Namal College, Mianwali

AIFG

AI Based Tanks Game

Authors:
Abdul Malik, Shakir Ullah, Badar Shahzad and Abdul Rehman Khan Niazi

Instructor:
Dr. Junaid Akhter

May 23, 2018

Contents

1 Introduction

In this undertaking we are required to develop an AI agent for a game titled Tanks, which will be capable enough to become adaptive and take optimal moves to try to win the game. In this game there are four players, three of them are taking random moves while one of them is the superior one, which takes an intelligent move. For this essence we have implemented different strategies which empowers the AI agent to behave more like human. This document is based on all the implemented techniques to make AI agent look smarter. Testing and pruning has also been discussed in this archive.

2 Background

2.1 Overview

To develop an AI base game is very challenging and somehow it seems so much fun to work on. But the games that are based on AI are develop by different techniques, and strategies. AI is mainly used to generate response for non-player character in games to act up as similar to human. Tanks is a four players game that can be played in single mode such as AI can play against AI. In our demonstration we will present our program with implementation of Evolutionary and Neural Network algorithms. Furthermore, we also explain the evolution of ideas and algorithms which we tried and are not included in the final implementation.

2.2 Game Rules

- Occupy as many terrain Grid Squares as possible.
- Destroy the opponents Tanks.
- Dodge your own tank from other tanks.
- Player will be awarded with points, if the tank visits a new cell or destroy an opponent tank.
- Points will be awarded when a player visits an unvisited cell even if it is visited by the opponent players.
- Points also depends on terrain.

- Each tank has its own vision range and can see strait forward to its own pave. Additionally, it can also see diagonally but in tanks hull direction.
- Tank can shoot the opponent only if the opponent tanks get reveal in the direction of hull.
- When a Tank destroy the other tank it will become a hurdle at that place.
- The player who got highest scores will be the winner even if it is dead while the game ends.
- The game only gets end when a fixed number of turns is reached.
- When a tank fires, its location should be broad casted globally.

2.3 Evolutionary Algorithm

At first evolutionary algorithms were introduced in the late 1960s. They were developed to solve the problems that were difficult for traditional analytical methods. Instead of games, their exist multiple applications for these types of algorithms such as machine learning, hardware design and robotics [1]. The evolution process in evolutionary algorithm can be seen in the figure shown below. It contains different steps that are repeatedly called until a condition or set is satisfied. To end this process the condition could be a number of evolved generations or if the time limit is reached.

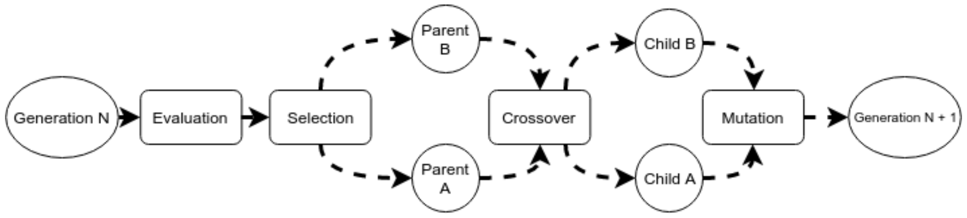


Figure 1: Steps of Evolutionary Algorithms [2]

2.4 Artificial Neural Network (ANN)

ANN is a popular machine learning technique, it is made on the model of a human brain. Neural network is similar to the human brain in the following ways.

- A neural network obtains knowledge from learning.
- A neural networks knowledge is kept inside in the inter-neuron connection strengths known as weights.

There are three kinds of layers in artificial neural network; input layer, hidden layers and output layer. The number of neurons and hidden layers can be different in distinct layers and networks, respectively. The basic structure of artificial neural network is shown in Figure 2.

