# AI Game Bot

**Tejal Kadam[1], Archisha Chandel[2], Akanksha Dubey[3], Prof. Rahul Patil[4]**
[1, 2, 3, 4]Dept of Computer Engineering
[1, 2, 3, 4]Bharati Vidyapeeth College of Engineering, Kharghar, Navi Mumbai, India

***Abstract-*** *In this paper we present a generic method used for training a bot which learns to play the Nintendo game-'Mario'. We use the method of reinforcement learning for implementation. The goal is to build a bot that plays better than humans. This Artificial Intelligence-based tool can be used to simulate a human player. It automatically harvests resources and advances in the game without human intervention and can be used to compete against real-world players.*

## I. INTRODUCTION

When doing research in computational and/or artificial intelligence applied to games, it is important to have suitable games to apply the AI algorithms to. This applies regardless of whether one is doing research on using games to test and improve artificial intelligence (games provide challenging yet scalable problems which engage many central aspects of human cognitive capacity), or whether one is doing research on using CI/AI methods to improve games (for example with player satisfaction modelling, procedural content generation and creation of believable and interesting bots). No single game will satisfy all projects and directions, as different games pose different challenges. But, the community can gain from standardising on a certain set of games, which are freely available and on which competing AI methods can be easily compared.

The bot can be implemented using a simple reinforcement learning approach and using the NEAT algorithm. In the first approach, the bot is trained to play mario with the help of a reward-based system. Most training models have an explicit connection between inputs and outputs that are time-invariant. In environments where the elements, their associated inputs and attributes vary as a function of time and depend on the previous predictions. To overcome this problem, reinforcement learning accepts time variant data and uses a reward granting system to train the bot. The system grants positive rewards for correct actions taken by the agent (bot) and negative rewards otherwise. The system's goal is to obtain maximum rewards which encourage the agent to use actions that help it perform better.

The NEAT algorithm is an advanced approach which uses an evolutionary model to train the bot. NEAT consists of genomes and species. Each iteration of the algorithm is a generation. After each iteration the algorithm compares the fitness of each genome i.e. it checks how successful the genome is in the current generation. The species are evaluated on the basis of their fitness and the species that perform poorly are eliminated. The algorithm also creates new genomes by merging of two genomes or by the process of mutation of a single genome. This algorithm makes use of a "survival of the fittest" approach. Since this process is evolutionary it greatly improves the final performance of the bot.

We will be using Gym, an OpenAI product, to recreate the Mario environment which we will use to implement our reinforcement learning algorithm. Tensorflow is used to perform numerical computations using data flow graph. Our model will check for mario's postion and the total score. It compares the current state of the game to the previous state of the game and makes predictions based on the position of all the elements. For instance, in the game mario has to move forward in order to advance. Therefore, the machine grants positive rewards if mario has moved to the right and negative rewards if his position shifts left or if the lives run out. In this way the bot tries and makes the moves that contribute most to the total rewards. Thus, the machine learns to play, making predictions based on previous observations.

## II. BACKGROUND

A. Previous game-based competitions and benchmarks

Chess is probably the oldest known artificial intelligence benchmark, and has played an important role in CI/AI research since Turing first suggested that the game could be automatically played using the Mini Max algorithm. In the famous Kasparov vs. Deep Blue match in 1997, a computer program beat the human grandmaster for the first time and became a chess player, best in the world.

The exact significance of this event is debated, but what was proven beyond doubt was that an AI implementation can excel at a particular game without necessarily having a broad behavioural repertoire or being able to adapt to a variety of real-world challenges. The related board game Checkers (draughts), which was used for influential early machine learning experiments has recently been completely solved

using tree-search methods and can now be played perfectly (perfect play by both players leads to a draw). Another game where computers have already beat the human masters is Scrabble; the best Scrabble-playing programs (such as Maven) can win over all humans without searching more than one turn ahead, as for the benefit of quick and complete access to the dictionary.

1. GO

With the 2500-year-old Chinese game of Go, as the game has more possible positions than there are atoms in the universe. Last year, Google DeepMind's AI software AlphaGo played world Go champion Lee Sedol. In order to win, AlphaGo had to get good at the game – and it did. By playing itself millions of times, AlphaGo beats Lee Sedol in a game often seen as the "Holy Grail" of AI. Recently, Google's DeepMind has announced it will.

2. ATARI GAMES

DeepMind – a London-based AI company acquired by Google in 2014 – was presented with a few classic Atari games. Initially Google's DeepMind was not great but with improvements and time it started to play better than humans did.

3. DOTA 2

Open AI showed off the latest iteration of its *Dota 2* bots, which had been developed to the level of playing. Not only that but winning a full five-on-five game against humans. AI helped build and train models that could train themselves to the extent of perfection. The Open AI team won against well- known *Dota* personalities Ben "Merlini" Wu, William "Blitz" Lee, Ioannis "Fogged" Lucas — all former professional players — along with David "Moon Meander" Tan and play-by-play commentator Austin "Capitalist" Walsh.

