Artificial Intelligence Algorithms and Techniques in the computation of Player-Adaptive Games

Dr.Ranjitha M [1], Kazaka Nathan [2] and Lincy Joseph[3]

[1,2,3] Department of Computer Science, Kristu Jayanti College(Autonomous), Bangalore ,India

Abstract. A game usually used as a synonym for entertainment also serves as an educational tool . Originally targeting the enjoyment, the meaning of the game has evolved to much greater conceptions and applications. Games require physical or mental or sometimes both the simulation. Many games help develop practical skills, serve as a form of exercise, or otherwise perform an educational, simulation, or psychological role. Having to be built on some key elements, which are goals, rules, challenge, and interaction. Various strategies and algorithms such as path finding and decision trees, have been developed to simulate those interactions between the human user and the computer in front of him. In video games, artificial intelligence (AI) is used to generate responsive, adaptive or intelligent behaviours primarily in non-player characters (NPCs) similar to human-like intelligence. This paper makes a short analysis of those preferred techniques and suggest from the study and the outcomes, the efficiency of each . The paper also focuses on the heuristic function, implementation platforms and design guidelines of the various searching algorithms used in adaptive games.

Keywords: Gaming Strategy; Gaming Algorithm; Artificial Intelligence; Player-Adaptive; Neural Network

1. Introduction

Artificial intelligence involves two basic ideas[1]. First, it involves studying the thought processes of human beings. Second, it deals with representing those processes via machines (like computers, robots, etc.). The field of game with applied intelligence has been more and more in the center of interest for multiple research. That is, machine learning techniques are employed in games with the aim of providing an entertaining and satisfying gaming experience for the human player. Research work in this area includes evolving racing tracks to suit a player [2], online driving adaptation in a racing game [3] and playing an even game in role playing game [4]. This survey paper puts together on a comparative goal numbers of games using an adaptive AI algorithm that can balance its level of difficulty according to the human player's level of reasoning. The various gaming algorithm to be discussed in this paper are: Alpha Beta Search Algorithm, Minimax and alpha-beta pruning, Evolutionary, Back Propagation (ANN), Monte Carlo Tree Search Algorithm and A* search algorithm have been applied on a list of selected games. This area seems to be of much interest for research as the games requires constant interaction, human intervention and quick decision making. The above mentioned algorithm's main goals are to identify all possible strategies and find the best one accordingly. But as in the case of games, which are non - deterministic as the settlers of Catan, the decision making becomes more difficult and cannot be chosen according to the probability as it reduces the fun of the game.

2. Methodologies Used To Design Gaming Algorithms

2.1. AI Algorithm For GOMOKU

Gomoku, also called five in a Row or "the game k-in-a-row", is an abstract strategy board game. It is traditionally played with Go pieces (black and white stones) on a Go board, using 15×15 of the 19×19 grid intersections. Two players, represented as B and W, alternately place one stone, black and white respectively, on one empty square (intersection) of an m by n board. B is assumed to play first. The player who first obtains k consecutive stones of his own color wins the game. (Horizontally, vertically or diagonally) [5]. The game is known to favour Black when played in the free style. Many variants of Gomoku exists, restricting the players in some sense to reduce Black's advantage [6].

2.1.1 GOMOKU-Design and Implementation

The details of the design of the Gomoku game and intelligence used will be discussed in this section. The game has been implemented using C++ and the rules followed are as follows.
Gomoku game engine was at first, designed and implemented to allow a basic play where 2 human players are involved to compete each, against the other. The illegal moves are defined under the following conditions: whether the column entered is valid, whether the row entered is valid or if the space is occupied. This game was implemented using a minimax search tree with alpha-beta pruning. [7] .However, even with alpha-beta pruning, the search space during each move is still very big due to the fact that players are allowed to play their stones on any unoccupied intersection on the board .To improve the computational search time of the minimax search, the alpha-beta pruning algorithm was implemented.

