CHAPTER 1

INTRODUCTION

The main aim of this project is to develop a Genetic Algorithm based approach for constructing an AI which can play the Flappy Bird game much more efficiently as compared to a normal human being. Flappy Bird is a game where the player tries to make the bird navigate through the maximum number of pipes while it's alive. If the bird hits any pipe, or the ground, it dies and the game ends. Pipes can come up randomly and can have variable heights and spacing between them. The player only has two options: - first, to make the bird move upwards, and the second, do nothing and consequently let the bird fall downwards to the ground.

The player tries to navigate the bird safely through maximum possible number of pipes. Each pipe crossed successfully adds to the max score of the generation. Since we solve this using Genetic Algorithms (GA) here's a brief introduction to it: Genetic algorithms are adaptive heuristic search algorithms based on the evolutionary ideas of natural selection and genetics. As such they represent an intelligent exploitation of a random search used to solve optimization problems. Although randomized, these are by no means random, instead they exploit historical information to direct the search into the region of better performance within the search space. Each problem using GA requires a fitness function which measures the quality of the solution toward an optimization problem.

The basic techniques of these are designed to simulate processes in natural systems necessary for evolution, especially those follow the principles first laid down by Charles Darwin of survival of the fittest. Genetic Algorithms allow us to explore a space of parameters to find solutions that score well according to a fitness function.

CHAPTER 2

LITERATURE SURVEY

The related work in this domain is primarily by Google Deep mind are able to successfully train agents to play the Atari 2600 games using deep reinforcement learning, surpassing human expert-level on multiple games. These works inspired this project, which is heavily modeled after their procedure. They use a deep Q-network (DQN) to assess the Q-function for Q-learning and also use trained experience replay to de-correlate learning.

Their approach is naturally state-of-the-art and was the main catalyst for deep reinforcement learning, after which many papers tried to make improvements. We have read papers of other similar projects based on the idea of building an AI that can master the game and found them extremely intriguing and helpful and have tried a simpler approach since we are beginners in the field of artificial intelligence and found that genetic algorithms and multi-layer neural network combination is best for these sort of optimization problems.

2.1 Existing System and Related Work:

- **Google DeepMind**

  https://deepmind.com/research/publications/playing-atari-deep-reinforcement-learning

  The related work in this area is primarily by Google DeepMind are able to successfully train agents to play the Atari 2600 games using deep reinforcement learning, surpassing human expert-level on multiple games. These works inspired this project, which is heavily modeled after their approach. They use a deep Q-network (DQN) to evaluate the Q-function for Q-learning and also use experience replay to de-correlate experiences.

- **Deep Reinforcement Learning for Flappy Bird by Kevin Chen**
  http://cs229.stanford.edu/proj2015/362_report.pdf
  This paper uses screenshots as input and grayscales the picture before applying the algorithm. It clears background objects and uses Q learning to implement the sameproject.

- **Playing Flappy Bird AI by Naveen Appiah**
  http://cs231n.stanford.edu/reports/2016/pdfs/111_Report.pdf

  This paper uses deep reinforcement learning. It is a way to teach the agent to make the right decisions under uncertainty and with very high dimensional inputbymaking it experiencing scenarios

2.2Objective

We had seen videos of Genetic Algorithm based Approaches applied to solving optimization problems, mostly games like Mario, Atari Deep Mind, etc. We were quite influenced by these approaches and hence we decided to implement a similar thing on the Flappy Bird Game, as it is not much of a complicated game, and hence apt for beginners like us. Hence, we decided to make an AI for this game using Genetic Algorithms and Neural Networks.

2.3Scope

The scope of this project is to demonstrate neuro evolution by creating an AI bot that improves in a iterative fashion. It also shows how genetic algorithm-based approaches can be used to solve problems where there is no definite solution or goal and the input parameters somewhat fluctuate.

This project can also be used for educational purposes for demonstration on how AI is superior in complex tasks and how this type of technological combination of neural networks and genetic algorithm can be used to create AI bots for other games as well making the experience much more interactive.

CHAPTER 3

PROPOSED SYSTEM

3.1 Overview of Genetic Algorithms

Genetic Algorithms mimic the survival of the fittest, in a way that, generations are maintained, and from one generation, only those characters are able to pass to the next generation which have survived/performed well in that generation.A genetic algorithm is a search heuristic that is inspired by Charles Darwin's theory of natural evolution. This algorithm reflects the process of natural selection where the fittest individuals are selected for reproduction in order to produce offspring of the next generation.The evolution usually starts from a population of randomly generated individuals, and is an iterative process, with the population in each iteration called a generation. In each generation, the fitness of every individual in the population is evaluated; the fitness is usually the value of the objective function in the optimization problem being solved. The more fit individuals are stochastically selected from the current population, and each individual's genome is modified (recombined and possibly randomly mutated) to form a new generation. The new generation of candidate solutions is then used in the next iteration of the algorithm. Commonly, the algorithm terminates when either a maximum number of generations has been produced, or a satisfactory fitness level has been reached for the population. Each generation is characterized by a set of strings, which are analogous to the chromosomes in the DNA. Crossover, Mutations and other biological effects occur and in turn affect the offspring in the next generation. The same holds true for a genetic algorithm too. Each individual represents a point in a search space and a possible solution.

The individuals in the population are then made to go through a process of evolution. In biology, individuals compete for resources, some of them win and others lose. The winning individuals are likely to contribute more share to the offspring as compared to the losing ones. Offspring inheriting features from two winning individuals has a good chance of being better than each of its parent. The logic remains the same her also.

Some of the important terms related to Genetic Algorithms are:

- **Search Space–**

  Creating a population of nodes inside the search space where every node of the population makes a decision to provide the solution to the problem statement is the first step. In our project, the search space would be the population of agents created that have random weights and make random decisions at the start and further refine them towards the goal state.If we are solving some problem, we are usually looking for some solution, which will be the best among others. The space of all feasible solutions (it means objects among those the desired solution is) is called search space (also state space). Each point in the search space represent one feasible solution. Each feasible solution can be "marked" by its value or fitness for the problem. We are looking for our solution, which is one point (or more) among feasible solutions - that is one point in the search space.

  The looking for a solution is then equal to a looking for some extreme (minimum or maximum) in the search space. The problem is that the search can be very complicated. One does not know where to look for the solution and where to start. There are many methods, how to find some suitable solution (ie. not necessarily the best solution), for example hill climbing, tabu search, simulated annealing and genetic algorithm. The solution found by this methods is often considered as a good solution, because it is not often possible to prove what is the real optimum.

  The search space can be whole known by the time of solving a problem, but usually we know only a few points from it and we are generating other points as the process of finding solution continues.

- **Selection**

  Selection is the process of picking the best performing or the fittest agent, so to speak from the search space by comparing the fitness values of the entire search space and save it. The fitness value parameters are instrumental for defining the metric with which each agent of the search space is evaluated and selected. As you already know from the GA outline, chromosomes are selected from the population to be parents to crossover. The problem is how to select these chromosomes. According to Darwin's evolution theory the best ones should survive and create new offspring. This is not particular method of selecting parents. Main idea of this selection is that big part of chromosomes should survive to next generation.