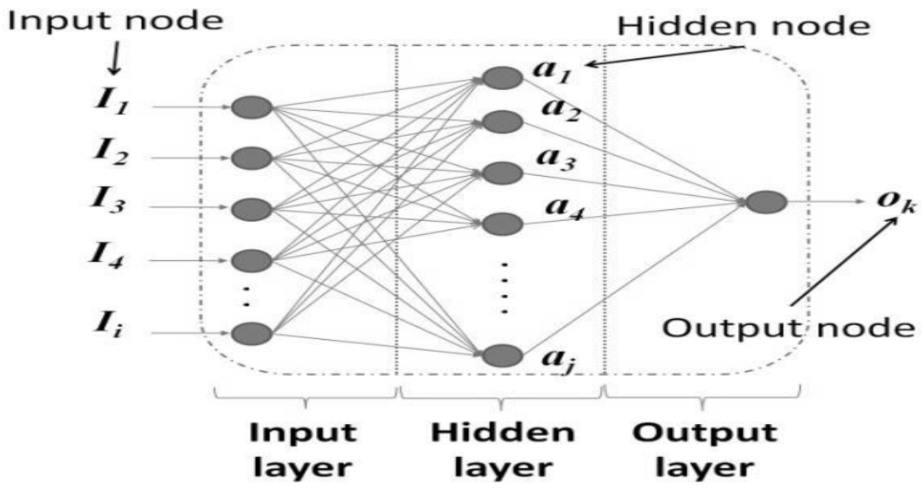


Figure 2: Basic Architecture of ANN

In artificial neural network, the network is first train by providing the input data and the desired output (in case of supervised learning). The ANN can be train using different training functions. These functions are, Variable Learning Rate Back propagation (GDX), Scaled Conjugate Gradient (SCG), Levenburg-Marquardt (LM), Steepest Descent, Adaptive moment estimation (Adam) etc. After training the network properly, the next phase is testing of the network. In testing, the network is only provided with the input data and the output is generated by the system using its previous knowledge or learning.

3 Design Ideas

3.1 Min-Max Algorithm

Min-Max is a recursive algorithm which is usually used to take a next move in a game. It generates all possible moves for an AI agent as well as for opponent player hence, making a tree structure (figure 3) of all possible moves in a game. It starts tracking back to root node returning optimal values from each branch of a tree, once it reaches to the leaf nodes. Generally Min-Max does the following things:

- Returns a value i-e $+1$, 0 or -1, if a leaf node is found. Leaf nodes calculate their values based on agents condition i-e win, lose, draw or continue.
- Visits all boxes on the board.
- The Min-Max function is called on each box.
- On each node it evaluates the returning values from each branch of that node and returns the best value.

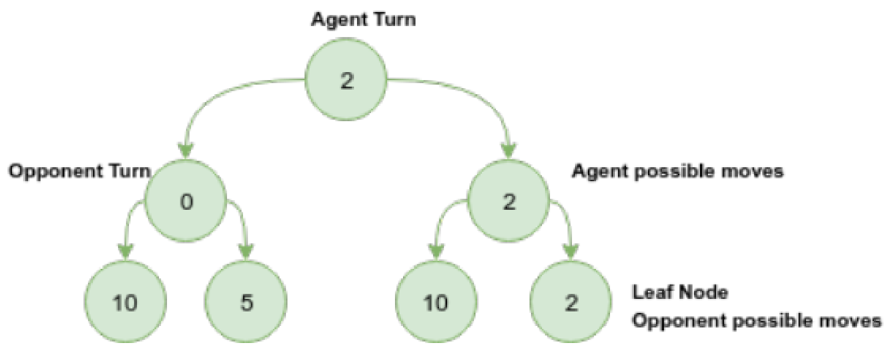


Figure 3: Min-Max tree

3.1.1 Limitation in Min-Max

We have already implemented Min-Max algorithm for Tic Tac Toe and Snails grid world. As we know that in Tic Tac Toe we have $3x3$ board therefore, it got executed excellentally for Tic Tac Toe but when we come across with games having higher dimensions of boards i-e Tanks ($15x15$) then such type of algorithm may not work because of its time complexity. Therefore, choosing Min-Max algorithm for such type of games would be no more favorable.

3.2 A* Algorithm

A* is the one of the most widely used and popular methods for finding the shortest path between two points. A* algorithm is an extension of Dijkstras algorithm with some features of breadth-first-search. It introduces a heuristic into a regular searching algorithm, basically planning ahead at each period so that the best decision is made.

