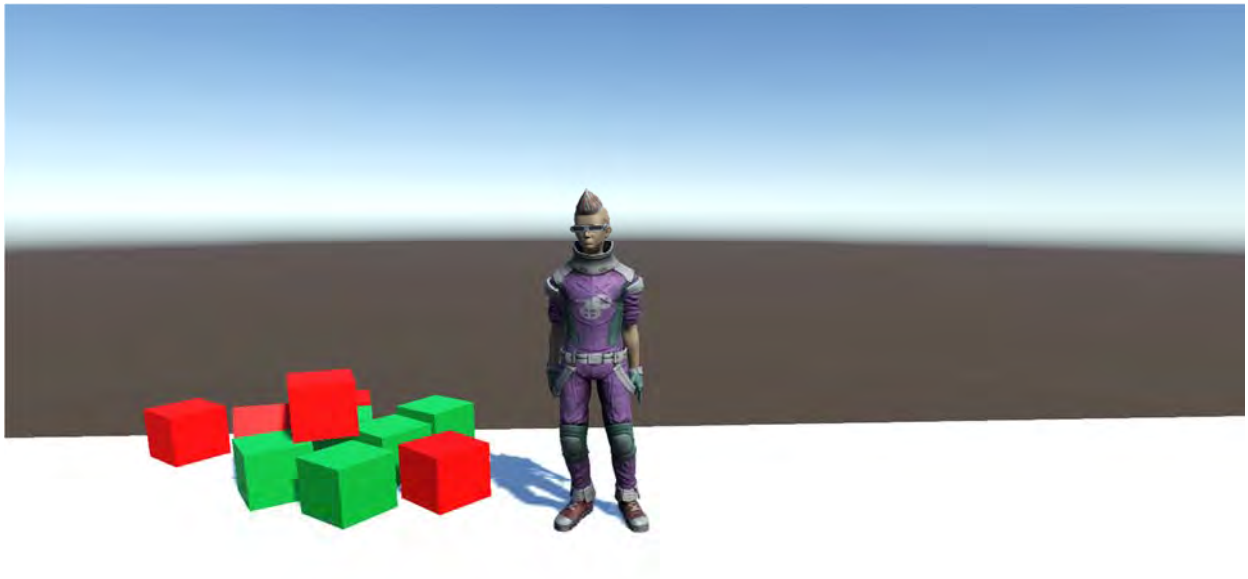

Figure 12: Bot standing beside the harmless objects in second game.

The figure 12 shows that the bot is standing beside the thrown objects towards it and doesn't have clues about which one to take and which one to accept. Whereas in the next figure (figure 07), it's shown that the bot is not moving or saving himself from the thrown object towards it as it already knew the object is not harmful and will add points to the game scores.