  GA then works in a following way. In every generation are selected a few (good - with high fitness) chromosomes for creating a new offspring. Then some (bad - with low fitness) chromosomes are removed and the new offspring is placed in their place. The rest of population survives to new generation.

There are many methods how to select the best chromosomes, for example roulette wheel selection, Boltzmann selection, tournament selection, rank selection, steady state selection and some others.

- **Crossover**

  Crossover is the process of creating a new individual by using best performing individual of previous generation and similarly generating an entire population of agents. Since the new search space is derived from the previous best agents it will generally be better than the previous generations. The crossover operator is analogous to reproduction and biological crossover. In this more than one parent is selected and one or more off-springs are produced using the genetic material of the parents. Crossover is usually applied in a GA with a high probability. In this section we will discuss some of the most popularly used crossover operators. It is to be noted that these crossover operators are very generic and the GA Designer might choose to implement a problem-specific crossover operator as well.

- **Mutation**

  Mutation is one of the imperative steps of genetic algorithm as it ensures variety in the decision making of each individual to be distinct from one another. It is performed after crossover and after every new generation of the search space is created for the next cycle. In simple terms, mutation may be defined as a small random tweak in the chromosome, to get a new solution. It is used to maintain and introduce diversity in the genetic population and is usually applied with a low probability – pm. If the probability is very high, the GA gets reduced to a random search.

  Mutation is the part of the GA which is related to the "exploration" of the search space. It has been observed that mutation is essential to the convergence of the GA while crossover is not. Without mutation we won't have variety and it is important that each agent is mutated individually rather than mutating the entire population at once. It ensures that the entire new generation doesn't make the same decisions and has diversity. In this section, we describe some of the most commonly used mutation operators. Like the crossover operators, this is not an exhaustive list and the GA designer might find a combination of these approaches or a problem-specific mutation operator more useful.

3.2 Architecture

Our neural network architecture uses a feed forward neural network which can also be called a multi layered perceptron with two layers containing one hidden layer and one output layer. The simplest kind of neural network is a single-layer perceptron network, which consists of a single layer of output nodes; the inputs are fed directly to the outputs via a series of weights. The sum of the products of the weights and the inputs is calculated in each node, and if the value is above some threshold (typically 0) the neuron fires and takes the activated value (typically 1); otherwise it takes the deactivated value (typically -1). Neurons with this kind of activation function are also called artificial neurons or linear threshold units. In the literature the term perceptron often refers to networks consisting of just one of these units. A similar neuron was described by Warren McCulloch and Walter Pitts in the 1940s.

This class of networks consists of multiple layers of computational units, usually interconnected in a feed-forward way. Each neuron in one layer has directed connections to the neurons of the subsequent layer. In many applications the units of these networks apply a sigmoid function as an activation function.

The universal approximation theorem for neural networks states that every continuous function that maps intervals of real numbers to some output interval of real numbers can be approximated

Arbitrarily closely by a multi-layer perceptron with just one hidden layer. This result holds for a wide range of activation functions, e.g. for the sigmoidal functions.

Multi-layer networks use a variety of learning techniques, the most popular being back-propagation. Here, the output values are compared with the correct answer to compute the value of some predefined error-function. By various techniques, the error is then fed back through the network. Using this information, the algorithm adjusts the weights of each connection in order to reduce the value of the error function by some small amount. After repeating this process for a sufficiently large number of training cycles, the network will usually converge to some state where the error of the calculations is small. In this case, one would say that the network has learned a certain target function. To adjust weights properly, one applies a general method for non-linear optimization that is called gradient descent. For this, the network calculates the derivative of the error function with respect to the network weights, and changes the weights such that the error decreases (thus going downhill on the surface of the error function). For this reason, back-propagation can only be applied on networks with differentiable activation functions.

There are five input nodes in the neural network that are fed to the hidden layer which has eight hidden nodes which use sigmoidal activation function to process the inputs from the input layer and pass it to the output layer which uses softmax activation function which would give a probabilistic result helping the agent in decision making for maneuvering through the obstacles.

Dense Layers:

Dense layer is the regular deeply connected neural network layer. It is most common and frequently used layer. Dense layer does the below operation on the input and return the output. A dense layer represents a matrix vector multiplication. (Assuming your batch size is the values in the matrix are the trainable parameters which get updated during back propagation.

Input Layer:

The input layer of a neural network is composed of artificial input neurons, and brings the initial data into the system for further processing by subsequent layers of artificial neurons. The input layer is the very beginning of the workflow for the artificial neural network. Artificial neural networks are typically composed of input layers, hidden layers and output layers. Other components may include convolutional layers and encoding or decoding layers.

One of the distinct characteristics of the input layer is that artificial neurons in the input layer have a different role to play – experts explain this as the input layer being constituted of "passive" neurons that do not take in information from previous layers because they are the very first layer of the network. In general, artificial neurons are likely to have a set of weighted inputs and function on the basis of those weighted inputs – however, in theory, an input layer can be composed of artificial neurons that do not have weighted inputs, or where weights are calculated differently, for example, randomly, because the information is coming into the system for the first time. What is common in the neural network model is that the input layer sends the data to subsequent layers, in which the neurons do have weighted inputs.

Hidden Layer:

In neural networks, a hidden layer is located between the input and output of the algorithm, in which the function applies weights to the inputs and directs them through an activation function as the output. In short, the hidden layers perform nonlinear transformations of the inputs entered into the network. Hidden layers vary depending on the function of the neural network, and similarly, the layers may vary depending on their associated weights.

Hidden layers, simply put, are layers of mathematical functions each designed to produce an output specific to an intended result. For example, some forms of hidden layers are known as squashing functions. These functions are particularly useful when the intended output of the algorithm is a probability because they take an input and produce an output value between 0 and 1, the range for defining probability.

Output Layer:

Hidden layers, simply put, are layers of mathematical functions each designed to produce an output specific to an intended result. For example, some forms of hidden layers are known as squashing functions. These functions are particularly useful when the intended output of the algorithm is a probability because they take an input and produce an output value between 0 and 1, the range for defining probability. A typical traditional neural network has three types of layers: one or more input layers, one or more hidden layers, and one or more output layers. Simple feed-forward neural networks with three individual layers provide basic easy-to-understand models. More sophisticated, innovative neural networks may have more than one of any type of layer – and as mentioned, each type of layer may be built differently. A traditional artificial neuron is composed of some weighted inputs, a transformation function and activation function corresponding to the biological neuron's axon. However, output layer neurons may be designed differently in order to streamline and improve the end results of the iterative process.

In a sense, the output layer coalesces and concretely produces the end result. However, to understand the neural network better, it is important to look at the input layer, hidden layers and output layer together as a whole.