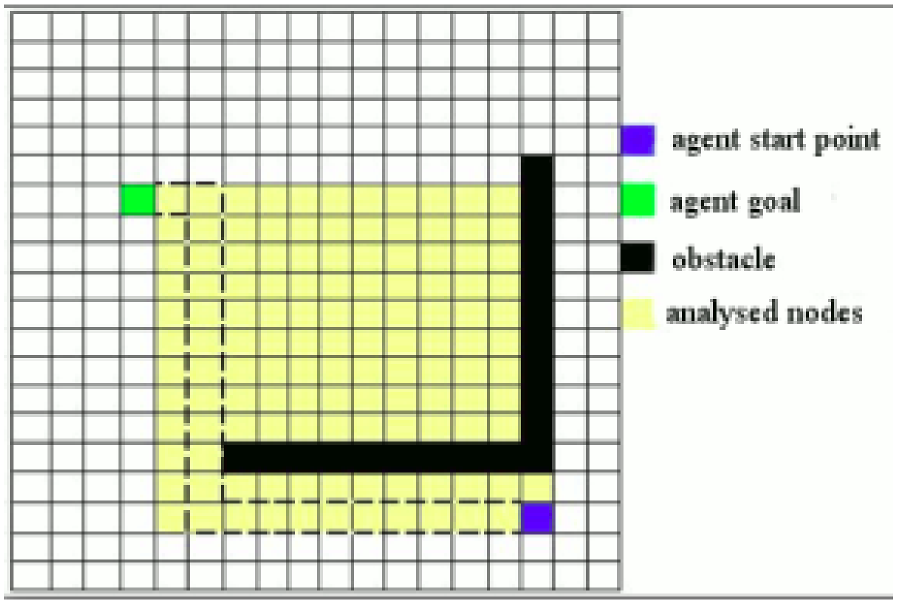


Figure 4: A* Algorithm

The important features of the A* algorithm are the construction of a "closed list" to record areas previously evaluated, a "fringe list" to record areas that are being evaluated, and the calculation of distances toured from the start point with estimated distances to the final point (goal). The fringe list, also called the open list, is a list of all places directly adjacent to the agent coordinate. The closed list is a record of all places which have been discovered and evaluated. Figure 4 gives an idea about working of A*.

3.2.1 Limitation in A*

In Snails grid world Min-Max was taking too much time therefore, we used A* towards its AI implementation. A* algorithm was quite good as compared to the Min-Max algorithm in term of time complexity and efficiency. The solution that we designed using A* algorithm was not generalized as we were not able to apply it on the other games as well. Therefore, the decision was not to use this algorithm for the AI implementation of Tanks game.

3.2.2 Artificial Neural Network and Evolutionary Algorithm

Because of the limitations in Min-Max and A*, we wanted to devise a generalized algorithm that can be used for any board game. Then we come up with an idea of Artificial Neural Network (ANN). But for ANN we needed large amount of training data. To train an ANN, we have used Evolutionary Algorithm as a part of AI. A long discussion has been done on these algorithms in the upcoming passages.

4 Game Implementation

4.1 Backend Representation

The backend of a game describes how the game works, updates and changes. As we know that tanks is a four player game, there should be four different representation for each player at the backend. Each tank will have a local board in which we will be maintaining the visited and unvisited boxes because each tank gets 10 score on exploring an unvisited box. Tank can also move its hull left and right. So, we need to maintain a number at the backend for hull direction as well. When a tank sees the opponent tank it can fire also because on killing a tank, the killer tank gets increment of 50 scores. There are hurdles also in the game therefore, we need to have some numbers at the backend in the board. For each tank we have to maintain two boards, one is the local board and the other one is global board. In the global board we need to maintain the tanks positions, hurdle and a number for dead tanks. In the local board, we have to maintain the tanks positions and visited and unvisited boxes in their respective local boards. Overall, we need to maintain five options for each tank. And these are,

1. Turn (position of the tank)
2. Local board of the tank

3. Direction of hull
4. Score of the tank
5. Dead or alive

For players position, the representation at the backend are 11, 22, 33 and 44 for first, second, third and fourth player respectively. When a player visits a box its local board will be updated with 111, 222, 333 and 444 for first, second, third and fourth player respectively. The hurdles are represented with -2. When a tank is dead we make it another hurdle i.e. we represent it with -3 at the backend.

4.1.1 Global Board

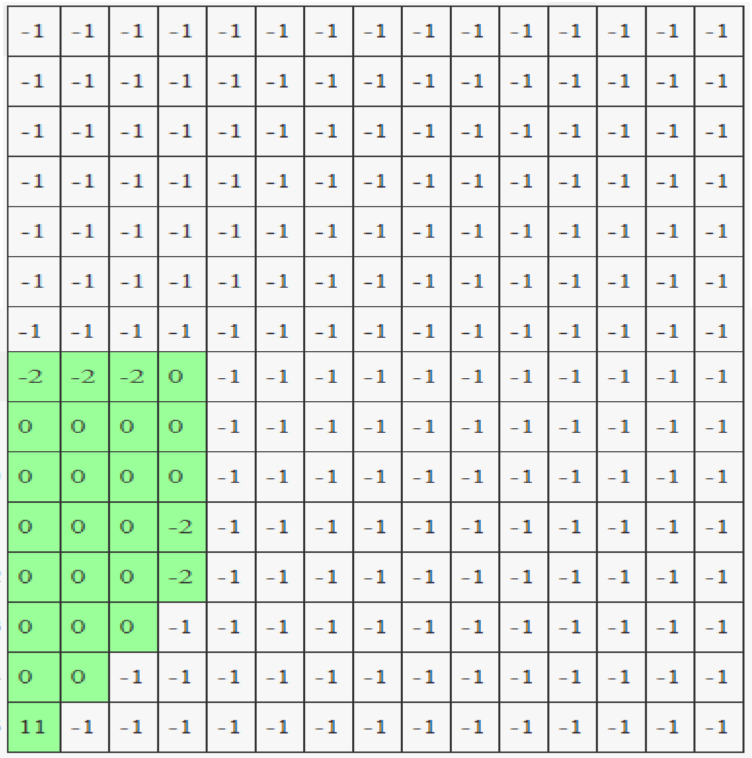
Under given table shows the backend representation of global board in the game.