2.1.2 GOMOKU Activation Function

A heuristic function, inspired by the concept behind threat space search was developed for evaluating the payoff of each move. The kzey to winning the game is to create at least a double, in general. A heuristic value was assigned to one of each of the empty squares, stating how desirable and advantageous it would be to place the next move in that location and is determined by the sum of the value of all the threats created as a result of playing in that position. The heuristic function is therefore used to evaluate the state at the leaf nodes of the minimax search tree. The operation of the evaluation function is better illustrated using Figure 1 below. In the Figure I, a move at A result in only one four-in a row, hence the heuristic function returns a value of 50, whereas a move at B results in two four-in-a-row (i.e. a double threat), hence the heuristic function returns a value of 50+50=100. The values of each threat is arbitrarily chosen and summarized in Figure I. [8].

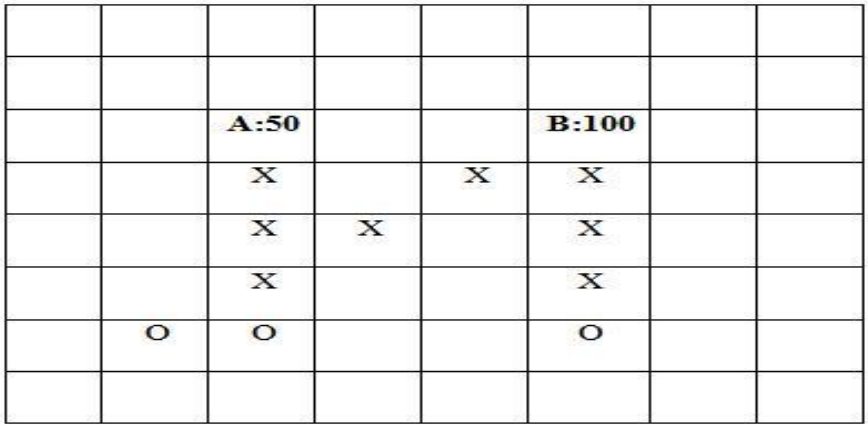

Figure I: Evaluation Function

2.1.3. Results- GOMOKU

The AI game for Gomoku was developed based on the minimax search tree with alpha-beta pruning algorithms, and on top of them was introduced an evaluation function discussed above inspired by the threat space search. The resulting algorithm was tested on the Gomocup which is a Gomoku AI tournament as well as by a group of human respondents to determine the effectiveness of the game AI. One more factor based on an offensive capability was introduced into the program to improve the AI performance and tested both on the Gomocup as well as on human players. The results provided some guidelines to design 8 levels of difficulty for use in the adaptive version of the AI. The proposed adaptive game AI was able to scale the level of difficulty and adapt its moves during the game based on how good the human player performs against it. The adaptive AI was play tested by 50 human respondents and they were each asked 3 questions after playing 5 games. The adaptive AI was able to scale up as well as down the level of difficulty to match up with 38 of the human candidates within 3 games and 43 of the candidates through a question answer survey have admitted enjoyed playing against the AI after it matched their own playing level.

2.2. AI Algorithm for OTHELLO

Reversi (marketed by Pressman under the trade name Othello) is a strategy board game for two players, played upon an 8 x 8 squared board without checkers (uncheckered). Accompanying the board, there should be 64 pieces, often called disks, each of which is dark on one side and light on the other.The basic rule of Othello, is that, if there are player's discs found in between opponent's discs, then the discs that belong to the player are taken off from him and become the opponent's discs. Othello has 3 main type of mode game that can be the player versus player, player versus computer, or even computer versus computer.

2.2.1. OTHELLO -Design and Implementation

The Othello game in this survey paper has been implemented using the C programming language and Alpha Beta Search algorithm is used for searching. A Neural Network has also been developed along with it. The rules for the game is that the Original Riversi stipulates that for the first four moves, the players must play to one of the four squares in the middle of the board and no pieces are captured or reversed. The objective of the game is to have the majority of disks turned to display your color when the last playable empty square is filled. Each piece played must be laid adjacent to an opponent's piece so that the opponent's piece or a row of opponent's pieces is flanked by the new piece and another piece of the player's color. All of the opponent's pieces between these two pieces are 'captured' and turned over to match the player's color. It can happen that a piece is played so that pieces or rows of pieces in more than one direction are trapped between the new piece played and other pieces of the same color. In this case, all the pieces in all viable directions are turned over. The game is over when neither player has a legal move or when the board is full.