Activation functions:

Activation function(s) used on hidden layers are mostly the same for all hidden layers. It's unlikely to see ReLU used on the first hidden layer, followed by a Hyperbolic tangent function — it's usually ReLU or tanh all the way. It states that we need to apply a standard exponential function to each element of the output layer, and then normalize these values by dividing by the sum of all the exponentials. Doing so ensures the sum of all exponential values adds up to 1.

Sigmoid :

The main reason why we use sigmoid function is because it exists between (0 to 1). Therefore, it is especially used for models where we have to predict the probability as an output. Since probability of anything exists only between the range of 0 and 1, sigmoid is the right choice. The function is differentiable. That means, we can find the slope of the sigmoid curve at any two points. The function is monotonic but function's derivative is not. The logistic sigmoid function can cause a neural network to get stuck at the training time.

Softmax:

Softmax turn logics (numeric output of the last linear layer of a multi-class classification neural network) into probabilities by take the exponents of each output and then normalize each number by the sum of those exponents so the entire output vector adds up to one — all probabilities should add up to one. Cross entropy loss is usually the loss function for such a multi-class classification problem. Softmax is frequently appended to the last layer of an image classification network such as those in CNN (VGG16 for example) used in ImageNet competitions.

3.3 Hardware and Software Requirements

- **Hardware :**

Processor - Intel i3 or above, AMD equivalent

Ram - 2GB or more

GPU - Intel HD or any integrated GPU

- **Software :**

OS – Windows 7/8/10, Linux, MacOS (with active internet connection [not needed if using local libraries instead of CDNs)

Browser –Well updated browser supporting JavaScript (Chrome,Firefox,etc)

Editor –Any (if required)

Intended Audience: Project team members & Users.

CHAPTER 4

PROJECT DESIGN

4.1 Flowchart of Genetic Algorithm:

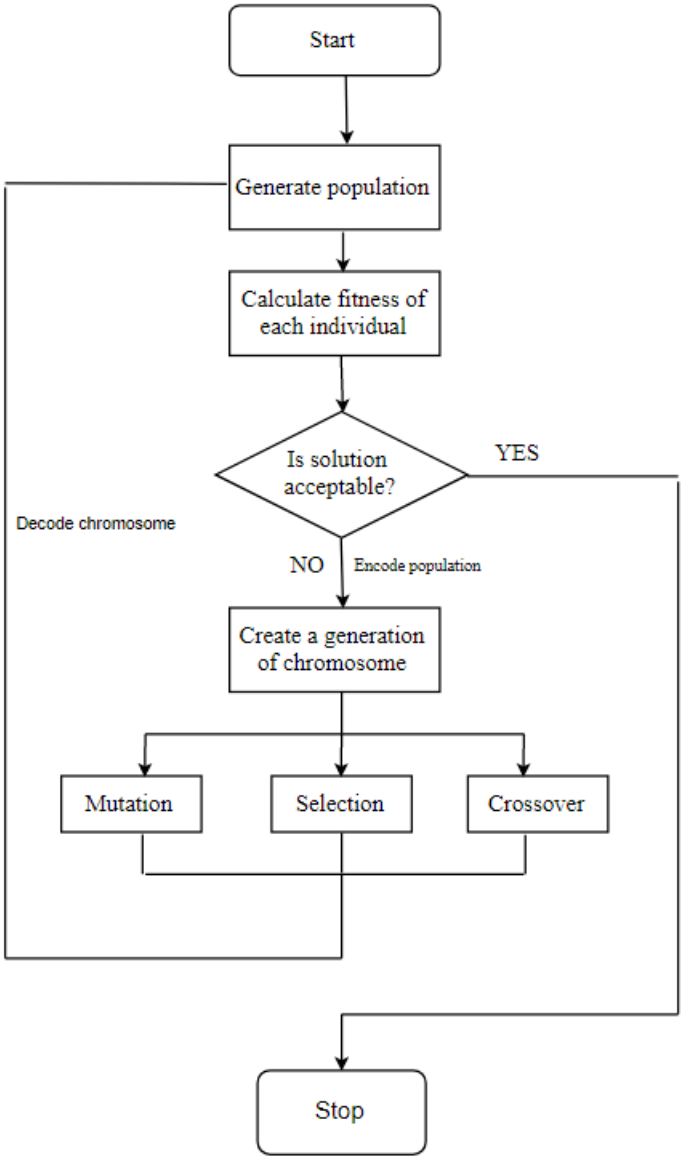

Fig 4.1 Flowchart of Genetic Algorithm

The flow-chart above shows the general flow of a genetic algorithm, but our case we use the weights of the neural network of every agent in the population for crossover process.