Table 1: Global Board

0	33	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	22	0
0	0	-2	-2	0	0	0	0	0	0	0	-2	-2	0	0
0	0	-2	0	0	0	0	0	0	0	0	0	2	0	0
0	0	0	0	0	0	0	-2	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	-2	0	-2	0	-2	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	-2	0	0	0	0	0	0	0
0	0	-2	0	0	0	0	0	0	0	0	0	-2	0	0
0	0	-2	-2	0	0	0	0	0	0	0	-2	-2	0	0
0	11	0	0	0	0	0	0	0	0	0	0	0	44	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

4.2 Local Board of Agent

The backend representation of local board and the scope of AI agent is represented in table shown below. This local board gets initialize when the game starts.



-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-2	-2	-2	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
0	0	0	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
0	0	0	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
0	0	0	-2	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
0	0	0	-2	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
0	0	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
0	0	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
11	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1

Figure 5: Table 2: AI agent Local Board and Scope of Vision

4.3 UI of the Game

Following figure shows the front end representation of above mentioned global board.

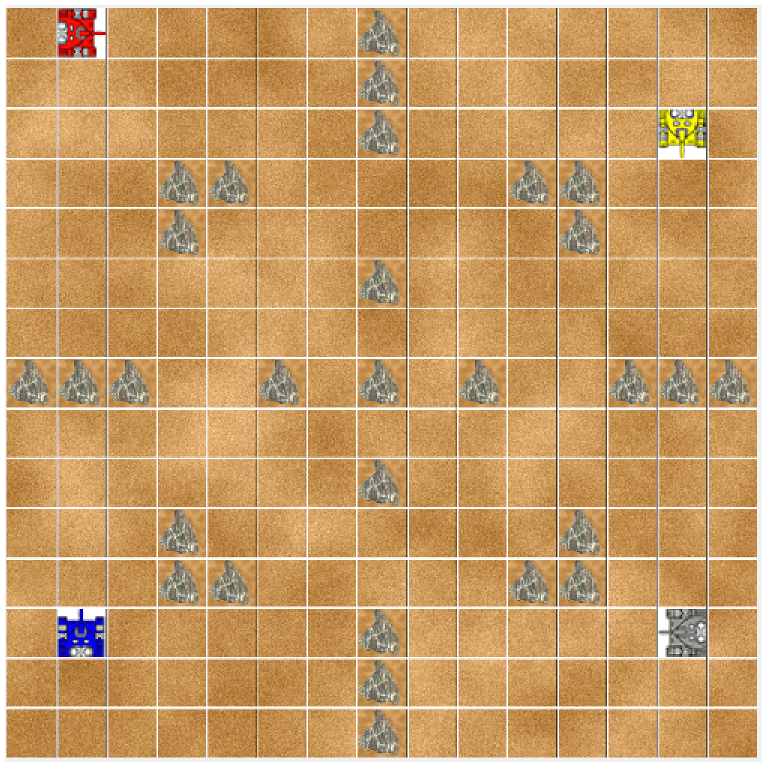


Figure 6: Game User Interface

4.4 AI Implementation

As discussed in the earlier sections of this document, AI of this game is implemented using Evolutionary Algorithm and Neural Networks. Since, there are four players in this game; for each player we have implemented some techniques to find out the next move against a particular turn. To make AI agent look more smarter and intelligent enough to win the game, combination of Evolutionary Algorithm and Neural Networks has been deployed. This combo only works for an AI agent. Generation of moves for the rest of players are being done by random functions. Under given sections provide a detailed overview about the implementation of the considered algorithm, in this game.

4.4.1 Random Move

As discussed earlier that three of the tanks will randomly take their moves. Now for that we have written a function which generate 7 numbers from 1 to 7 randomly. On the base of this random value the tanks take their move. Under given code snapshot shows how random move is being generated for these players.

```
function move = randomMove()
    move = randi(7);
end
```

4.4.2 Artificial Neural Network

Artificial Neural Network is implemented to take decisions of next move for AI agent. It takes local board of agent as an input and gives a set of seven outputs. The index with highest value is taken as a move decided by Neural Network. The architecture and environment variables of Neural Network used in this game is shown and discussed below.

```
% envriornmental setup
size_of_population = 250;
hidden_layer_size = 10;
input_layer_size = 225;
output_layer_size = 7;
scores = [];
```