The Alpha-Beta algorithm is a method for speeding up the minimax searching routine by pruning off cases that will not be used anyway. This method takes advantage of the knowledge that every other level in the tree will maximize and every other level will minimize. It works as follows: start off with $\alpha = -\infty$ and $\beta = \infty$ (alpha=-infinity and beta=infinity) ; traverse the tree until the depth limit is reached; assign a value for alpha or beta based upon what level preceded the depth limit level. Whenever max is called, it checks to see if the evaluation of the move it has been given is greater than or equal to beta. If it is, then max returns the value that would not have been chosen by min anyway and neither would the subtree that max would have created, as it is waste of time searching through them. The same logic is applied with min except that, for min, it checks if the move it has been given is less than or equal to alpha. The algorithm is given in Figure 2.

```
//board:current board boardition
//possibleMoves: search depth
//alpha:lower bound of expected value of the tree
//beta:upper bound of expected value of the tree
Int AlphaBeta(board,possibleMoves,alpha,beta)
{
If(possibleMoves=maxDepth||game is over) return Eval(board) // evaluate leaf board position from
     current player's standpoint
Result=-infinity;//present return value
possibleMoves=GeneratePossibleMoves(board);generate successor moves
for i=1 to count(possibleMoves) do //look over all the moves
{
     execute(possibleMoves[i]);//ecxecute current move
     value=_alphaBeta(board,possibleMoves+,-beta,-alpha);//call other player and switch sign of
returned value
if(value>result)result=value;// co pare returned value and result value, note new best result
 if necessary
    if(result>alpha)alpha=result;//adjust the search window
    Undo(possible_moves[i]);//retract current move
If(alpha>=beta)return alpha,// cuttoff
}
     return result
}
```

Pseudo-Code for Othello [10]

2.2.2 OTHELLO-Activation Function

In Neural Network [9] a function that is used to calculate activation function is a Sigmoid Function, the most commonly used one for calculating the Activation Function, where:

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

2.2.3. Results-OTHELLO

The algorithms and techniques used for the above Othello game has proved that Alpha-Beta algorithm coupled with an evolutionary neural network can produce a better quality static board evaluation rather than normal static board evaluation function. Moreover, from this, the testing proves that the produced AI is good enough to defeat the classic static board evaluation function that is using Negamax algorithm. [10]

2.3. AI Algorithm used in the SETTLERS OF CATAN

In this work, authors apply MCTS (Monte Carlo Tree Search) to the multi-player, non-deterministic board game Settlers of Catan. An agent that is able to play against computer-controlled as well as human Players is implemented here.

The Settlers of Catan, is a multiplayer board game designed by Klaus Teuber and first published in 1995 in Germany by Franckh-Kosmos Verlag (Kosmos). Players assume the roles of settlers, each attempting to build and develop holdings while trading and acquiring resources. Players gain points as their settlements grow; the first to reach a set number of points, typically 10, wins.

2.3.1. SETTLERS OF CATAN -Design and Implementation

It is implemented using Java software module named Smart Settlers. MCTS requires domain knowledge and it uses the following two methods-using non-uniform sampling in the Monte-Carlo simulation phase and modifying the statistics stored in the game tree. It takes the following into consideration: The starting position, the domain knowledge of Monte-Carlo simulations and Monte-Carlo tree search.

a. Effect of seating position: The results of their experiments were that the seating order effect introduces an unknown bias to the performances of agents. In order to overcome that, the seating order was randomized for all the next experiments where completely different agents were compared.
b. Domain Knowledge in simulation strategy : A balance between exploitation and exploration must be found in the simulation strategy and hence the weights were selected accordingly because , if the selection strategy is too deterministic, then the exploration of the search space becomes too selective, and the quality of the Monte-Carlo simulation suffers Hence the weights are adjusted accordingly .Here the probability of choosing the subsequent action depends on their weights however when this was done the actual performance of the agent dropped.
c. Domain Knowledge in tree search: Domain knowledge can be added to the tree aspects of MCTS. Here virtual wins are given to preferred ones and non-virtual wins are given to the non-preferred ones and here quite a limited amount of domain data was added. The addition of the virtual wins increased the playing strength of the agent.