4.2 Architecture:

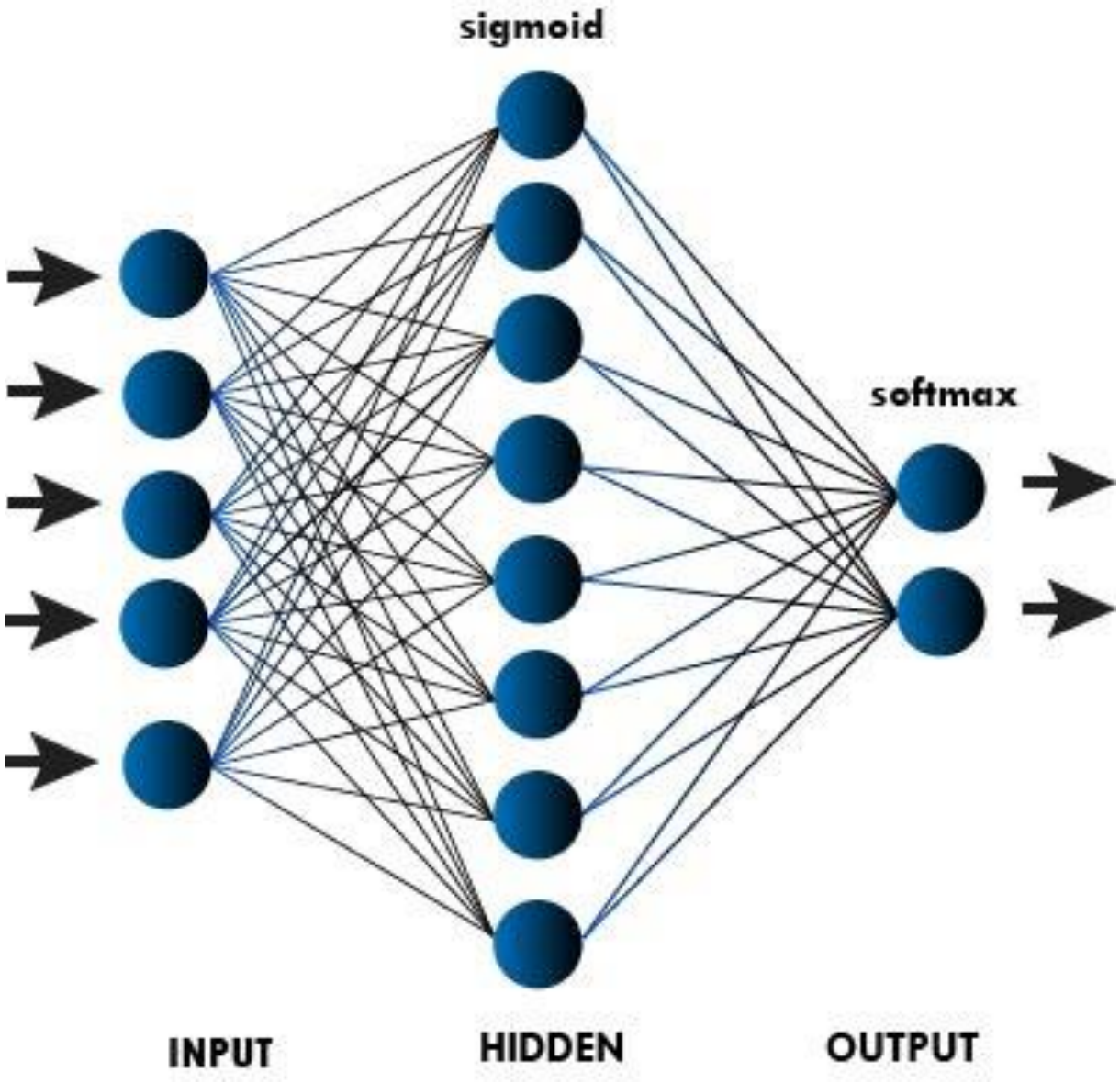

Fig 4.2 Architecture of Neural Network

Inputs to the above neural network are:

Agent y position.

Top obstacle

Bottom obstacle

Closest obstacle

Agent y velocity

4.3 Web Technologies used:

TensorFlow:

TensorFlow is a free and open-source software library for dataflow and differentiable programming across a range of tasks. It is a symbolic math library, and is also used for machine learning applications such as neural networks. The deep learning artificial intelligence research team at Google, Google Brain, in the year 2015 developed TensorFlow for Google's internal use. This Open-Source Software library is used by the research team to perform several important tasks.
TensorFlow is at present the most popular software library. All the computations associated with TensorFlow involves the use of tensors.

A tensor is a vector/matrix of n-dimensions representing types of data. Values in a tensor hold identical data types with a known shape. This shape is the dimensionality of the matrix. A vector is a one-dimensional tensor; matrix a two-dimensional tensor. Obviously, a scalar is a zero dimensional tensor.

In the graph, computations are made possible through interconnections of tensors. The mathematical operations are carried by the node of the tensor whereas the input-output relationships between nodes are explained by a tensor's edge. Thus TensorFlow takes an input in the form of an n-dimensional array/matrix (known as tensors) which flows through a system of several operations and comes out as output. Hence the name TensorFlow. A graph can be constructed to perform necessary operations at the output.

We used tensorflow for creating models of our neural network architecture with dense layers, feeding it with inputs, converting them to tensors for processing, using and specifying an activation function to process data in the dense layers, use data Sync to get back the data from tensors and make copies of the weights to create a copy of the model with mutation.

JavaScript:

JavaScript (JS) is a lightweight, interpreted, or just-in-time compiled programming language with first-class functions. While it is most well-known as the scripting language for Web pages, many non-browser environments also use it, such as Node.js, Apache CouchDB and Adobe Acrobat. JavaScript is a lightweight, interpreted programming language. It is designed for creating network-centric applications. It is complimentary to and integrated with Java. JavaScript is very easy to implement because it is integrated with HTML. It is open and cross-platform. Javascript is the most popular programming language in the world and that makes it a programmer's great choice. Once you learnt Javascript, it helps you developing great front-end as well as back-end software using different Javascript based frameworks like jQuery, Node.JS etc.

Javascript is everywhere, it comes installed on every modern web browser and so to learn Javascript you really do not need any special environment setup. For example Chrome, Mozilla Firefox, Safari and every browser you know as of today, supports Javascript.

Javascript helps you create really beautiful and crazy fast websites. You can develop your website with a console like look and feel and give your users the best Graphical User Experience.

JavaScript usage has now extended to mobile app development, desktop app development, and game development. This opens many opportunities for you as Javascript Programmer.

Due to high demand, there is tons of job growth and high pay for those who know JavaScript. You can navigate over to different job sites to see what having JavaScript skills looks like in the job market.

Great thing about Javascript is that you will find tons of frameworks and Libraries already developed which can be used directly in your software development to reduce your time to market.

p5.js:

p5.js is a JavaScript library that starts with the original goal of Processing—to make coding accessible for artists, designers, educators, and beginners—and reinterprets this for today's web. Using the original metaphor of a software sketchbook, p5.js has a full set of drawing functionality. The main focus of processing is to make it easy as possible for beginners to learn how to program interactive, graphical applications, to make a programming language more user-friendly by visualizing it.

The advantage of using the JavaScript programming language is its broad availability and ubiquitous support: every web browser has a JavaScript interpreter built-in, which means that p5.js programs can be run on any web browser.

Also, Processing is that language which emphasizes on feasibility for programmers to create software prototypes very quickly, to try out a new idea or see if something works. For this reason, Processing (and p5.js) programs are generally referred to as “sketches.”

Difference between P5.js and JavaScript?

JavaScript is a core language that provides all the features to build any functionalities into browsers. It can use Loop, Function, Conditional, DOM Manipulation, events, canvas, etc. Hence, by using it to develop and design any framework.

p5.js is a library of JavaScript. P5.js is running on Pure JavaScript provides some functions that make JavaScript user life easy to draw in the web.

setup(): It is the statements in the setup() function. It executes once when the program begins. Create Canvas must be the first statement.

draw(): The statements in the draw() are executed until the program is stopped. Each statement is executed in sequence and after the last line is read, the first line is executed again.

Using the original metaphor of a software sketchbook, p5.js has a full set of drawing functionality. However, you're not limited to your drawing canvas, you can think of your whole browser page as your sketch! For this, p5.js has add-on libraries that make it easy to interact with other HTML5 objects, including text, input, video, webcam, and sound.

HTML:

Hypertext Markup Language is the standard markup language for documents designed to be displayed in a web browser. It can be assisted by technologies such as Cascading Style Sheets and scripting languages such as JavaScript.HTML elements are the building blocks of HTML pages. With HTML constructs, images and other objects such as interactive forms may be embedded into the rendered page.

HTML provides a means to create structured documents by denoting structural semantics for text such as headings, paragraphs, lists, links, quotes and other items. HTML elements are delineated by tags, written using angle brackets. Tags such as <img /> and <input /> directly introduce content into the page. Other tags such as <p> surround and provide information about document text and may include other tags as sub-elements. Browsers do not display the HTML tags, but use them to interpret the content of the page.

HTML stands for Hypertext Markup Language. It allows the user to create and structure sections, paragraphs, headings, links, and block quotes for web pages and applications.HTML is not a programming language, meaning it doesn't have the ability to create dynamic functionality. Instead, it makes it possible to organize and format documents, similarly to Microsoft Word.