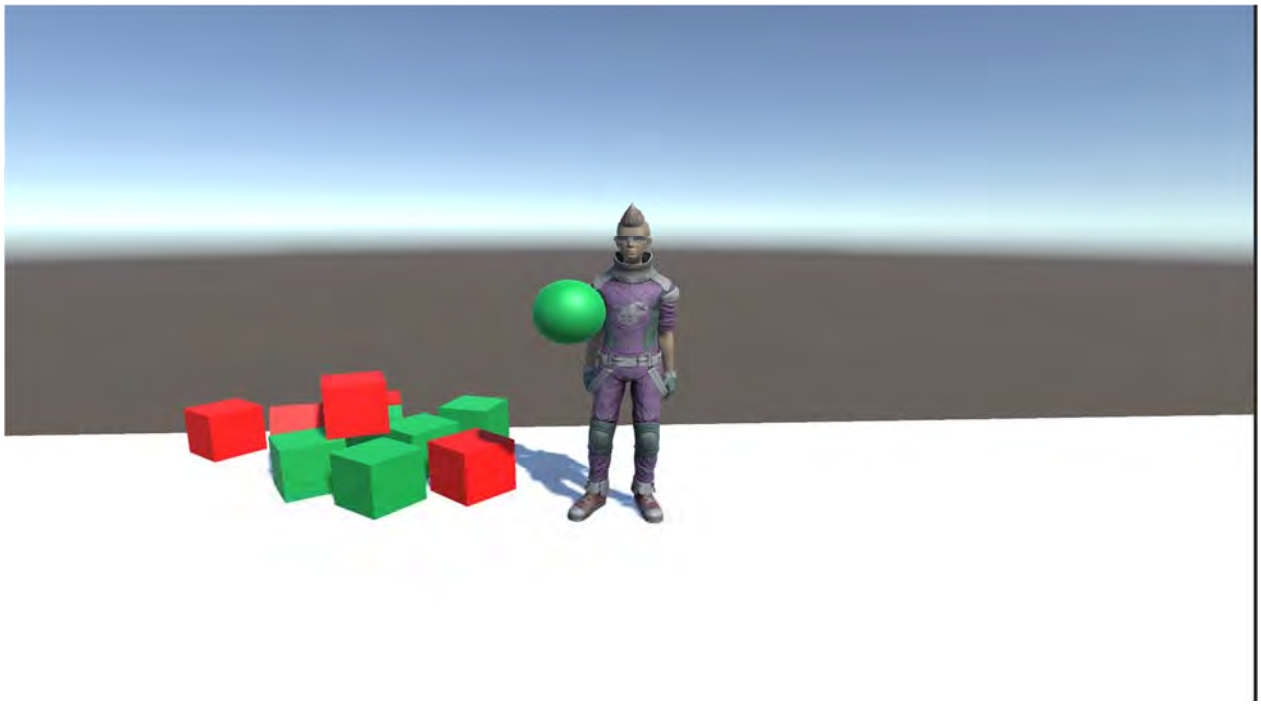

Figure 13: Bot receiving the harmless objects

In the next scenario in figure 13, the bot tries to avoid the object which is red and harmful for the bot. However, at the beginning of the game, the bot did not have any idea about being harmless or harmful of the bots as no instruction was provided to the bot about the different shaped objects thrown toward it. With the expansion of time, it did self-learn and could able to recognize the differences between the harmful and harmless objects and started acting accordingly.

In the third game we trained a bot to avoid streaming enemy and top and bottom bar to survive and for coming up with a strategy of its own. Mainly inputs are the hit distance from four directions and their velocity. Outputs are its own velocities with the help of which it moves. The more the bot is trained increasing the max steps the more accurate strategy it find to tackle the bot and find a safe place for itself. It is important to note that the training environment is very important because the more challenging the environment would be the better the training would be.

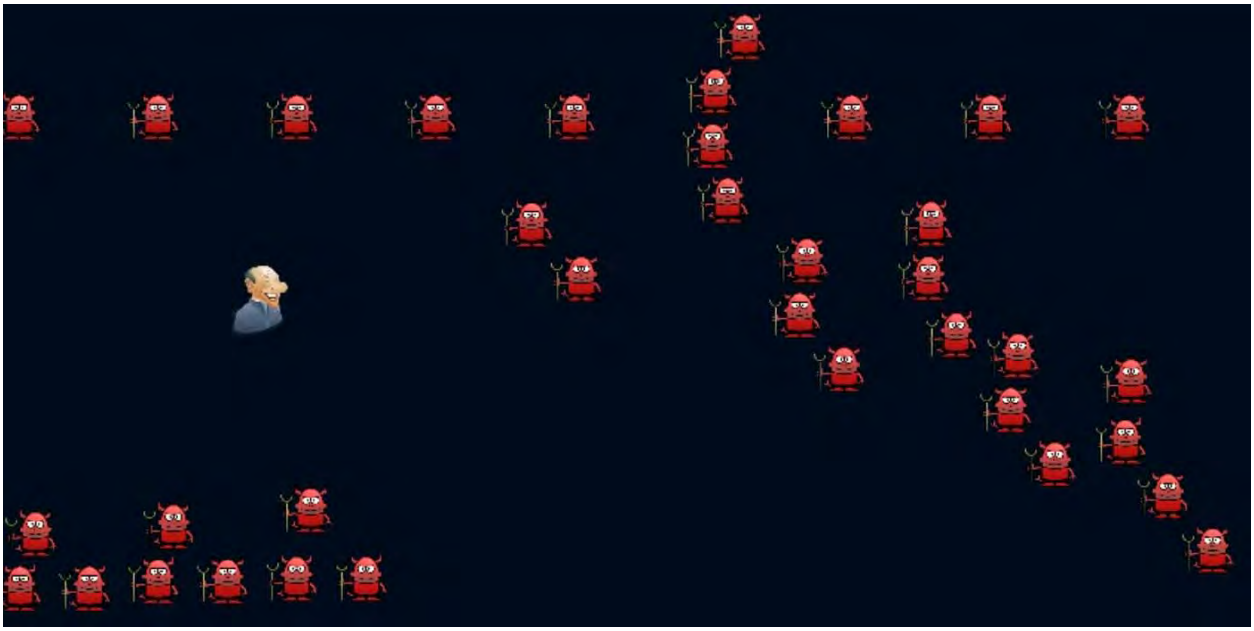

Figure 14: Bot avoiding monsters

4.2 Reaction properties

The reaction of the bot for this game has been measured in two different segment. Based on the time and covering distance for the first game where for the second game the reaction has been measured based on how the bot dodge on the different types of objects thrown towards the bot.