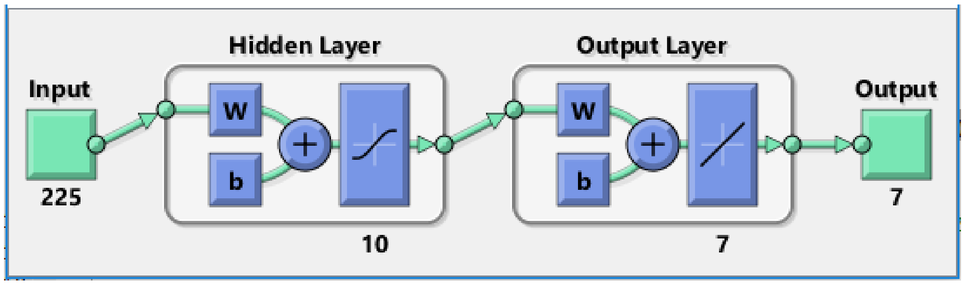


Figure 7: Artificial Neural Network Architecture

Since, the dimension for each local board and global board is 15x15 therefore, the size of input for a Neural Network is 225. There is also a hidden layer of size 10. All inputs are connected to each Neuron of hidden layer. Furthermore, each hidden layer Neuron is linked with output Neurons hence, making a Neural Network.

4.4.3 Neural Network Training

Neural Network requires a lot of training data to train itself, so that the best decisions can be made. During the training process, Neural Network establishes its weights and biases. Since, we are not given with any data set to train this Neural Network therefore, we are forced to generate either training data or we are required to set weights of Neural Network manually. Neural Network takes a lot of iteration and thousands of training inputs to obstinate its weights and biases. Generation of training input data will require us to play this game thousands of times and manual arrangement of weights would require us to do all of the complex computations and calculations that a Neural Network does on its own to estimate the best possible weights. Considering all of these complications, evolutionary algorithm is used under this project to set the best possible weights for a Neural Network. Following section accommodates a detailed analysis on the generation of best possible weights for a Neural Network.

4.4.4 Evolutionary Algorithm

Evolutionary Algorithm is a generic population based meta-heuristic optimization algorithm. There are many variants of evolutionary algorithm but the main idea behind all these techniques is to give birth to a population and select the fittest candidates to seed the next generation by applying recombination and mutation to them [3]. Under this project, Evolutionary Algorithm is being used in generation of best weights for a Neural Network. Under given flecks demonstrate how this algorithm has been implemented in order to set the best possible weights for a Neural Network.

- **Population Instantiation**
  Since, there is no training data for the initialization of weights and biases for a Neural Network as discussed above therefore, population instantiation is needed. This population contains the weights and biases, initialized randomly for Neural Networks. The number of weights and biases in a population is precisely dependent upon the size of input layer, hidden layers and output layer. Following computation shows

the way to compute the number of weights and biases for a Neural Network.

Under this project the size of a population is taken as 250, which means that a population contain $250xN$ numbers of weights and biases that represent 250 individual Neural Networks.

- **Creation of a Neural Network**
  After the initialization of a population of size 250, we create a Neural Network and set the random calculated weights and biases against each population index. Following code shows how weights of Neural Network is being set manually under this project.

```
% dummy input
xt = randi([-10 10], 225, 50);
tt = randi([1 7], 7, 50);
% create a neural network
net = newff(xt, tt, {10}, {'tansig'}, 'trainr', 'learngd');

% setting weights of neural network for input layer
net.IW{1, 1} = input_weights_matrix;

% set LW for neural network
net.LW{2, 1} = hidden_weights_matrix;

% set biases for input layer
net.b{1, 1} = input_biases_matrix;

% set biases for hidden layer
net.b{2, 1} = hidden_biases_matrix;
```

- **Fitness Evaluation**
  Once ANN is created using above mentioned methods we test this Neural Network against the three players of the game. Fitness evaluation uses this Neural Network to make the movement decisions for the AI agent. When the game gets complete, this function returns the score of AI agent under the considered Neural Network. This function is used to evaluates all of the weights and biases for a population. Following code shows how fitness is being computed for a population.

```
for p=1:size_of_population
    fprintf('Population_%d\n', p)
    net = generateNeuralNetwork(population{p},
        input_layer_size, hidden_layer_size,
        output_layer_size);
    scores(p) = evaluateFitness( net );
end
```

- **Cross Over**
  After computing fitness for a population of fixed size i-e 250, we sort the evaluated population in descending order. This sorting is based on the scores computed by the fitness evaluation function. Sorting of population allows us to pick those candidates that remained best while testing of Neural Network. In cross over, after sorting we make best pairs and exchange their weights by specifying a random limit. After cross over each pair produces its two descendants. After this process a new population is generated of same size i-e 250.

```
function population = crossOver( population, populationSize,
    numberOfWeights )

    for p=1:2:populationSize
        randomNumber = randi([ 1 numberOfWeights ]);
        dummy = population{p}(1, randomNumber:numberOfWeights);
        population{p}(1, randomNumber:numberOfWeights) =
            population{p+1}(1, randomNumber:numberOfWeights);
        population{p+1}(1, randomNumber:numberOfWeights) =
            dummy(1, :);
    end
end
```

