Survey on Implementation of AI algorithms in 2-D Games

Mani Krishnan K
CSE Department,
SRM Institute of Science and Technology,
Vadapalani,Chennai,Tamilnadu,India.

Nandhakumar N
CSE Department,
SRM Institute of Science and Technology,
Vadapalani,Chennai,Tamilnadu,India.

Balaji J
CSE Department,
SRM Institute of Science and Technology,
Vadapalani, Chennai,Tamilnadu,India.

M.Indumathy
CSE Department,
SRM Institute of Science and Technology
Vadapalani, Chennai, Tamilnadu, India.

Abstract: The usage of Artificial Intelligence (A.I.) in games to design a non-playable agent is large. These agents, often known as NPC's (Non Playable Characters) are used to interact with the players to provide immersion. The field of Artificial Intelligence induced game development is flourishing due to recent advancements in the deep learning field. This study presents the effect of different Artificial Intelligence algorithms on games. For the purpose of this study, the games in which the Artificial agents are implemented are a puzzle game where the user is presented with a maze through which he/she has to navigate through to find the exit point and a 2-Dimensional platform game which consists of a game character that is tasked with avoiding obstacles that are generated randomly .

I. Introduction (*Heading 1*)

Algorithms play a major role in the making of a video game. In today's world, where games are evolving and have options to interact with the players, implementation of Artificial Intelligence plays a major role in the creation of agents that actively take part in the interaction process. This paper is a survey of 4 algorithms split between 2 games. The first game used here is a maze game and the second one is a 2-Dimensional platformer. The maze game's goal is for the player to reach the end node from the start node and the algorithms used to create an agent for doing the same are Breadth First Search and Depth First Search. These are graph traversal algorithms that take different approaches to solve the problem given. The second game, T-rex run, uses Neural Networks with genetic algorithm and Simulated Annealing.

II. Game Structure

A. Maze Solver

The Maze game algorithm application first presents the user with a field to enter the grid size. The user has options to set the start point and the end point and then customize the maze in any way possible. Then a button which performs the calculations is enabled. When the button is pressed, it performs the algorithms on the given maze and presents the result with the algorithm name and the time taken by the algorithm to complete the puzzle and highlights the algorithm that was able to complete first.

This application uses the OpenGL library [1] (GLU) to generate the maze and the other UI elements. OpenGL is a rendering and software package that can be used to perform many graphical renderings. It is used here to render the grids and the path traversed by the algorithms while the application is running.

B. T-Rex Jump

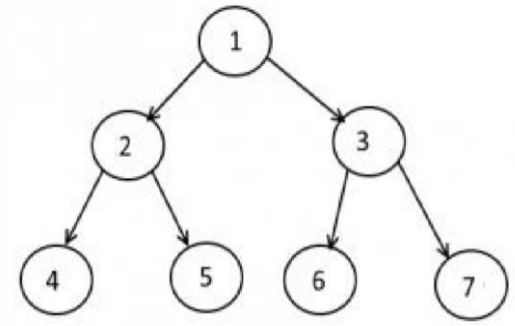
T-Rex Jump is a 2-Dimensional platformer jumping game where the player is tasked with avoiding obstacles that are generated randomly. This action of avoiding can be done by jumping the obstacles (cacti). The focus will be on the T-Rex (Dinosaur) which the player takes control of. The obstacles are generated randomly and the score is calculated based on the time spent by the player without hitting any obstacles.

This application uses Pygame [2], a wrapper for the Simple DirectMedia Layer library which provides access to the multimedia hardware components of the host computer. Asset files for the player and the obstacles can be imported using this library. The obstacles appear at random intervals and have a speed range between 200 and 500 pixels per second and the entire application runs on a 500 X 500 pixel resolution window.

III. Algorithms

A. Breadth First Search

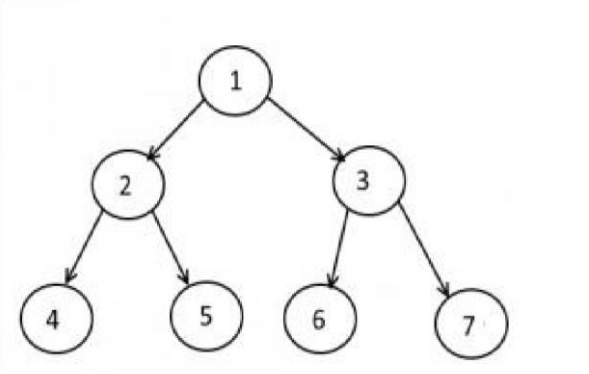
Breadth First Search (BFS) is a graph traversing algorithm. Graph traversing refers to the visiting of every single vertex and edge in the given graph exactly once in a defined order [3]. BFS starts at the source node or starting node and traverses the graph layer by layer exploring the neighbor nodes. Neighbor nodes are the nodes that are directly connected to the starting node. In future iterations, it moves layer wise thereby moving towards and visiting the neighbor nodes that are present in the next layer.