HTML Evolution. What Differs Between HTML and HTML5?

Since the first days, HTML has gone through an incredible evolution. W3C constantly publish new versions and updates, while historical milestones get dedicated names as well.

HTML4 (these days commonly referred to as "HTML") was published in 1999, while the latest major version came out in 2014. Named HTML5, the update has introduced many new features to the language.

One of the most anticipated features of HTML5 is native support for audio and video embedding. Instead of using Flash player, we can simply embed videos and audio files to our web pages using the new <audio></audio> and <video></video> tags. It also includes in-built support for scalable vector graphics (SVG) and MathML for mathematical and scientific formulas.

HTML5 introduced a few semantic improvements as well. The new semantic tags inform browsers about the meaning of content, which benefits both readers and search engines.

Pros:

A widely used language with a lot of resources and a huge community behind.

Runs natively in every web browser.

Comes with a flat learning curve.

Cons:

Mostly used for static web pages. For dynamic functionality, you may need to use JavaScript or a backend language such as PHP.

It does not allow the user to implement logic. As a result, all web pages need to be created separately, even if they use the same elements, e.g. headers and footers.

Some browsers adopt new features slowly.

CSS:

Cascading Style Sheets is a style sheet language used for describing the presentation of a document written in a markup language like HTML. CSS is a cornerstone technology of the World Wide Web, alongside HTML and JavaScript.CSS is designed to enable the separation of presentation and content, including layout, colors, and fonts.

This separation can improve content accessibility, provide more flexibility and control in the specification of presentation characteristics, enable multiple web pages to share formatting by specifying the relevant CSS in a separate .css file which reduces complexity and repetition in the structural content as well as enabling the .css file to be cached to improve the page load speed between the pages that share the file and its formatting.

Separation of formatting and content also makes it feasible to present the same markup page in different styles for different rendering methods, such as on-screen, in print, by voice (via speech-based browser or screen reader), and on Braille-based tactile devices. CSS also has rules for alternate formatting if the content is accessed on a mobile device.

The CSS specifications are maintained by the World Wide Web Consortium (W3C). Internet media type (MIME type) text/css is registered for use with CSS by RFC 2318 (March 1998). The W3C operates a free CSS validation service for CSS documents.

In addition to HTML, other markup languages support the use of CSS including XHTML, plain XML, SVG, and XUL.CSS3 version introduced new features that allowed developers to style HTML elements with less CSS code. CSS3 is most notorious for modules that speed up the specification process.

At first, browsers did not support CSS3 features, and it took a while for them to become fully compatible.CSS3 does not deprecate older CSS code because it is only an addition to the features offered by CSS1. This list provides the main arguments in the CSS3 vs CSS debate:

CSS3 allows developers to style HTML elements easier. They are less dependent on image files and can complete CSS styling with fewer lines of code. The aim of CSS1 was for appearance formatting, and it did not allow responsive designs.CSS3 supports responsive designs and manages media queries (used for determining users' devices and screen sizes).

Since CSS3 new features let you produce fewer lines of code, you can boost the website speed.

Since CSS3, CSS animations move without JavaScript or Flash code.

CHAPTER 5

IMPLEMENTATION DETAILS

5.1 Modules & Description:

sketch.js (main p5 file):

This file has a draw function that is looped over continuously at 60fps,using the metaphor of a sketch, p5.js has a full set of drawing functionality that helps in building our game from scratch.

It is mainly responsible for the connection of all other classes of the neural network, game files and calling important functions mostly related to the smooth functioning of p5 library which is required as per the the documentation. It also defines all the important functionality of the game metrics and parameters like the number of bots to be trained in a population, how often to generate the pipes and other functions that in turn change the way the neural networks function since the same inputs are feeded into them changing the entire outcome of the process.

pipe.js (file responsible for generation of obstacles):

This file is responsible for the generation of obstacles to display over the course of the game. The obstacles have randomized opening positions and heights, also the spacing is further reduced after every generation.

The calculations for the length of the top and bottom obstacles are done very carefully so as to make sure they never coincide with each other ever while the still making sure the next pipes are not close enough and making sure the pipes have the same amount of width. It is also imperative that the pipes and the lengths of the pipes are somewhat random with equal amount of spacing between the pipes, which if not maintained will cause a major bug in the game since the bot may not be able to theoretically pass the obstacles because the spacing was less than the size of the bot itself.

bird.js (Responsible for creating our learning agents):

This file is responsible for creating and displaying the agent onscreen, calling crossover and mutation related functions, determining whether any agents or obstacles went off-screen or not, and applying the physics of the game to the agent. These bots are responsible for playing the game and mastering it even in extreme increasing difficulties. They are of the same dimensions [size] and represent the neural network which are fed the same inputs. The greater the population sizes the higher chance of finding the one bot that masters the game or best performer for that matter.

Just as when pipes go off-screen they are sliced out of the array to reduce memory leak, the same needs to be done with the birds [bots] after the generation has ended and the best bot is used to create a new generation, after which we no longer require the older generation of bots since the new generation of bots created by the best bot of last generation was already transferred to the new population.

nn.js (Responsible for creating structure of neural network):

This file has mostly tensorflow functionality, responsible for creating the shape and model of our neural network architecture and implementing crossover function.

This files contains the main model of the neural network that will adapt to the increasing difficulty after every generation and still be able to master the game and play it in a situation where it would be impossible for a human expert to play in. Since this file contains the model and the architecture it is responsible for specifying the number of layers and the number of nodes in each layer. Other than that it also mentions the activation function that is to be used in each layer where the data is to be processed which in our case is sigmoid in the hidden layer and softmax in the output layer.

With creation of the model it is also responsible for the crossover functionality of the genetic algorithm since it has the model and make a copy of it with getting and setting required weights.

ga.js (Genetic algorithm implementation file):

As stated, it is responsible for all the genetic algorithm processes, calculating fitness of every individual agent and increasing difficulty of the game after each generation.

It is responsible for finding a fitness parameter of each of the bots in the population and comparing the fitness values of every bot, and in turn selecting the best performing bots for crossover of next generation and saving their weights. Since selection process is done the saved birds are then used to create a new population of birds [bots] that will most likely be better at the game than the previous generation as a whole. After discarding the old generation and creating copies which is the crossover process, it is also responsible to add mutation to the new generation of bots which is done by changing the weights of the bots by a small number which was found by Gaussian random function in our case and completing the mutation process. The last thing it does is feeding this new set of generation to the game and starting the process all over again and looping till the optimal bot is found.

index.html (main webpage):

Responsible for rendering the sketch on the canvas and linking all libraries, css and previous game files.

main.css :

Styling score counter and background.