- **Mutation**
  After the descendants are created by recombination, they are mutated. This resembles the naturally occurring accidents that can happen when DNA is copied. Mutation is being done by iterating over each weight of newly created population and generating a random number between 0-1. If the randomly generated number is less than or equal to 0.02, we reinitialize that particular index with a random number ranges between -15 to 15. So, the mutation rate in this project is 2 percent. Under given code shows the mutation being done on a given population.

```
function population = mutation( population, populationSize,
    number_of_weights )
    for p=1:populationSize
        for w=1:number_of_weights
            random_number = rand();
            if random_number <= 0.02
                random_weight = randi([-15 15]);
                population{p}(1, w) = random_weight;
            end
        end
    end
end
```

When the mutation gets done, implemented technique repeats the whole mentioned process infinitely until we set a limit. The initialization of weights is carried out only ones in this whole process. In this scenario, to set appropriate weights of Neural Network we went up to 30 generations. After reaching the specified limit we got a well trained Neural Network. Now, after receiving ANN, AI agent is ready to play game against opponents.

5 Testing and Analysis

Testing of AI agent is being done against opponent players by running Tanks game 5 times. Under given table shows the scores of players against each run of the game.

Table 2: Testing

Experiment	Agent Score	Player 2 Score	Player 3 Score	Player 4 Score
1	460	230	260	310
2	360	370	280	270
3	410	240	180	200
4	660	210	300	390
5	330	310	220	440

As mentioned above that the limit that we set for the generations was 30. The graph shown below represents the highest score of AI agent against each generation.

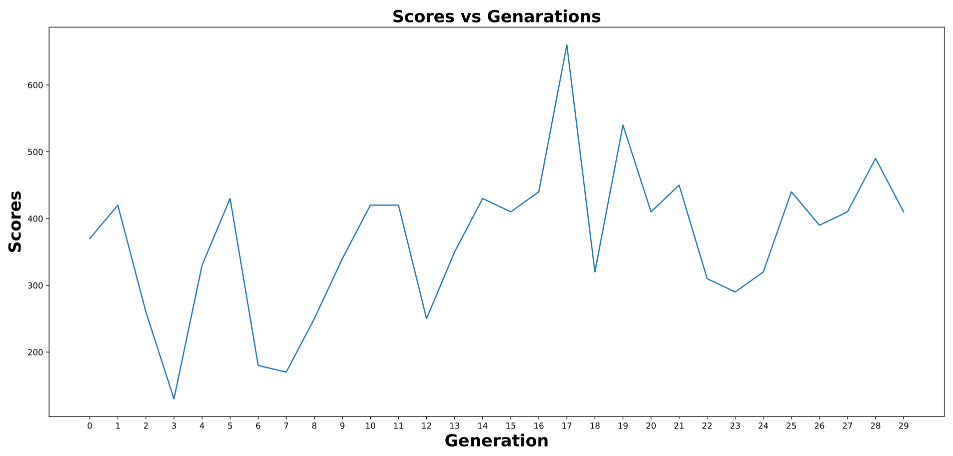


Figure 8: Highest Score in Each Generation

Following graph shows the mean score of each population in a generation.

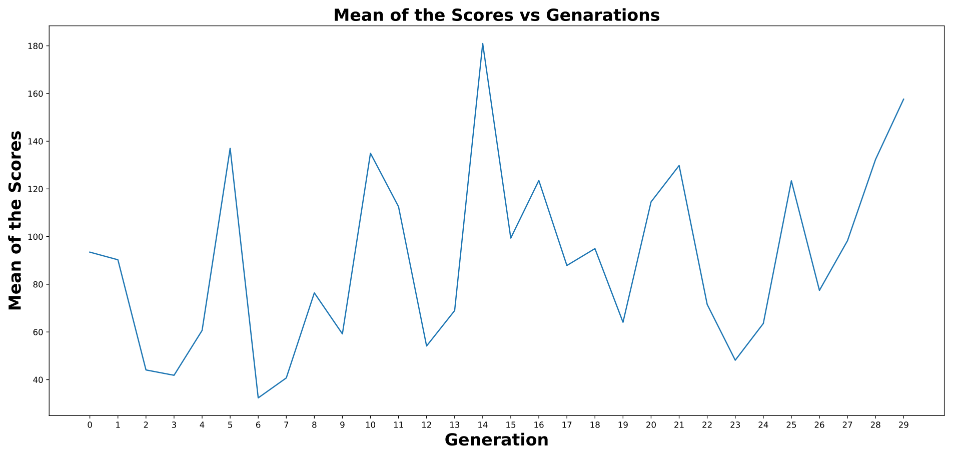


Figure 9: Mean Score in Each Generation

6 Appendix

Since, many functions are implemented to make AI agent intelligent. The main functions that are part of ANN and Evolutionary Algorithm are listed below.