a. First Game Properties

For the first game, there is two different parameters. Number one is the covered distance by the bot in every generation. With the increase of time and number of runs, the distance covered by the bot gets increases.

Measurement of the time until the bot dropped into to fallen from any of the edges is the second parameter for first game. With no knowledge or previous learnt path, the amount of spend time might be low however, that gets increases with every generation.

b. Second Game Property

The only property to measure the bot reaction of second game is the scores it can gain in one lifetime in the game. In this stage, the bot loses it lifetime by getting hit through harmful

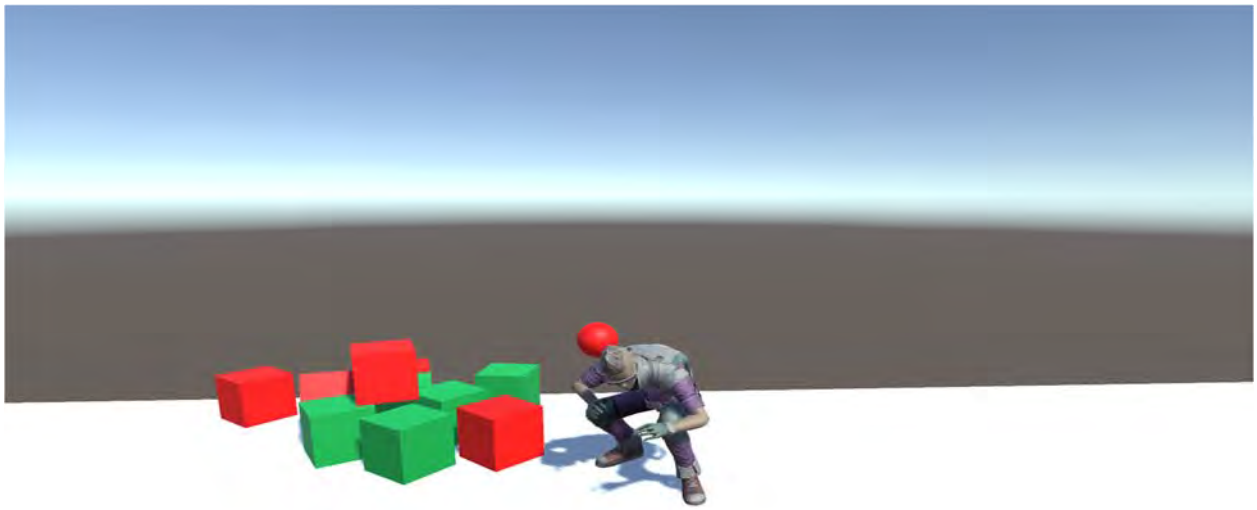

Figure 15: Bot's react on thrown object

objects thrown towards it. And the fact to remember that, at the beginning, the bot is not aware of any of the object's size, shape, color or even harmful or harmless. This is how the reaction of the bot has been measured based on the above mentioned properties.

c. Third Game Properties

In the third game the accuracy of the tackling is checked by observing the game play. Also the reward increase with increase of steps is analyzed. The more the rewards the better the training turns out to be.

4.3 Bot's Fitness analysis

At the beginning of the first game, we have seen that, as the bots didn't have any knowledge about which action they should take. So as a result, each of them started to random tasks. But after the 5th generation, we have seen that they have self-trained themselves and started to go straight. Here to find out the best fit value, we can consider the time they have survived and the distance they have passed. The bot that has passed more distance must have survived longer. So,

the bot which has passed the highest distance and survived the highest the time will be the fittest of all.

In the second game, we are considering three parameters: weight1, weight 2 and bias weigh. When the bot will learn which object is harmful and which is harmless, the bias weigh will be same as the real weight and after that, they will keep giving a fixed value in the further generation. The total error will be zero. That fixed value will be considered as the fittest value of the bot.

Total error becomes 0; where weight and bias was same.

