An Approach to the Development of a Game Bot based on Reinforcement Learning

E. Saraswathi, Riyaz Ahmed, Vaibhav Kanna, Ashwin Kumar Thachat

Abstract: ***The current researches for developing a Game Bot for digital games have been studied thoroughly. The bots created based on those studies play the game exactly as an expert would. The bots decide their next move by overlooking a number of moves ahead of the next move. Our approach of creating a Game Bot requires low computational resources without compromising the difficulty and expertise of play. We have proposed to develop our Game Bot through Reinforcement Learning. Reinforcement Learning is an exciting new field of machine learning, in which Bots learn by playing games. The bots are thrown in a gaming environment, and then trained to learn by observing their actions and rewards. We are going to build a Game Bot that will autonomously play against and beat the Atari game Neon Race Car.***

Keywords: Game Bot, Reinforcement Learning, Machine Learnin.

I. INTRODUCTION

All the recent researches that create Game Bot have been studied thoroughly. Google's Deep Mind team developed a game agent called "Alpha Go".

Another game agent, "AKARA" was developed by Information Processing Society of Japan. This game agent plays Shogi, deciding the next move using the algorithm based on council system.

The most recent breakthrough in AI bots was when a team called OpenAI created a Game Bot that could play the online multiplayer game DOTA (Defense Of The Ancients). The OpenAI DOTA Game Bot beat the dream team of DOTA (2018) consecutively on three straight games with 92% win rate. All the players agreed the Open AI's DOTA Game Bot to be the hardest opponent they've ever faced.

Our Game Bot is going to be similar to that of the OpenAI's Game Bot in a way that both of them use Reinforcement Learning unlike many other Game Bots that decide their moves using high Computational Algorithm. Our Bot is not going to need much of Computational Resources. The bot is going to learn the game by itself by repeatedly playing the game inside the game environment that has been fed to it.

Revised Version Manuscript Received on 30 October 2018.

Mrs. E. Saraswathi, Assistant Professor, Faculty of Engineering & Technology, Department of Computerscience and Engineering, SRM Institute of Science and Technology, Chennai (Tamil Nadu), India.

Riyaz Ahmed, Undergraduate Students, Department of Computer Science and Engineering, SRM Institute of Science and Technology, Chennai (Tamil Nadu), India.

Vaibhav Kanna, Undergraduate Students, Department of Computer Science and Engineering, SRM Institute of Science and Technology, Chennai (Tamil Nadu), India.

Ashwin Kumar Thachat, Undergraduate Students, Department of Computer Science and Engineering, SRM Institute of Science and Technology, Chennai (Tamil Nadu), India.

The only input that are going to be fed are the game environment and the controls. In this research, we target the development of a bot that plays and beats the game "Neon Race Car". Our method won't be having tree search as we want to reduce or completely neglect the search capability.

Reinforcement learning (RL) is the subfield of machine learning concerned with decision making and motor control. It studies how an agent can learn how to achieve goals in a complex, uncertain environment. RL is very general, encompassing all problems that involve making a sequence of decisions: for example, controlling a robot's motors so that it's able to run and jump, making business decisions like pricing and inventory management, or playing video games and board games. RL can even be applied to supervised learning problems with sequential or structured outputs.RL algorithms have started to achieve good results in many difficult environments. RL has a long history, but until recent advances in deep learning, it required lots of problem-specific engineering. Deep Mind's Atari results, BRETT from Pieter Abbeel's group, and Alpha Go all used deep RL algorithms which did not make too many assumptions about their environment, and thus can be applied in other settings.

However, RL researches is also slowed down by two factors. The need for better benchmarks. In supervised learning, progress has been driven by large labeled datasets like Image Net. In RL, the closest equivalent would be a large and diverse collection of environments. However, the existing open-source collections of RL environments don't have enough variety, and they are often difficult to even set up and use. Lack of standardization of environments used in publications. Subtle differences in the problem definition, such as the reward function or the set of actions, can drastically alter a task's difficulty. This issue makes it difficult to reproduce published research and compare results from different papers.