5.2 Screenshots:

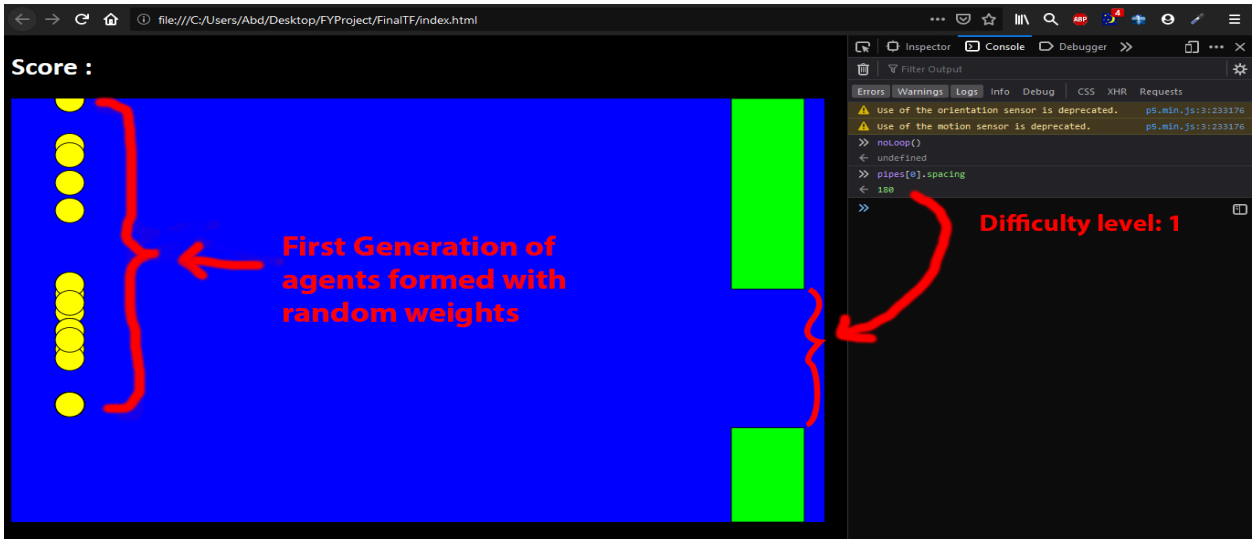

Fig 5.1First Generation of Agents

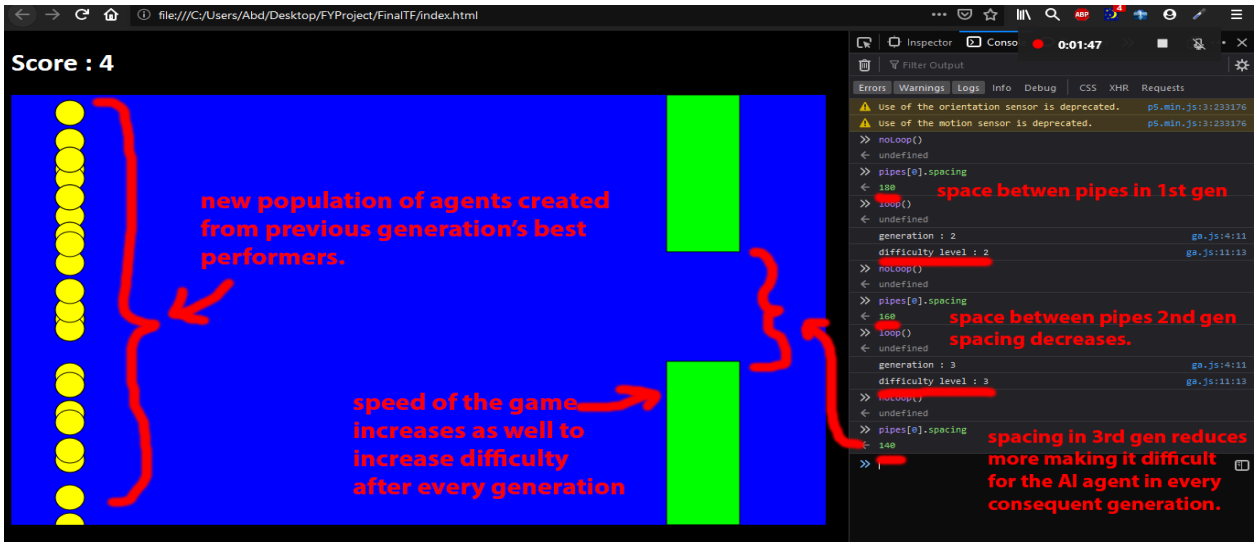

Fig 5.2 New Population from Fittest Parent

Fig 5.3Best Performing Agent

CHAPTER 6

TESTING

6.1 Development of test cases (Functional):

Test Case ID	Objective	Actions	Excepted Result	Test Status
TC1	Move up	1-User wants to move upwards	The agent moves upwards	Pass
TC2	Float down	1-User want to descend towards the floor	The agent descends downwards. tip	Pass
TC3	Collision detection	1-User wants feedback if agent touches the obstacles	The obstacles turn red if agent collides with them	Pass
TC4	Agent termination	1-User wants the agent to be killed if Collision is detected or the floor is in contact	Agent dies on collision and touching the floor	Pass
TC5	Speed increment	1-User wants the speed of the game to increase after every generation	Speed of the game increased after each generation	Pass
TC6	Game difficulty	1-User wants the spacing of pipes to decrease after every generation	Pipe spacing reduced after each generation	Pass
TC7	Renewed Generation	1-User wants to create new generation from previous generation agents	New generation of agents created	Pass
TC8	Counters check	1-User wants counters of max score, generation number, difficulty levels to be displayed accurately	Counters display accurate score, levels and generation number	Pass

Table 6.1 Development of Test Cases

CHAPTER 7

SUMMARY

GA has been successful in complex engineering applications that involved multiple objective, not well-defined optimization function. Thus, in our project by using genetic algorithm with an underlying multi layered neural network we will be able to train an agent to learn the game. Genetic Algorithms mimic the survival of the fittest, in a way that, generations are maintained, and from one generation, only those characters are able to pass to the next generation which have survived/performed well in that generation. Each generation is characterized by a set of strings, which are analogous to the chromosomes in the DNA. Crossover, Mutations and other biological effects occur and in turn affect the offspring in the next generation. The same holds true for a genetic algorithm too. Each individual represents a point in a search space and a possible solution.

Our neural network architecture uses a feed forward neural network which can also be called a multi layered perceptron with two layers containing one hidden layer and one output layer. The hidden layer uses sigmoid as the activation function and the output layer uses softmax as the activation function to get probabilistic results for decision making. The project demonstrates an evolutionary approach to solve a non-optimized problem using AI and neural networks.

CHAPTER 8

RESULT AND ANALYSIS

We have created a working implementation of the game and AI bot that demonstrates neuro-evolution of an AI bot that can outperform any human expert of the game even in extreme difficulty and speed by using and combination of genetic algorithm and feed forward neural network. It can clearly be seen that there is not much concrete mathematical foundation to genetic algorithm. The algorithm itself is heavily derived from biology. Further improvements can be made using convolutional neural networks, using deep learning or reinforcement learning techniques or improved by changing the neural network architecture and making further refined calculations to create a fitness function that would improve performance. This approach can also be used to implement AI's for more games that improve the overall experience.