4.4 Regression analysis

Table 1: Dodge Ball Data

1 UnityEngine.Debug:Log(Object)
W1: -0.73950719833374 W2: 0.679593086242676 B: -0.313713431358337 UnityEngine.Debug:Log(Object)
TOTAL ERROR: 0 UnityEngine.Debug:Log(Object)
0 UnityEngine.Debug:Log(Object)
W1: -0.73950719833374 W2: 0.679593086242676 B: -0.313713431358337 UnityEngine.Debug:Log(Object)
W1: -0.26049280166626 W2: 0.679593086242676 B: 0.686286568641663 UnityEngine.Debug:Log(Object)
TOTAL ERROR: 1 UnityEngine.Debug:Log(Object)
1 UnityEngine.Debug:Log(Object)
W1: -0.26049280166626 W2: 0.679593086242676 B: 0.686286568641663 UnityEngine.Debug:Log(Object)

TOTAL ERROR: 0 UnityEngine.Debug:Log(Object)
1 UnityEngine.Debug:Log(Object)
W1: -0.26049280166626 W2: 0.679593086242676 B: 0.686286568641663 UnityEngine.Debug:Log(Object)
W1: -0.26049280166626 W2: 0.679593086242676 B: -0.313713431358337 UnityEngine.Debug:Log(Object)
TOTAL ERROR: 1 UnityEngine.Debug:Log(Object)
0 UnityEngine.Debug:Log(Object)
W1: 1.26049280166626 W2: 0.679593086242676 B: 0.686286568641663 UnityEngine.Debug:Log(Object)
W1: 1.26049280166626 W2: 0.679593086242676 B: -0.31373431358337 UnityEngine.Debug:Log(Object)
TOTAL ERROR: 2 UnityEngine.Debug:Log(Object)
W1: 1.26049280166626 W2: 0.679593086242676 B: -0.31373431358337 UnityEngine.Debug:Log(Object)
1 UnityEngine.Debug:Log(Object)
W1: 1.26049280166626 W2: 0.679593086242676 B: -0.31373431358337 UnityEngine.Debug:Log(Object)
TOTAL ERROR: 0 UnityEngine.Debug:Log(Object)
W1: 1.26049280166626 W2: 0.679593086242676 B: -0.31373431358337 UnityEngine.Debug:Log(Object)
TOTAL ERROR: 0 UnityEngine.Debug:Log(Object)

Table 2: Function Data

W1: 0.482770413160324 W2: -0.970821857452393 B: -0.0900560617446899 UnityEngine.Debug:Log(Object)
W1: 0.482770413160324 W2: 0.0291781425476074 B: 0.90994393825531 UnityEngine.Debug:Log(Object)
TOTAL ERROR: 1 UnityEngine.Debug:Log(Object)
W1: 0.482770413160324 W2: 0.0291781425476074 B: -0.0900560617446899 UnityEngine.Debug:Log(Object)
W1: 0.482770413160324 W2: 1.02917814254761 B: 0.90994393825531 UnityEngine.Debug:Log(Object)
TOTAL ERROR: 2 UnityEngine.Debug:Log(Object)
W1: 0.482770413160324 W2: 1.02917814254761 B: -0.0900560617446899 UnityEngine.Debug:Log(Object)
TOTAL ERROR: 0 UnityEngine.Debug:Log(Object)
Test 0 0:0 UnityEngine.Debug:Log(Object)
Test 0 1:1 UnityEngine.Debug:Log(Object)
Test 1 0:1 UnityEngine.Debug:Log(Object)
Test 1 1:1 UnityEngine.Debug:Log(Object)

Total 50 populations generate every time. Generation reset time 5 second. Fitness function for each bot is distance travelled +time alive. The bot will die if falls of the green platform. Mutation occurs 1 in 100.

Table 3: Generations and performance

Generation 1:	7 goes to left 7 goes to right 7 goes to back 7 goes to forward 12 crouches 10 jumps
Generation 2:	10 goes to left 8 goes to right 8 goes to back 24 goes to forward 0 crouches 0 jumps
Generation 3:	10 goes to left 8 goes to right 4 goes to back 28 goes to forward 0 crouches 0 jumps
Generation 4:	5 goes to left 0 goes to right 0 goes to back 44 goes to forward 0 crouches 1 jumps
Generation 5:	0 goes to left 1 goes to right 0 goes to back 48 goes to forward 0 crouches 1 jumps

Generation 6:	0 goes to left 0 goes to right 0 goes to back 50 goes to forward 0 crouches 0 jumps