4. PACMAN

Though AI had conquered over most Atari games, conquering over Pacman proved to be a little more difficult than expected due to its randomness. Microsoft Maluuba researchers developed a multi-agent reinforcement learning model to achieve top score in the game, something not achieved by humans before. In Maluuba's learning model, different characters track different goals with different payouts. A character tracks only a single object like pellet, fruit, ghost or edible ghost. For example, since ghosts can end the game by catching Pacman, the importance of recommendations of ghost tracking agents is of higher importance than other agents.

5. SUPER MARIO

A team from Georgia Institute of Technology also published a research paper describing their AI system that can recreate classic titles like Super Mario by watching Mario play. The algorithm does not have access to the codes and learns by just watching the character play. The re-creations it makes are not perfect but are passable.

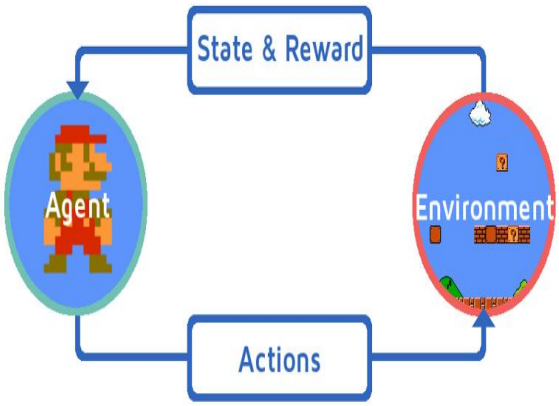
The AI system is not learning *everything* about the game, that is it does not familiarize itself with all the rules, obstacles, characters etc. It is given two very important sets of information. First is a visual dictionary and second is a set of basic concepts. It's supplied with two important sets of information: first, a visual dictionary featuring all the sprites in the game and of course secondly, a set of basic concepts, like the position of objects and their velocity, which it uses to analyze what it sees. With these tools in hand, the AI breaks down the gameplay frame-by-frame, labels what it sees, and looks for rules that explain the action.

B. Platform games as an AI challenge

Platform games can be defined as games where the player controls characters in an environment characterised by differences in altitude between surfaces ("platforms") interspersed by holes. The character can typically move in a straight (walk) and jump, and sometimes perform other actions as well; the game world features gravity, meaning that it is seldom straightforward to negotiate large gaps or altitude differences.

C. Learning

The objective of the game is to progress through levels by defeating enemies, collecting items and solving puzzles without dying. In the games, the player character (usually Mario) jumps on platforms and enemies while avoiding their attacks and moving to the right of the scrolling screen. *Super Mario* game levels have single-exit objectives, which must be reached within a time limit and lead to the next sequential level.

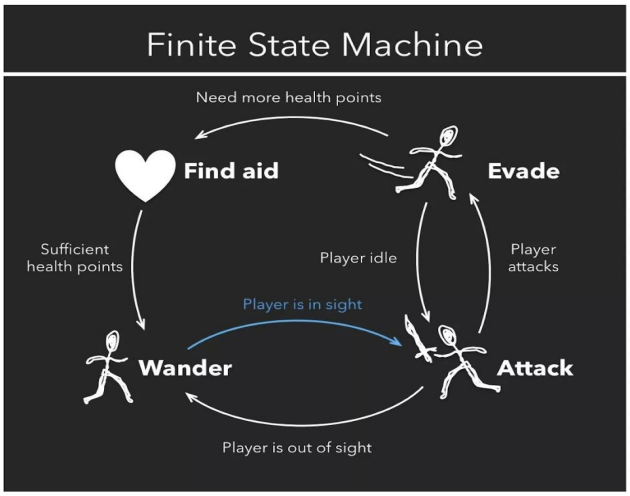



Levels and Worlds in Super Mario:

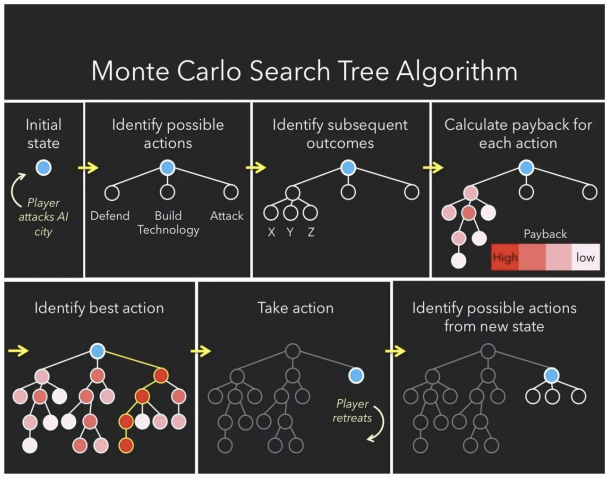
Super Mario World contains nine worlds and seventy-three (seventy-five if the Top Secret Area and Yoshi's House are counted as levels) levels in total, twenty-five of which have secret exits. Almost all worlds contain four regular levels and at least one secret level. Levels marked in yellow contain one exit, although levels marked in red contain an alternative, secret exit. Other points of interest include the Switch Palaces, Warp Pipes, and the Super Star-shaped portals to the Star World that are unlocked only when players find the associated secret exit. Switch Palaces activate respectively colored permeable Dotted Line Blocks and turn them into solid ! Blocks that can be stood on or hit from below. Once Switch Palace levels have been completed, they cannot be visited again.