II. RULE OF GAME

A. Common Rules of The Game of Neon Run Are Quite Different in Different Areas.

Because of graphical and physical limitations, the analysis is performed according to the following rules. The game of neon run is played by a single player facing the race court environment trying to hold the 1st position or completing the circuit in the least time possible. On the circuit, there is a fixed path over which the car can accelerate and the sides of that path, coming to contact which, will start to decelerate the car. Initially, the car starts at the starting point. There will be other bots that race along with the player. Crashing into other cars will cause the player's car to decelerate.

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication

The objective is to complete the race circuit finishing first place. The following will be the controls: accelerate, brake, turn left and turn right. The game stops when the player crosses the finish line. The winner is the player who crosses the finish line first.

B. Suggested Rules

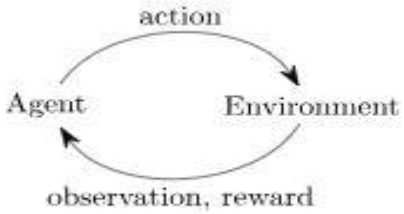
Before building a reinforcement learning agent, it is necessary y to check if the game can be solve dbya simple code with the fed instructions. The game is played using reinforcement learning. The game agent is coded in python. The coded script is integrated with the game and the bot it allowed to learn the game. The progress is noted for every attempt till the bot beats the game. It is always suggested that the bot avoids contact with the sides of the road and crashing into another car.

III. METHOD

A. Reinforcement Learning with Neural Network

The algorithm mde scribed in this section saves computational time and to makes it more septic to Neon Race Car. Consider an agent working in an environment. At time t, the agent is in state, st. The agent can observe the environment's internal state and denote this with A(st). Based on the observation made, the agent can then take an action, at. The action is taken by the agent that changes the environment's internal state to st+1 and then the agent either fails or gets a reward, rt. The environment used her is Atari's game – Neon Race Car.

First, we import the gym and openai universe. Gym is a toolkit for developing and comparing reinforcement learning algorithms. It makes no assumptions about the structure of your agent, and is compatible with any numerical computation library, such as Tensor Flow or Theano. The gym library is a collection of test problems — environments — that you can use to work out your reinforcement learning algorithms. If you prefer, you can also clone the gym Git repository directly. This is particularly useful when you're working on modifying Gym itself or adding environments. These environments have a shared interface, allowing you to write general algorithms. We will have to have pip and git installed. There are two coded python files to solve the gym environment. One contains the config for the neuroevolution process and the other one is the program that creates the neural networks and solves the game. In the first file, there will be an option to insert or change the environment of the game. It can be any game on the openai gym website. In this case, we are going to be using the Atari Neon Race Car game environment. The following will be the parameters: maximum steps (the maximum number of steps to take per single genome), episodes (the number of times to run a single genome), render (to render the game while the algorithm is learning), generations (the number of generations to evolve the network), checkpoint (to create a save point to start or resume the simulation) and num-cores (the number of cores to be allotted by the system for parallel execution). If you want to change the game, you will need to edit a few parameters. As an example, let's say we want to play Pacman. In order to play Atari games, we must use the ram version. Currently only ram versions are compatible with our program. Our program also provides the ability to continue a simulation after it finishes. When the simulation finishes, it will generate a checkpoint file. If we start a new simulation on the same game, we'll be able to use this checkpoint file to pick up where the simulation left off.



If we ever want to do better than take random actions at each step, it'd probably be good to actually know what our actions are doing to the environment. The environment's step function returns exactly what we need. In fact, step returns four values. These are: observation (object): an environment-specific object representing your observation of the environment. For example, pixel data from a camera, joint angles and joint velocities of a robot, or the board state in a board game; reward (float): amount of reward achieved by the previous action. The scale varies between environments, but the goal is always to increase your total reward; done (boolean): whether it's time to reset the environment again. Most (but not all) tasks are divided up into well-defined episodes, and done being True indicates the episode has terminated; info (dict): diagnostic information useful for debugging. It can sometimes be useful for learning. However, official evaluations of your agent are not allowed to use this for learning. This is just an implementation of the classic "agent-environment loop". Each time step, the agent chooses an action, and the environment returns an observation and a reward.