In the third game we can see the rewards increasing with steps.

```
Step: 10000. Mean Reward: 5.087421383647795. Std of Reward: 3.6290536586906463.
Step: 20000. Mean Reward: 5.497986577181202. Std of Reward: 2.472844999000461.
Step: 30000. Mean Reward: 25.355555555555693. Std of Reward: 23.144095052571796.
Step: 40000. Mean Reward: 65.89333333333317. Std of Reward: 44.56061739049425.
Step: 50000. Mean Reward: 56.37222222222256. Std of Reward: 13.374719334772674.
Saved Model
Step: 60000. Mean Reward: 45.00476190476229. Std of Reward: 6.468640557149358.
Step: 70000. Mean Reward: 44.26500000000037. Std of Reward: 5.728898236135891.
Step: 80000. Mean Reward: 73.46666666666846. Std of Reward: 102.34919095376465.
Step: 90000. Mean Reward: 62.124999999999226. Std of Reward: 59.8685591525273.
Saved Model
Step: 110000. Mean Reward: 136.59444444441806. Std of Reward: 386.9103305133369.
Step: 120000. Mean Reward: 58.364705882352496. Std of Reward: 49.36422783371729.
Step: 130000. Mean Reward: 63.140000000000235. Std of Reward: 21.943983230032654.
Step: 140000. Mean Reward: 55.516666666667156. Std of Reward: 7.888405697252426.
Step: 150000. Mean Reward: 57.029411764706325. Std of Reward: 9.703290206442647.
Saved Model
Step: 160000. Mean Reward: 52.41578947368469. Std of Reward: 7.793739941226514.
Step: 170000. Mean Reward: 59.681250000000375. Std of Reward: 12.167488378358609.
```

CHAPTER 05

Future Work and Conclusion

5.1 Future of AI in FPS games

So when can we expect to play a game where our enemy bots will pass the Turing Test? The answer is it may be not very soon but we are taking big steps toward that. In recent games the use of AI is mostly done with supervised learning. We know that in supervised learning the training data has very clear solution to problems as if God has given us the solutions to all problems that we can follow. But the real world is different. We continuously search for new solutions to new problems which we might have no idea about. But eventually we solve most problems very well as human even though no solution set is given to us. The situation in FPS games is exactly the same. The enemy bot might not know what might happen to the environment or what type of new situations might arrive. So it needs to decide in an unsupervised manner and should be bold enough to try new solutions. AI in current games are not bold enough to take decisions like this. Memory is one of the main reasons for this. In order to run a situation to an algorithm we need to dig into dependencies of a lots of data from environments and after that if we need to run neural network or genetic algorithms with those data to come to a decision, then the process would take a lot of memory. Right now our technologies don't provide everyone a very cheap solution for that. In games we always have memory limitations and memory greedy algorithms can't handle the game environments for normal user PCs.

But the good news is that technology is advancing very fast. Price of memory is decreasing at a very fast rate. That means the possibilities of using unsupervised AI algorithms are becoming more and more open. And if normal user pc can handle those large chunks of data with those memory greedy algorithms then our enemy bots can be as smart as us. Now here we are not talking about just AI in all games. We are talking about AI in fps games where the enemy bots will create a charm that will hold us in disbelief that the bot cannot be not human. Obviously in our present games there is no enemy that pass the Turing test. It will only be possible if we let out bots think like we think. For example in an fps game like call of duty, inside the game environment we see a lot of things like a box, a wall, a piece of rock, fogs, a tree

etc. But we cannot interact with everything we see in current games neither every object we see in games matter while an important decision is taken by us. For example hiding behind a tree in a game like call of duty might not make us invisible from enemy. Why are we coming up with this limitation? Why can't we make every object matter which we can see in game screen? There is only one reason. Memory. Every game must be memory efficient so some objects in games are just for beautifications. They just don't play any roles in the games. Now imagine if we could actually use and interact every object in an fps game what we could actually do. Imagine what our enemy bots could do. Our enemy bots can even make a new weapon out of a stone after trying and figuring out that stone can hurt us. So finding new possibilities and making new decisions and taking new risks should be part of our future AI. In fact in genetic algorithm the phase of mutation inspire us to take new decisions out of just using fitness function. Just patterns don't give us a very effective solutions. For example, I can intentionally follow a wrong pattern to give the enemy a wrong idea about what I am going to do. So if an enemy totally depends on finding patterns it might be easily deceived by a clever player. We need an enemy which doesn't depend on patterns instead depends on fitness function or reward based system where taking a decision will result in getting reward for longer time. In fps games it's not possible to use anything like min max algorithm since the number of variables is huge. But if memory problem is solved we can even use the min max algorithms with probabilities for taking a very cautious decision in which way our enemy can get super smart may be even smarter than human.