Train Agent

```
% envriornmental setup
size_of_population = 250;
hidden_layer_size = 10;
input_layer_size = 225;
output_layer_size = 7;
scores = [];

%calculate total number of weights
number_of_weights = hidden_layer_size * input_layer_size +
    hidden_layer_size * output_layer_size + hidden_layer_size +
    output_layer_size;

%generate population according to size of population
population = generatePopulation(size_of_population,
    number_of_weights);

for i=1:30
    fprintf('Generation_%d\n', i)
    for p=1:size_of_population
        fprintf('Population_%d\n', p)
        net = generateNeuralNetwork(population{p}, input_layer_size,
            hidden_layer_size, output_layer_size);
        scores(p) = evaluateFitness( net );
    end

    [ scores, indexes ] = sort( scores,'Descend' );
    score_file = sprintf('scoreG_%d.mat',i);
    save(score_file,'scores');

    sorted_population = {};
    for p = 1:size_of_population
        sorted_population{p} = population{indexes(p)};
    end

    population_file = sprintf('populationG_%d.mat', i);
    save(population_file, 'sorted_population');

    population = crossOver( sorted_population, size_of_population,
        number_of_weights );
    population = mutation( population, size_of_population,
        number_of_weights );
end
```

Generate Initial Population

```
function population = generatePopulation( size_of_population,
    number_of_weights )
    population = cell(size_of_population, 1);
    for size=1:size_of_population
        population{size} = randi([-10 10], 1, number_of_weights);
    end
end
```

Generate ANN

```
function net = generateNeuralNetwork( set_of_weights,
    input_layer_size, hidden_layer_size, output_layer_size )

    % random generated input
    xt = randi([-10 10], 225, 50);
    tt = randi([1 7], 7, 50);
    % create a neural network
    net = newff(xt, tt, {10}, {'tansig'}, 'trainr', 'learngd');

    % calculating weights of input layer
    %no_of_input_layer_weights = input_layer_size *
        hidden_layer_size;
    %no_of_hidden_layer_weights = hidden_layer_size * 7;

    % fetch input layer weights and hidden layer weights
    %input_layer_weights = set_of_weights(1,
        1:no_of_input_layer_weights);
    %hidden_layer_weights = set_of_weights(1,
        (no_of_input_layer_weights+1):(no_of_input_layer_weights +
        no_of_hidden_layer_weights));
   % fetch input layer biases
    %input_layer_biases = set_of_weights(1,
        (no_of_input_layer_weights + no_of_hidden_layer_weights +
        1):(no_of_input_layer_weights + no_of_hidden_layer_weights +
        hidden_layer_size));
```

```
%hidden_layer_biases = set_of_weights(1,
    (no_of_input_layer_weights + no_of_hidden_layer_weights +
    hidden_layer_size + 1):(no_of_input_layer_weights +
    no_of_hidden_layer_weights + hidden_layer_size +
    output_layer_size));

% covert input layer weigths into IW matrix
input_weights_matrix = [];
n_previous = 1;
n_next = input_layer_size;
for i=1:hidden_layer_size
    input_weights_matrix =
        [input_weights_matrix;set_of_weights(1,
        n_previous:n_next)];
    n_previous = n_previous + input_layer_size;
    n_next = n_next + input_layer_size;
end
%disp(size(input_weights_matrix));
% setting weights of neural network for input layer
net.IW{1, 1} = input_weights_matrix;

% convert hidden layer weights LW matrix
hidden_weights_matrix = [];
n_previous = (n_next - input_layer_size) + 1;
n_next = (n_next - input_layer_size) + hidden_layer_size;

for i=1:output_layer_size
    hidden_weights_matrix =
        [hidden_weights_matrix;set_of_weights(1,
        n_previous:n_next)];
    n_previous = n_previous + hidden_layer_size;
    n_next = n_next + hidden_layer_size;
end

%disp(size(hidden_weights_matrix));
% set LW for neural network
net.LW{2, 1} = hidden_weights_matrix;

%set biases for input layer
input_biases_matrix = [];
n_previous = (n_next - hidden_layer_size) + 1;

input_biases_matrix = [input_biases_matrix;set_of_weights(1,
    n_previous:n_next)];
input_biases_matrix = input_biases_matrix';
```

```
    n_previous = n_previous + hidden_layer_size;
    n_next = n_next + output_layer_size;

    % set biases for input layer
    net.b{1, 1} = input_biases_matrix;

    %disp(size(input_biases_matrix))
    % set biases for hidden layer
    hidden_biases_matrix = [];
    hidden_biases_matrix = [hidden_biases_matrix;set_of_weights(1,
        n_previous:n_next)];
    hidden_biases_matrix = hidden_biases_matrix';

    %disp(size(output_biases_matrix))

    % set biases for hidden layer
    net.b{2, 1} = hidden_biases_matrix;
end
```