BFS Traversal Pattern - 1,2,3,4,5,6,7

B. Depth First Search

Depth First Search (DFS) is also a graph traversing algorithm that takes a different approach in visiting the entire set of nodes given. Instead of completing the graph layer by layer, DFS starts at the root node and visits as far as possible along each branch or neighbor node [4]. The algorithm then backtracks to the last visited neighbor node and then traverses branch-wise from that node.



DFS Traversal Pattern – 1,2,4,5,3,6,7

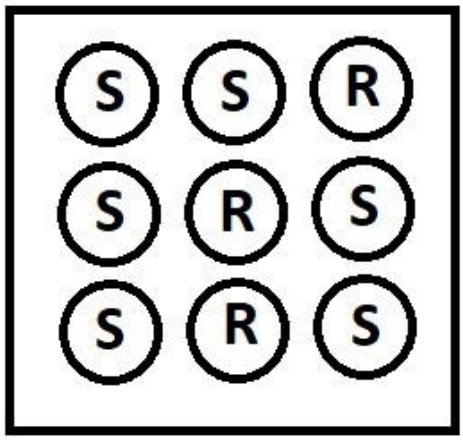
C. Genetic Algorithm

Genetic Algorithm (GA) is a programming methodology that is heavily inspired from biological evolution. It is a problem solving technique that is aimed at achieving an optimal solution. Genetic Algorithm is used and works best when not much is known about the problem. The only thing the algorithm requires is where the particular solution works best. They use selection and evolution to give out multiple solutions to a problem.

The input is a group of potential solutions that fit the problem. The GA then performs a set of functions.

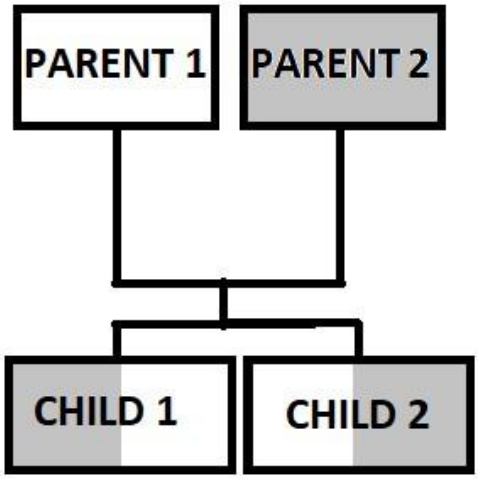
- **Selection**

A function called *fitness function* is designed to evaluate the candidates quantitatively. The GA then evaluates the candidates based on the *fitness function* and then proceeds to categorizing them. The candidates with the best performance are considered to be promising. These promising individuals are now sent to the next phase.



- **Crossover**

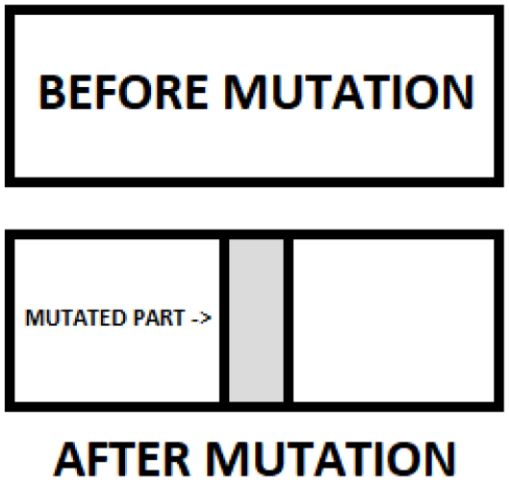
The promising candidates are now subjected to *crossover* [5]. During crossover, two random candidates are selected and they produce off springs which contain traits (chromosomes) of both the parents.



- **Mutation**

After the crossover process a new set of candidates are mixed with the old ones. Now, randomly they are selected for mutation [6], where they are allowed to develop new traits or change existing ones. This is to ensure diversity among the contenders.

These processes repeat until potential solution is obtained.



D. Simulated Annealing

Simulated Annealing [7] is an algorithm based on maximization and minimization methods which is best used in solving constrained and unconstrained problems and is aimed at finding an optimal solution to the problem put forward.

It is a probability based technique which is designed to approximate the global optimum of the problem. It starts by generating a random algorithm and then generates neighbors which are slightly modified from the algorithm generated previously. Acceptance criteria for this algorithm is based on the result provided by the previous neighbor generated.