## III. IMPLEMENTATION

In the gaming industry we don't want to create the best possible A.I., we want to create the most enjoyable A.I. for players to interact/compete with. The simplest algorithm which is finite state machine where designers create a list of all possible events a bot can experience. The designers then assign specific responses the bot would have to each situation.In finite state machine, each situation would be assigned a specific action by the developers creating the game. The Finite state machine algorithm is not feasible to use in every game. Just imagine using Finite state machine in a strategy game for example. If a bot was already programmed to respond the same way every time, the player would quickly learn how to outsmart the computer. This creates a repetitive gaming experience which, as you might expect, would not be enjoyable for the player.




The ***Monte Carlo Search Tree algorithm*** was created to prevent the repeatability aspect of FSM. The way MCST works is by first visualizing all of the possible moves a bot has available to it currently. Then, for each of those possible moves, it analyses all of the moves a player could respond with, then it considers all of its possible responding moves it could make in response, etc.




**The Problem with Neural Networks :** Neural Networks have helped us solve so many problems. But there's a huge problem that they still have. Hyperparameters! These are the only values that cannot be learned… Until now. **Note: Hyper-parameters are values required by the NN to perform properly, given a problem. We can use Genetic Algorithms to learn the best hyper-parameters for a Neural Network!** Now, we don't have to worry about "knowing the right hyperparameters" since, they can be learned using a GA. Also, we can use this to learn the parameter's(weights) of a NN as well.

**Genetic Neural Networks** in Video Games implements the reward system where there exists certain set of

inputs and there is a desired state that the agent has to achieve and on getting closer to the desired state it achieves a reward. Neuro Evolution of Augmenting Topologies is a type of genetic algorithm. The way in which we encode our individuals lays out the path and show how our algorithm will handle the key evolutionary processes of selection, mutation, and crossover (also known as recombination). Every encoding will fall into one of two categories, direct or indirect. The direct encoding will explicitly specify everything about an individual. If the result represents a neural network this means that each gene will directly be linked to some node, connection, or property of the network. This can be a binary encoding of 1s and 0s, a graph encoding (linking various nodes by weighted connections), or something even more complex. The point is that there will always be a direct connection between genotype and phenotype that is very obvious and readable. An indirect encoding is the exact opposite. The indirect encodings tends to specify rules and parameters of processes for creating an individual, instead of directly specifying how a structure may look like . As a result, indirect encodings are much more compact. The flip side is a setting the rules for an indirect encoding and hence can result in a heavy bias within the search space, thus, without substantial knowledge about how the encoding will be used, it is much harder to create an indirect encoding . The Neuro Evolution of Augmenting Topologies algorithm chooses a direct encoding methodology because of this. The Input nodes and output nodes are not evolved in the node gene list. Hidden nodes can be added or removed. And then connection nodes, they specify where a connection comes into and out of, the weight of such connection, whether or not the connection is enabled, and an innovation number.

**A generational neural network** is structured the same way as a standard neural network. It starts off with a certain number of input nodes, which then feed into one or more hidden layers, eventually providing an output. Weights are assigned to the paths of actions that could be chosen by the agent and weights are what our model will be adjusting as it learns which inputs to strengthen or weaken to provide the most accurate outputs. The way a generational neural network 'learns' is by first deciding the size of each generation.

Next, it creates micro-variations in the weights for each of the paths in the first generation, then it runs each of the paths in the first generation and selects the most successful path (the path that received the most points). Let's say we pick the top 10 paths (top 5%) that received the most points in our first generation. These 10 paths then become the 'parents' of the second generation. The weights of these 10 paths are used to define the second generation's starting point. The second generation of 200 paths will again create micro-variations in these weights and the top performers will be selected as 'parents' of the third generation, and so on.