2.3.2. Results- SETTLERS OF CATAN

The test is been conducted against JSettlers and Human.
With JSettlers: Here it was tested with three different AI's - a random player.
MCTS with 1000 simulated games per move and MCTS with 10000. For each AI 100 games are played. From the experiment it is concluded that the random player is very weak, MCTS with 1000 simulated games wins 27% of the games and MCTS with 10000 simulated games wins 49% of the games and has high score even when it does not win.
With Human: Against Human, the criteria is that the agent makes moves that must coincide with moves that a human would take. To make the analysis easier there were 2 strategies taken into consideration and it was found that the agent always followed only one strategy and the reason behind it probably would be because of the drawback of MCTS of not looking forward in the game to a sufficient depth. This drawback can be eliminated by increasing the number of games but that would decrease the speed and an alternative is to improve the selection criteria.[11]

2.4. AI Algorithm used in SIMULATED MOTOR CROSS

The techniques used to ride a simulated motor bike in AI is to be improved by improving the training given to AI using two algorithms -evolutionary and back propagation. To improve the back propagation Algorithm in this paper, two optimization techniques for augmenting the training are used: bagging [12] and boosting [13]. The Force is a motocross game featuring terrain rendering and rigid body simulation applied to bikes and characters.

2.4.1. SIMULATED MOTOR CROSS -Design and Implementation

There are a set of procedural rules in motor cross where in AI can work on. This is done with the help of Artificial Neural Networks. ANN's can adapt to a new change and can explore the path when presented with a new path. They are adaptable on their paths and can be evolved and trained. The ANN requires the following inputs:

- Position of the bike in way point space
- Front and right directions of the bike in way point space.
- Velocity of the bike in way point space.
- Height of the ground, ground samples in front of the bike relative to bike height.
- Position of track center lane, track center lane samples in front of the bike in bike space.

The outputs of the ANN gives the output of controls a human can use

- Accelerate, decelerate.
- Turn left, right.
- Lean forward, backward

Artificial Neural Networks: ANN's are software simulations that depicts human brain in some cases. Here Multilayered perceptron(MLP) is being used.The input layer of MLP is where the neurons are passive and hold the activation function to which the network must respond ie in this game the information of the terrain, the player is meeting. The output layer corresponds to the actions here such as turn left/right, accelerate/de-accelerate, and lean, forward/backward. Between these two is a hidden layer .The MLP is used in two phases: activation passing and learning.

2.4.2. SIMULATED MOTOR CROSS -Activation Function

Activation is passed from inputs to hidden neurons through a set of weights, W. At the hidden neurons, a nonlinear activation function is calculated; this is typically a sigmoid function The activation function is given as:

$$act_i = \sum_{j=1}^{n} w_{ij} x_j, \forall_i \in 1, \dots, H$$

$$h_{i=} \frac{1}{1+\exp(-act_i)}$$

Where hi is the firing of the i^{th} hidden neuron . This is then transmitted to the output neurons through a second set of weights , V so that:

$$act_i = \sum_{j=1}^{n} w_{ij} x_j, \forall_i \in 1, \dots, O$$

$$o_{i=} \frac{1}{1+\exp(-act_i)}$$

The activation function is passed from inputs to outputs and the whole machine tries to learn the mapping from input to output. Back propagation algorithm is as follows,

Let the P^{th} input pattern be $\mathbf{x}^P$, which after passing through the network evokes a response $\mathbf{o}^P$ at the output neurons. Let the target value associated with input pattern $\mathbf{x}^P$ be $\mathbf{t}^P$. Then the error at the i^{th}

output is $E_i^P = t^P{}_i - o^P{}_i$ which is then propagated backwards (hence the name) to determine what proportion of this error is associated with each hidden neuron. The algorithm is:

1) Initialise the weights to small random numbers
2) Choose an input pattern, $\mathbf{x}^P$, and apply it to the input layer
3) Propagate the activation forward through the weights till the activation reaches the output neurons
4) Calculate the δs for the output layer $\delta_i^P = (t^P{}_i - o^P{}_i) f^0(Act^P{}_i)$ using the desired target values for the selected input pattern.
5) Calculate the δ^P = PN $\delta P\delta$s for the hidden layer using $wji.f0(ActPi)$
   i j=1 j
6) Update all weights according to $\Delta_P w_{ij} = \gamma.\delta_i^P.o^P{}_j$ 7) Repeat steps 2 to 6 for all patterns.