We were quite fascinated by the fact that such simple "random" updates can develop such undefeatable game players. We felt great while working on this project. Also, the idea of evolution behind the Genetic Algorithms is quite interesting.

CHAPTER 9

REFERENCES

Books,

[1] T. Bäck, D.B. Fogel, Z. Michalewicz (Eds.), *Handbook of Evolutionary Computation*, IOP Publishing, 1997. Pages about Evolutionary Computation

[2] Sanjay Madhav, *Game Programming Algorithm & Techniques,* Stanford.

Referred Papers,

[1] Kevin Chen,*Deep Reinforcement Learning for Flappy Bird,* Stanford.

[2] Naveen Appiah, SagarVare,*Playing FlappyBird with Deep Reinforcement Learning*, Stanford.

[3] HalduraiLingaraj, *A Study on Genetic Algorithm and its Applications,* Researchgate.

Articles,

[1] S. S. Chaudhry &W. Luo *Application of genetic algorithms*

[2] Pulkit Sharma *An Introductory Guide to Deep Learning and Neural Networks*

[3] Bian Wu, *A Guideline for Game Development-Based Learning: A Literature Review*

CHAPTER 10

PUBLICATION BY THE CANDIDATES

Flappy Bot using AI

Abdeali Saria
Department of Computer Engineering
Shah & Anchor Kutchhi Engineering College
Mumbai, India
abdeali.saria@sakec.ac.in

Gaurav Pandav
Department of Computer Engineering
Shah & Anchor Kutchhi Engineering College
Mumbai, India
gaurav.pandav@sakec.ac.in

Mihir Desai
Department of Computer Engineering
Shah & Anchor Kutchhi Engineering College
Mumbai, India
mihir.desai@sakec.ac.in

Prof. Tina Maru
Department of Computer Engineering
Shah & Anchor Kutchhi Engineering College
Mumbai, India
tina.maru@sakec.ac.in

***Abstract* -In today's world, an intelligent and optimal problem-solving approach is required in every field. Machines are becoming more and more efficient; many applications have been made recently to solve complex problems. In artificial intelligence, Genetic algorithm is a heuristic search technique to find the most optimized solution for a given problem based on crossover, mutation, selection and some other techniques inspired by Darwin's theory of evolution. This report demonstrates how it approaches to an optimization problem in general. An implementation of genetic algorithm on building an AI for an arcade game. Our game has an unclear finishing point as it goes on forever and the difficulty varies overtime making genetic algorithm a good fit to train the agent.**

Keywords—Genetic Algorithm, Collision Detection, Artificial Intelligence, Feed Forward Neural Network.

I. INTRODUCTION

The goal for the project is to create an AI (bot) efficient enough to master the proposed game below with genetic algorithm as the base for implementation with feed forward neural networks. Our game consists of a spherical agent trying to maneuver through the space between the obstacles in its course. These obstacles have different heights and spacing overtime, additionally the speed and number of obstacles increase progressively making it harder for the AI. The rule of the game is that our agent is always falling towards the floor and we flap (move up) to move between the obstacles. If the agent falls and touches the floor, or if the agent collides with one of the obstacles then the game ends. Every obstacle avoided increases the score which provides a metric to judge the performance of the AI.Genetic algorithm follows the same fundamental processes that mimic the evolutionary process of nature and justify Darwin's theory of evolution also demonstrating adaptability and survival of the fittest aspect of nature. The universe is finite, its resources finite, competition for these resources leads to the fittest ones dominating others.

Similarly, the fitness function of our project will determine the fitness of our agent which in turn will establish its performance factor. Genetic algorithm employs previous generation's information to further enhance the performance of the population of the descendant generations. We looked upon previous projects based on genetic algorithm and were amused by them and hence decided to implement an AI for the game using neural networks and genetic algorithm.

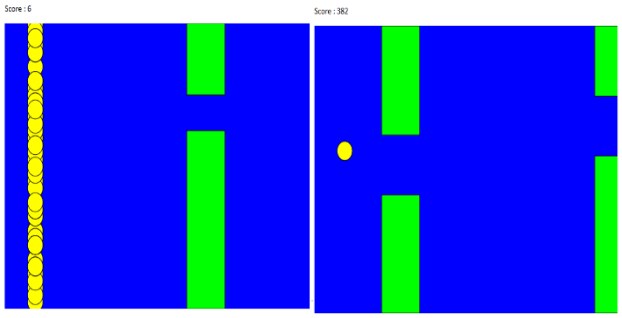

Fig 1.1 Population of agents Fig 1.2 Best performing agent

II. LITERATURE SURVEY

The related work in this domain is primarily by Google Deep mind are able to successfully train agents to play the Atari 2600 games using deep reinforcement learning, surpassing human expert-level on multiple games. These works inspired this project, which is heavily modeled after their procedure. They use a deep Q-network (DQN) to assess the Q-function for Q-learning and also use trained experience replay to de-correlate learning. Their approach is naturally state-of-the-art and was the main catalyst for deep reinforcement learning, after which many papers tried to make improvements. The main quality is that they were able to train an agent despite extremely high dimensional input (pixels) and no description about essential game parameters.

In fact, they are able to surpass a human expert on three out of seven Atari 2600 games. We have read papers of other similar projects based on the idea of building an AI that can master the game and found them extremely intriguing and helpful and have tried a simpler approach since we are

beginners in the field of artificial intelligence and found that genetic algorithms and neural network combination is best for these sort of optimization problems.

Outline of Genetic Algorithm process:

Genetic algorithm starts off with a population of nodes (agents in our project) all containing random weights or information from the input and therefore, performing random decisions to lead towards the goal destination. Since the algorithm is evolutionary, the information about the inputs, weights and decisions of previous generations are preserved, so that the future generations perform better towards reaching the goal state, justifying evolution. The historical information of previous generations has to follow an important constraint before it is stored for future use, which is, from the entire population of agents, information of only those agents that have performed better than the others will be saved to pass onto the next generation. However, the information to be passed onto the next generation is not passed in as raw data to the future population of agents, as it will most likely in all scenarios perform exactly the same way the previous generations performed from which the data was extracted. Hence, crossover and mutation is performed to make sure the behavior of the new set of agents have some variety, and they make different or rather, better decisions than the previous population of agents. The steps in genetic algorithm are briefed below:

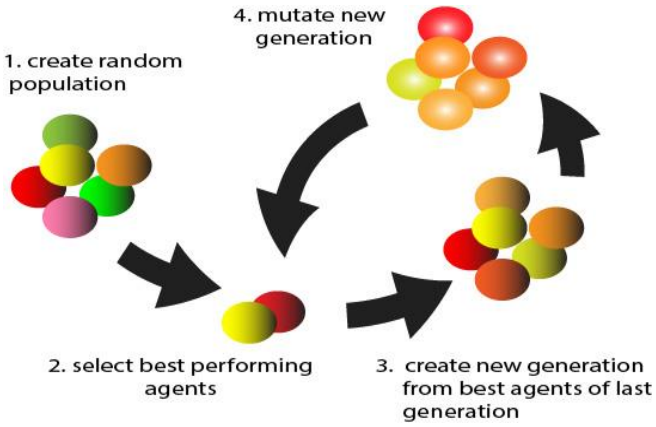

Fig 2.1 Process of genetic algorithm

Initialize Search Space:
Creating a population of nodes inside the search space where every node of the population makes a decision to provide the solution to the problem statement is the first step. In our project, the search space would be the population of agents created that have random weights and make random decisions at the start and further refine them towards the goal state.