So we can see memory efficiency is the major thing for games AI. It's just that we need to figure out how to make reinforcement learning more memory efficient with the available technologies. Also we need to train our bots efficiently in a specific way so that it becomes smart in the specific things rather trying to be smart in everything and waste memory for no reason. For example, what do we humans do if we want to train a soldier be a sniper for a battle? We train him with sample shooting object to increase his accuracy but not limited to that. Their endurance to pain is increased in order to live in a harsh environment. Their skills of making objects from new things are sharpened for their survival. They are always made biased for taking any decisions example US soldier hating and hurting Russian soldiers. Also they may be made smarter in math for accurate and quick measurement skill in times when it's needed. There is no training given to a soldier about poetry or art as it will count as waste of time for their purpose even though poetry and art are not valueless knowledge. The condition is same for training an AI

soldier. If we want to make an AI soldier we need to train it with some basic skills like how to shoot or how to run or walk, what is its purpose, where it should be biased about etc. In our current games trainings like these are not performed and our current bots are not ready for an unknown world situations yet. But with adequate technologies and memory efficiency we can achieve that state easily.

Another thing that we will have to make our enemy bot do is being able to take different decisions sometimes. We all have different personalities. Some of us are more daring and some of us are more fearful in taking decisions. An efficient decision comes from the balance of these two. So when we teach an AI, we cannot teach it to ignore the risk of any decisions and be super bold, neither to be so fearful that it doesn't even do anything. So during the training we need to set its biased value somewhere in the middle so when it comes to chances or probabilities, it does not get too afraid or too bold to take a decision. So we can see that we need to use multiple algorithms for taking multiple types of decisions in a game. Simple decision trees won't give us very convincing performance from a bot which pretends to be a human. So memory and time cost of all these algorithms are the walls on our way to use these algorithms widely in game AI. With our present rate of technological advancement we can hope to break these walls pretty soon.

5.2 Conclusion

In our behavior analysis result, we have seen exponential rate of improvement in the reaction of the bots in order to survive the game environment. Implementing something so complex was not an easy task, especially when there were such conditions involved. In order to achieve this goal, we had to be careful about the memory consumption and time complexity. We also needed to carefully select and monitor, if necessary regulate different properties of the specified algorithms. There are different ways to optimize an algorithm including switching blocks and importing faster methods. Definitely, we did not expect to create a super intelligent bot that can surpass human intelligence through self-driven evolution but we sure expected it to make the game experience more enjoyable while creating the dilution of a human player.

REFERENCES

1. A Panorama of Artificial and Computational Intelligence in Games; Georgios N. Yannakakis ; Julian Togelius https://ieeexplore.ieee.org/document/6855367/
2. Game Mechanics in the Design of a Collaborative 3D Serious Game; Kimmo Oksanen, Raija Hämäläinen; https://doi.org/10.1177/1046878114530799
3. Exploring Q-Learning Optimization in Traffic Signal Timing Plan Management - Scientific Figure on ResearchGate. Available from: https://www.researchgate.net/Q-Learning-algorithm-flow-chart_fig1_224256471 [accessed 22 Mar, 2018]
4. Introduction to Various Reinforcement Learning Algorithms. Part I (Q-Learning, SARSA, DQN, DDPG). Available from: https://towardsdatascience.com/introduction-to-various-reinforcement-learning-algorithms-i-q-learning-sarsa-dqn-ddpg-72a5e0cb6287 [accessed 22 Mar, 2018].
5. Reinforcement Learning. Available from: http://reinforcementlearning.ai-depot.com/ [accessed 22 Mar, 2018].
6. Markov games as a framework for multi-agent reinforcement learning; Michael L.Littman; https://doi.org/10.1016/B978-1-55860-335-6.50027-1
7. Genetic algorithm learning and evolutionary games; Thomas Riechmann; https://doi.org/10.1016/S0165-1889(00)00066-X
8. A Janusz, T Tajmajer, M Swiechowsk, 'Helping AI to Play Hearthstone: AAIA'17 Data Mining Challenge', Proceedings of the Federated Conference on Computer Science and Information Systems, vol. 11, pp. 121–125.