An alternative technique for computing the error in the output layer while performing backpropagation has been investigated. Instead of computing the error as $(t^P{}_i - o^P{}_i)$, the error has been computed as $(t^P{}_i - o^P{}_i)|t^P{}_i - o^P{}_i|$. This has for effect to train the ANN more when the error is large, and allow the ANN to make more decisive decisions, with regard to turning left or right, accelerating or braking and leaning forward/back.

2.4.3. Results- SIMULATED MOTOR CROSS

Here it is used for the creation of training data made from a recording of the game played by a good human player. The targets are the data from the human player i.e. how much acceleration/deceleration, left/right turning and front/back leaning was done by the human player at that point in the track. The aim is to have the ANN reproduce what a good human player is doing. The human player's responses need not be the optimal solution but a good enough solution and, of course, the ANN will learn any errors which the human makes. The system is implemented with Inputs-50,One hidden layer, Neurons=80 in the hidden and three in the output layer,Weights=4240. The cuts for crossover has been increased from one to ten.

Six bikes are racing along track L, and therefore six ANN's are evaluated at any given time. The evaluation time has been set to 10 minutes, which means 30 minutes per generation. The number of generations has been set to 100, with a population of 18 ANN's, elitism of 0 (number of the fittest chromosomes being passed directly from the parent population to the child population), a mutation rate of 0.001, a crossover rate of 0.8, a perturbation rate of 0.5, probability to select average crossover over 10 cuts crossover set to 0.2.

The training can take a long time to perform; however there are big advantages in the evolutionary algorithm approach. The artificial intelligence can adapt to new tracks and improve lap times with time; it is also possible that it can eventually perform better than a good human player.
Using this technique, after 24 hours of training, ANN's average lap time can go down from 2 minutes 45 seconds on the long track, to approximately 2 minutes 16 seconds. Not all individuals in the population are performing equally well. For comparison a good human player's lap time is 2 Minutes 10 seconds [14]. Genetic Algorithm[15] with alternative crossover methods and a population made of mutated already trained ANN's, is a technique investigated here that has proved to produce good result. . Performance was as good as human player. One of the problem here was the acceleration with respect to the input.

2.5. AI Algorithm for FIFA EA Sports Game

FIFA, also known as FIFA Football or FIFA Soccer, is a series of association football video games or football simulator, released annually by Electronic Arts under the EA Sports label.

It's a football simulator game that is played between human and human or between human and computer AI. The AI is implemented as well as for the human team, in order to maintain the consistency in the football team among all the eleven players, as well as in the computer side.
Typically for these games (Team Sport Related) they use a few basic AI algorithms, like A-Star search. for working out how to get the players to the appropriate position; and enabling passing from one player to another. These basic capabilities are then layered to create the higher level strategy.

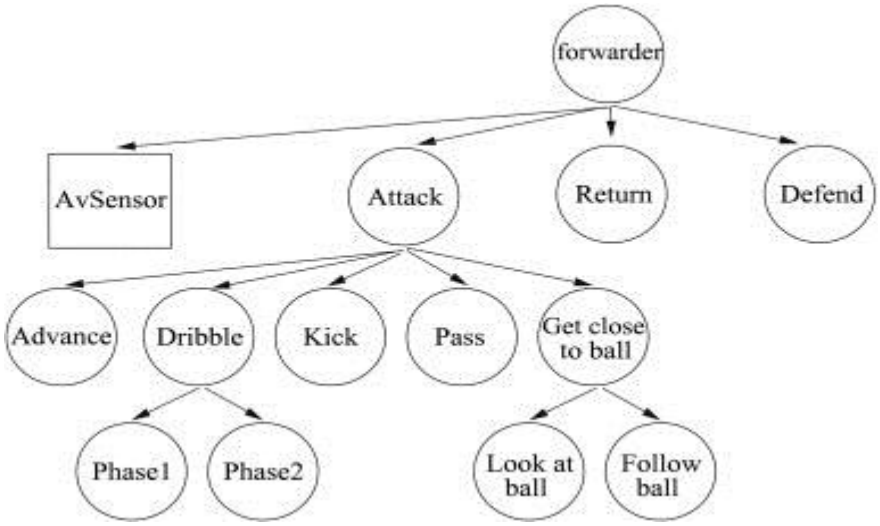

Figure 2. Diagram showing the idea of layered capabilities typically used in these games