IV. IMPLEMENTATION OF ALGORITHMS

A. Maze

For the maze game, BFS and DFS algorithms are used.The user selects the starting position and the ending position of the maze and also customizes the blocked and free paths. Once the starting and end nodes are specified, the algorithm execution begins, the starting node for the graph traversal happening with BFS and DFS is the starting point provided by the user and then the blocked paths act as leaf nodes. The algorithms then traverse throughout the grid and

then stop once the goal node is reached.If there are multiple solutions found, the algorithm takes the solution that took minimum number of steps to complete the maze.The application gives out the time taken by each algorithm to solve the maze.

B. *T-Rex Jump*

Genetic Algorithm and Simulated Annealing are tested on the T-Rex Jump game. These algorithms are coupled with an Artificial Neural Network that is provided with the required parameters to execute the algorithms. For the Genetic Algorithm, 100 instances of the game run at each iteration to finish the scoring and populate the candidate list. All the promising candidates are subjected to reproduction and randomly chosen for mutation. The mutation here happens by modifying the weights of the neural networks until the scores reach the acceptance criteria.

Simulated annealing here takes a temperature value which starts at 100 and goes down iteratively up to 10. Energy, a value obtained by taking the score of the current neighbor acts as a deciding point for the acceptance criteria. At first, the neural network runs for 50 different values of speed of the obstacle that are randomly generated to get a variable called *max-score* for the game. If the energy generated by the neighbor is lesser than the energy of the previous network, the new network is accepted else there is a probability of the network being accepted which is calculated from the current temperature and the change in energy.

V. OBSERVATIONS

A. *Maze*

For the purpose of this study, 15 randomly generated mazes were subjected to both the algorithms and the results are tabulated below.

Maze Number	Comparison of BFS and DFS			
	Time Taken(μ secs)		No of steps	
	BFS	DFS	BFS	DFS
1	17	16	31	31
2	22	30	36	56
3	33	55	36	48
4	18	16	36	36
5	1	25	1	51
6	7	36	10	10
7	8	10	26	40
8	5	3	7	7
9	21	9	18	18
10	33	46	25	27
11	33	27	23	73
12	5	40	6	84
13	30	55	18	18
14	32	42	22	64
15	22	37	12	88

From the results generated, we can see that BFS provides optimal solution for 10 mazes out of 15 while also taking minimum number of steps to solve the algorithm than when compared to DFS.

B. *T-Rex Jump*

Both Genetic Algorithm and Simulated Annealing paired with Artificial Neural Network (ANN) algorithms were run on the T-Rex game and the time taken for them to provide their best result was recorded.

It was observed that both the algorithms were able to train the AI agent in such a way that it was able to tackle every instance of the obstacle that it was subjected to. Therefore the time taken for the algorithms to train is taken into consideration for this comparison.

The Genetic Algorithm model was able to train the agent in 4 minutes and 2 seconds while the Simulated Annealing model took 2 minutes and 10 seconds to complete the training.

VI. CONCLUSION

Based on the observed results of the algorithms running on both the games, we can come to these conclusions

- Breadth-First-Search performs better than DepthFirst-Search in most of the cases by giving the answer in a shorter span of time along with the shortest path to the goal node. This may be because of the algorithm's tendency to explore all the possible paths simultaneously.
- Simulated Annealing was able to train the Neural Network faster when compared to Genetic Algorithm. This may be because of the Genetic Algorithm depending on the crossover and mutation functions to reach an optimal solution. Both the algorithms were able to train the network in such a way that the agent in the game always tackles the obstacles put in its path.

REFERENCES

[1] Shreiner, Dave, OpenGL Reference Manual. The Official Reference Document to OpenGL, Version 1.2.: The Official Reference Document to OpenGL, Version 1.4, 2004

[2] Pete Shinners (2011). PyGame - Python Game Development. Retrieved from http://www.pygame.org/, 2011

[3] Zhang, Feng & Lin, Heng & Zhai, Jidong & Cheng, Jie & Xiang, Dingyi & Li, Jizhong & Chai, Yunpeng & Du, Xiaoyong, An adaptive breadth-first search algorithm on integrated architectures, The Journal of Supercomputing, 2018

[4] Ms. Avinash Kaur , Ms. Purva Sharma , Ms.Apurva Verma ,A appraisal paper on Breadth-first search, Depth-first search and Red black tree, IJSRP, Volume 4, Issue 3, March 2014

[5] Andre, David and Astro Teller. "Evolving team Darwin United." In RoboCup-98: Robot Soccer World Cup II, Minoru Asada and Hiroaki Kitano (eds). Lecture Notes in Computer Science, vol.1604, p.346-352. SpringerVerlag, 1999.

[6] Thengade, Anita & Dondal, Rucha ,Genetic Algorithm – Survey Paper. IJCA Proc National Conference on Recent Trends in Computing, NCRTC. 5 , 2012

[7] Suman, Balram & Kumar, Prabhat ,A survey of simulated annealing as a tool for single and multiobjective optimization. Journal of the Operational Research Society, 2006.