Fitness Evaluation

```
function score = evaluateFitness( net )

    rows = 15; cols = 15;
    turn = 11;
    total_turns = 1000;
    globalBoard = initializeBoard(rows, cols);
    globalBoard(1, 15) = 22;
    globalBoard(15, 1) = 11;
    globalBoard(1, 1) = 33;
    globalBoard(15, 15) = 44;

    p1 = {};
    p1{1} = 11;
    p1{2} = initializeBoard( rows, cols );
    p1{2}( 15, 1 ) = 11;
    p1{3} = 1;
    p1{4} = 10;
    p1{5} = true;

    p2 = {};
    p2{1} = 22;
    p2{2} = initializeBoard(rows, cols);
    p2{2}(1, 15) = 22;
```

```
p2{3} = 2;
p2{4} = 10;
p2{5} = true;

p3 = {};
p3{1} = 33;
p3{2} = initializeBoard(rows, cols);
p3{2}(1, 1) = 33;
p3{3} = 4;
p3{4} = 10;
p3{5} = true;

p4 = {};
p4{1} = 44;
p4{2} = initializeBoard(rows, cols);
p4{2}(15, 15) = 44;
p4{3} = 3;
p4{4} = 10;
p4{5} = true;

agentLocalBoard = zeros(15, 15);
agentLocalBoard = agentLocalBoard - 1;
globalBoard = placeRocks(globalBoard);
%find agent's local board

while ( total_turns > 0 )
    if turn ~= p1{1, 1}
        move = randomMove();
        if move == 7
           [ x, y ] = find(globalBoard == turn);
           agentLocalBoard( x, y ) = turn;
        end
    end
    if ( turn == 11 && p1{5} == true )
        agentLocalBoard = findVision( agentLocalBoard,
            globalBoard, p1{1, 1}, p1);
        % give best move use neural network
        move = generateAgentMove( net, agentLocalBoard );
       [ p1, p2, p3, p4, globalBoard ] = makeLegalMove(
           globalBoard, move, p1, p2, p3, p4);
       turn = changeTurn ( turn );
       %grid = boardToGrid( globalBoard, p1, p2, p3, p4 );
       %imshow( grid );

    elseif ( turn == 22 && p2{5} == true )
```

```
            [ p2, p1, p3, p4, globalBoard ] = makeLegalMove(
                globalBoard, move, p2, p1, p3, p4);
            turn = changeTurn ( turn );
            %grid = boardToGrid( globalBoard, p1, p2, p3, p4 );
            %imshow( grid );

        elseif ( turn == 33 && p3{5} == true )
            [ p3, p1, p2, p4, globalBoard ] = makeLegalMove(
                globalBoard, move, p3, p1, p2, p4);
            turn = changeTurn ( turn );
            %grid = boardToGrid( globalBoard, p1, p2, p3, p4 );
            %imshow( grid );

        elseif( p4{5} == true )
            [ p4, p1, p2, p3, globalBoard ] = makeLegalMove(
                globalBoard, move, p4, p1, p2, p3);
            turn = changeTurn ( turn );
            %grid = boardToGrid( globalBoard, p1, p2, p3, p4 );
            %imshow( grid );
        end
        total_turns = total_turns - 1;
    end

    score = p1{1, 4};
end
```

Cross Over

```
function population = crossOver( population, populationSize,
    numberOfWeights )

    for p=1:2:populationSize
        randomNumber = randi([ 1 numberOfWeights ]);
        dummy = population{p}(1, randomNumber:numberOfWeights);
        population{p}(1, randomNumber:numberOfWeights) =
            population{p+1}(1, randomNumber:numberOfWeights);
        population{p+1}(1, randomNumber:numberOfWeights) = dummy(1,
            :);
    end
end
```

Mutation

```
function population = mutation( population, populationSize,
    number_of_weights )
    for p=1:populationSize
        for w=1:number_of_weights
            random_number = rand();
            if random_number <= 0.02
                random_weight = randi([-15 15]);
                population{p}(1, w) = random_weight;
            end
        end
    end
end
```

References

[1] Floreano, D., Mattiussi, C.: Bio-inspired artificial intelligence: theories, methods, and technologies. MIT press (2008)

[2] Johnsson, S.: Ai till brädspel: En jämförelse mellan två olika sökalgoritmer vid implementation av ai tillbrädspelet pentago. (2014)

[3] Streichert, F.: Introduction to evolutionary algorithms. paper to be presented Apr **4** (2002)