A* Search algorithm, also known as A-Star Search algorithm, is generally used in pathfinding and graph traversal. It is widely used because of the high rate of performance and accuracy [16]. In sports game like FIFA, this algorithm is used for a better experience of gaming for the human player.
A* (pronounced "A-star") is a graph traversal and path search algorithm, which is often used in computer science due to its completeness, optimality, and optimal efficiency. One major practical drawback is its $\boldsymbol{O(b^d)}$ space complexity, as it stores all generated nodes in memory. Thus, in practical travel-routing systems, it is generally outperformed by algorithms which can pre-process the graph to attain better performance, as well as memory-bounded approaches; however, A* is still the best solution in many cases.
A* is an informed search algorithm, or a best-first search, meaning that it is formulated in terms of weig hted graphs: starting from a specific starting node of a graph, it aims to find a path to the given goal node having the smallest cost (least distance travelled, shortest time, etc.) [17]. It does this by maintaining a tree of paths originating at the start node and extending those paths one edge at a time until its termination criterion is satisfied.
At each iteration of its main loop, A* needs to determine which of its paths to extend. It does so based on the cost of the path and an estimate of the cost required to extend the path all the way to the goal. Specifically, A* selects the path that minimizes

$$\boldsymbol{f(n) = g(n) + h(n)}$$

where n is the next node on the path, $g(n)$ is the cost of the path from the start node to n, and $h(n)$ is a heuristic function that estimates the cost of the cheapest path from n to the goal.
A* terminates when the path it chooses to extend is a path from start to goal or if there are no paths eligible to be extended. The heuristic function is problem-specific. If the heuristic function is admissible, meaning that it never overestimates the actual cost to get to the goal, A* is guaranteed to return a least-cost path from start to goal [17].
In FIFA game, the A* Search Algorithm will help to analyse the proximity of the opposition player and space around them to identify the better passing opportunities. When the human player has the

ball, his teammates will understand if you have the chance to make the pass and move accordingly to create the chance for you to pass or to take the shoot. The smarter players and increased activity off the ball give the opportunity to open up the opposition for a better chance in the game. Other sports games like NBL work on the same algorithm.

3. Short Analysis

A Short analysis for the various gaming algorithms discussed in the previous sessions. The observations regarding each algorithm is also give in Table–I.

Table I-Short Analysis of different Games

Game	**Algorithm**	**Efficiency or computing time**	**Observation**	**Results**
Gomoku	MiniMax Search Tree and Alpha Beta Pruning	The first five AI moves take 56 seconds to execute. After implementation, the same first five moves only take 14 seconds.	The impact is greater later on in the game when the space became bigger, as the search tree will become bigger as well.	The developed AI played multiple sets of games with different rankings and has been able to win overall 75% of all its games.
Othello (Riversi)	Alpha Beta Search Algorithm	Speed up the calculation process evaluation board in the search tree.	Better than the actual used AI based on NegaMax algorithm	For the first testing, we get a 5 won in row against Tournament Reversi Program. From the second testing, we get 10 won from 13 games against a Negamax alpha beta pruning algorithm.
Settlers of Catan	Monte Carlo Tree Search	The playing strength of the agent is quite notable it defeats the hand-coded AI of JSettlers, and is a reasonably strong opponent for humans.	Considered as the best algorithm for the game	A random game is considered weak With 1000 simulated games an efficiency of 27% With 10000an efficiency of 49% was obtained
Motor Cross	Artificial Neural networks, Back Propagation algorithm Evolutionary algorithm	With evolutionary algorithm Artificial intelligence can adapt to new track and improve lap times with time; possibly it can eventually perform better than a good human player	Performance is as good as human players	Generic algorithm was found to improve the performance
FIFA	A* Search	Optimal efficiency is about the set of nodes expanded, not the number of node expansions (the	In such circumstances (Worst Case) Dijkstra's algorithm could	The smarter players and increased activity off the ball give the opportunity to open up the opposition for a

		number of iterations of A*'s main loop). When the heuristic being used is admissible but not consistent, it is possible for a node to be expanded by A* many times, an exponential number of times in the worst case.	outperform A* by a large margin.	better chance in the game.