One of the latest advancements of A.I. was made by researchers at Open AI. Although Open AI created a game based on an algorithm whose sole purpose was simply to explore with a sense of natural curiosity. The reward system focused on rewarding exploration over progressing further into the game. The researchers placed this curiosity-driven model into a game of Super Mario Bros. and it successfully passed 11 levels out of pure curiosity. An yes obviously, there are downsides to this, as it takes immense computing power and the machine can get easily distracted. However, this would also be the same for a human player playing the game for the first time. As Luzgin quoted in his article, "babies appear to employ goal-less exploration to learn skills that will be useful later on in life."

In order to select good actions over bad, our agents must continually make value judgements. The total reward is represented by a Q-network that estimates the overall reward that an agent can expect to receive after taking a particular action. Mainly we use deep neural networks to represent the Q-network, and to train this Q-network to predict total reward. Previously all attempts to combine Reinforcement Learning with neural networks had largely failed due to unstable learning. To address such instabilities, our Deep Q-Networks algorithm stores all of the agent's experiences and then randomly samples and replays these experiences to provide diverse and de-correlated training data. We applied Deep Q-Network to learn to play games on the Atari 2600 console. Every time-step the agent observes the raw pixels on the screen, a reward signal corresponding to the game score, and selects a joystick direction. In our Natural paper we trained separate Deep Q-Network agents for 50 different Atari games, without any prior knowledge of the game rules.

However, deep Q-networks are only one way to solve the deep Reinforcement Learning problem. We recently introduced an even more practical and effective method based on asynchronous Reinforcement Learning. This approach exploits the multithreading capabilities of standard CPUs. Basically the idea is to execute many instances of our agent in parallel, but using a shared model. This provides a viable alternative to experience replay, since parallelisation also diversifies and de-correlates the data.

Our asynchronous actor-critic algorithm, combines a deep Q-network with a deep policy network for selecting actions. Eventually achieves state-of-the-art results, using a fraction of the training time of DQN and a fraction of the resource consumption of Gorila. And hence, by building novel

approaches to intrinsic motivation and temporally abstract planning, we have also achieved breakthrough results in the most notoriously challenging Atari games, such as Montezuma's Revenge.

## IV. CONCLUSION

A method for procedurally generating maps using variations of Markov chains as a tool for both learning and generation has been developed. This method learns from an established high quality maps in order to generate statistically similar maps. We have incorporated look-ahead, backtracking, and a fallback strategy into our map generation method. This improves the quality of the generated maps, by allowing the use of higher order Markov chains whenever possible, and only defaulting to lower order Markov chains when necessary. Lastly, our method also includes the ability to split the maps used for learning into different horizontal slices. Doing so allows our method to isolate certain qualities of the map that may be exclusive to specific portions of the given maps. Our method gave strong results. Using our baseline configuration, the A $*$ controller was able to complete 44% of maps generated (which does not necessarily mean that the remaining 56% were not playable). We would like to emphasize, in contrast with search-based generation procedures, map generation using our method (when small values for d like 0, 1, 2 or 3) is almost instantaneous, making it amenable for in-game uses. Furthermore, adding a fallback strategy and a different order of generation has helped ensure that the maps generated do not have any ill-formed structures. Notice that we did not include any sort of additional hard-coded knowledge in our method, and that the generated maps are one-hundred percent generated based on the learned probability distribution in the Markov chain. This shows that Markov chains are a viable method for procedurally generating playable, well-formed maps. If all these methods were to be incorporated into an actual game, additional rules to detect malformed structures and unplayability should be added.

In the future, we want to explore better ways to judge whether a map is playable and also to perform user studies to determine whether the maps are actually enjoyable. Additionally, we have currently only generate the level layout, without including enemies; which we plan to experiment with in the future, by modeling enemies as just another type of tile. Finally, we would like to experiment with applying naive Bayes approximations to very high-order Markov chains (e.g. order 10 or even 20), to explore whether higher order dependencies compensate for the loss in the approximation of the probability distributions during map generation. We also plan to explore hierarchical models for learning the overall structure of the map with all the details.

## REFERENCES

[1] S. Bakkes and J. Dormans. It involves player experience in dynamically generated missions and game spaces. In the Eleventh International Conference on Intelligent Games and Simulation (Game-On'2010), pages 72–79, 2010.

[2] W.-K. Ching, X. Huang, M. K. Ng, and T.-K. Siu. Higher-order markov chains. In Markov Chains, pages 141–176. Springer, 2013.

[3] K. Compton and M. Mateas. Procedural level design for platform games. In Proceedings of the Artificial Intelligence and Interactive Digital Entertainment International Conference (AIIDE), 2006.

[4] M. Hendrikx, S. Meijer, J. Van Der Velden, and A. Iosup. Procedural content generation for games: a survey. The ACM Transactions on Multimedia Computing, Communications, and Applications (TOMCCAP), 9(1):1, 2013.

[5] W. Lam and F. Bacchus. Learning bayesian belief network which is an approach based on the mdl principle. Computational intelligence, 10(3):269–293, 1994.