Selection:
Selection is the process of picking the best performing or the fittest agent, so to speak from the search space by comparing the fitness values of the entire search space and save it.

Crossover:
Crossover is the process of creating a new individual by using best performing individual of previous generation and similarly generating an entire population of agents. Since the new search space is derived from the previous best agents it will generally be better than the previous generations.

Mutation:
Mutation is one of the imperative steps of genetic algorithm as it ensures variety in the decision making of each individual to be distinct from one another. It is performed after crossover and after every new generation of the search space is created for the next cycle. Without mutation we won't have variety and it is important that each agent is mutated individually rather than mutating the entire population at once. It ensures that the entire new generation doesn't make the same decisions and has diversity.

III. PROPOSED SYSTEM

Genetic algorithm has been successful in complex engineering applications that involved multiple objective, not well-defined optimization. Thus, in our project by using genetic algorithm with an underlying multi layered neural network we were able to train an agent to learn and master the game, quickly and more efficiently as compare to humans. The project also demonstrates how AI's for other applications and games can also be created with the same idea of this project.

i) Search Space - Population of the agents that will attempt on learning the game, while starting with random weights.
ii) Selection - After calculating the fitness parameters of the population we select the fittest agents for further steps.
iii) Crossover - Using the selected agents that performed the best out of the population, we create a new generation of agents for improving the agent's fitness score.
iv) Mutation – The new generation of agents are randomly mutated using a Gaussian random function altering the weights of the neural network. These steps are repeated over generations till solution is found.

Architecture:

Our neural network architecture uses a feed forward neural network which can also be called a multi layered perceptron with two layers containing one hidden layer and one output layer. There are five input nodes in the neural network that are fed to the hidden layer which has eight hidden nodes which use sigmoidal activation function to process the inputs from the input layer and pass it to the output layer.

The output layer has two nodes as we can narrow the outputs down to a simple classification problem of whether to move up or to not move up. It uses softmax function as the activation function which returns a probabilistic value used for decision making and performs the action based on the output values of these two nodes.

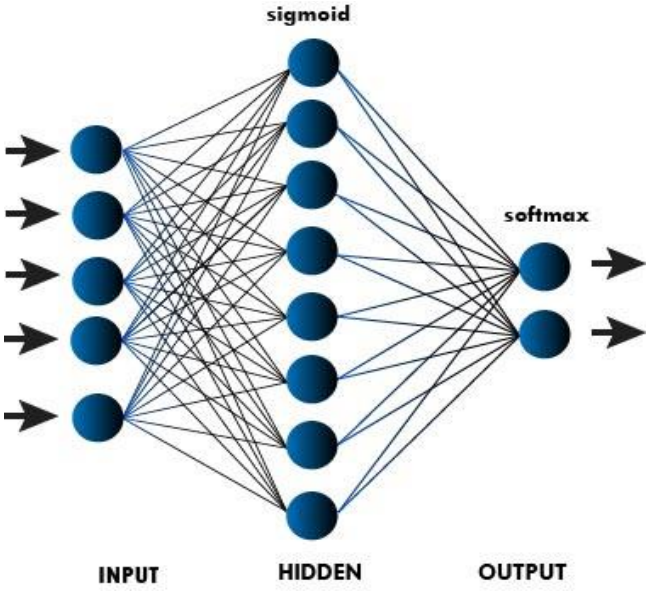

Fig 3.1 Neural network Architecture

IV. RESULT AND ANALYSIS

It can clearly be seen that there is not much concrete mathematical foundation to genetic algorithm. The algorithm itself is heavily derived from biology. The graph below visualizes the results that were obtained after a test run. It is evident that more often than not after each generation the performance of our agents improve drastically until it completely masters the game which was our final goal.

Further improvements can be made using convolutional neural networks, using deep learning or reinforcement learning techniques or improved by changing the neural network architecture and making further refined calculations to create a fitness function that would improve performance. This approach can also be used to implement AIs for more games that improve the overall experience.

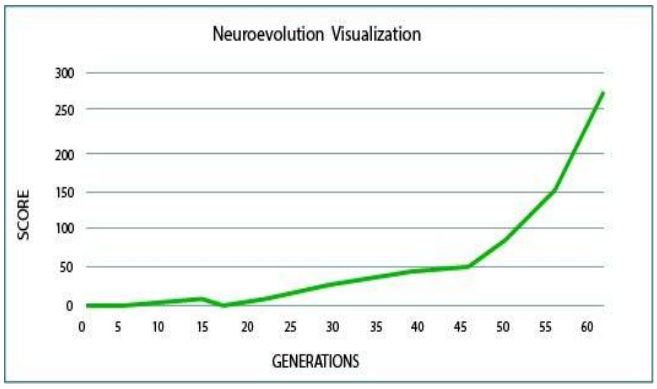

Fig 4.1 Results of test run

V. CONCLUSION

We have created a working game that detects collision when the agent collides with the obstacles and masters it with every passing generation. We were excited to see that such arbitrary mutations can create AIs that can outscore any human expert. Our AI agents use a feed forward neural network architecture which can be changed for better performance. We felt great while working on this project, and learnt a lot about evolutionary approaches or algorithms. The project demonstrates an evolutionary approach to solve a non-optimized problem using AI and neural networks.

VI. REFERENCES

1. Similar project: http://cs229.stanford.edu/proj2015/362_report.pdf2
2. http://theory.stanford.edu/~amitp/GameProgramming/AITechniques.html5
3. Demonstrations of Genetic Algorithm process: https://towardsdatascience.com/genetic-algorithm-explained-step-by-step-65358abe2bf?gi=dd893c99e3e
4. An introductory guide to Neural Networks: https://www.analyticsvidhya.com/blog/2018/10/introduction-neural-networks-deep learning/
5. A study on Genetic Algorithm and its Applications: https://www.researchgate.net/publication/309770246_A_Study_on_Genetic_Algorithm_and_its Applications
6. Genetic Algorithm theory: http://theory.stanford.edu/~amitp/GameProgramming/AITechniques.html5
7. Wiki pages of associated topics.

CHAPTER 11

PLAGIARISM REPORT

URKUND

Urkund Analysis Result

Analysed Document:	IEEE Paper refactored.docx (D67945965)
Submitted:	4/11/2020 9:02:00 AM
Submitted By:	tina.maru@sakec.ac.in
Significance:	0 %

Sources included in the report:

Instances where selected sources appear:

0