4. Conclusion

The above work was to compare different AI techniques used in gaming environment such as Algorithms employing artificial neural network. In the case of algorithm driven AI games the algorithm used were minimax search tree and Monte Carlo Tree Search. The adaptive techniques used the Artificial Neural Network learning methods to determine the moves and the strategies. The observation made from algorithm based technique was that, the algorithm based techniques needed a clear domain knowledge to be effective and also, the larger the game develops, the algorithm was found to be much effective as well. In neural network based techniques we found that the inputs taken into consideration, decided the outputs thereby making the selection of inputs to be accurate and precise to obtain the expected results. The overall speed of the games however increased and became much more interactive.

References

[1]. K. Chitra, R. Alamelumangai, Artificial Intelligence, 2012.

[2]. Togelius, J., De Nardi, R. and Lucas, S. M., "Towards automatic personalized content creation\ for racing games," IEEE Symposium on Computational Intelligence and Games, pp. 252-259, 2007.

[3]. Tan, C. H., Ang, J. H., Tan, K. C. and Tay, A., "Online Adaptive Controller for Simulated Car Racing," Proceedings of IEEE Congress on Evolutionary Computation, pp. 2239-2245, 2008.

[4]. Spronck, P., Sprinkhuizen-Kuyper, I. and Postma, E., "Difficulty scaling of Game AI", 5th International Conference Intelligent Games and Simulation, pp. 33-37, 2004.

[5]. Wu, I-Chen, Huang, Dei-Yen and Chang, Hsiu-Chen, "Connect 6", ICGA Journal (SCI), vol. 28, no. 4, pp. 234-241, December 2005.

[6]. Allis, L.V., Herik, H. J. van den and Huntjens, M. P. H, "Go-Moku Solved by New Search Techniques", Computational Intellig.

[7]. Moriarty, D. E. and Miikkulainen, R., "Improving Game-Tree Search With Evolutionary Neural Networks", IEEE World Congress on Computational Intelligence, vol. 1, pp. 496-501, June 1994.

[8] Kuan Liang Tan, Chin Hiong Tan, Kay Chen Tan and Arthur Tay, "Adaptive Game AI for Gomoku"

[9] Alexander J. Pinkney, Development of an Artificial Neural Network to Play Othello, 2009.

[10] Gunawana, Hendrawan Armantoa, Joan Santosoa, Daniel Giovannia, Kurniawana, Ricky Yudiantoa and Stevena, "Evolutionary Neural Network for Othello Game"

[11] Istv´an Szita, Guillaume Chaslot, and Pieter Spronck, "Monte Carlo Tree Searches in Settlers of Catan"

[12] L. Breimen, "Bagging predictors," Machine Learning, no. 24, pp. 123– 140, 1996.

[13] J. Friedman, T. Hastie, and R. Tibshirani, "Additive logistic regression: a statistical view of boosting," Statistics Dept, Stanford University, Tech. Rep., 1998.

[14] Benoit Chaperot and Colin Fyfe," Advanced Artificial Intelligence Techniques applied to a Motocross Game".

[15] D. E. Goldberg, Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley Professional, 1989.

[16] Sultana, Najma & Chandra, Sourabh & Paira, Smita & Alam, Sk. (2017). A Brief Study and Analysis of Different Searching Algorithms.

[17] Potdar, Girish & Thool, Ravindra. (2014). Comparison of Various Heuristic Search Techniques for Finding Shortest Path. International Journal of Artificial Intelligence & Applications. 5. 63-74. 10.5121/ijaia.2014.5405.