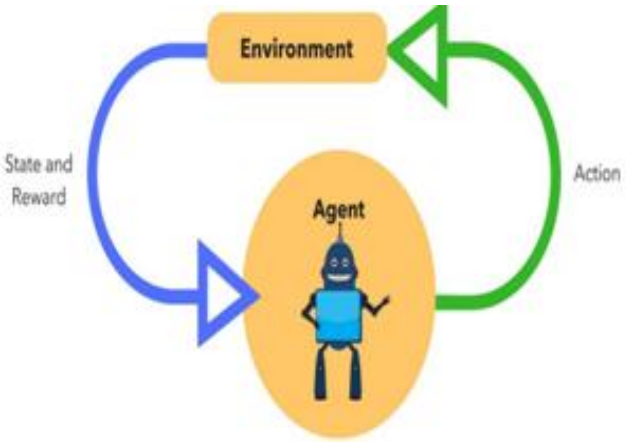
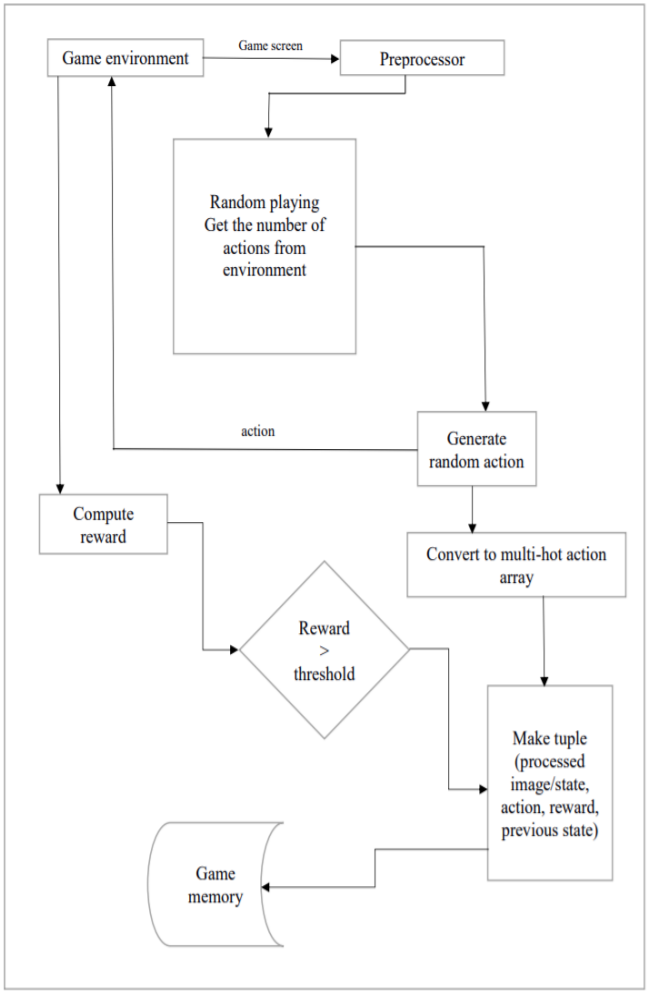


Fig. 1. The Exact Architecture of the Game BOT is as Follows:



This shallow architecture was designed in such a way so as to save the computational cost.

The algorithm initially, generates a random action. The generated action is tested in the environment and the results are observed. A reward is computed for every action and its reaction. If the reward is greater than the threshold, then the record is inserted as a tuple to the game memory. Else, another random action is generated by the algorithm and this process keeps repeating itself until the algorithm completely learns to beat the game and actually beats it.

IV. RESULTS

Initially, the game bot played the game in a very insensible manner. It was very hard to keep track of the actions that were generated and performed. At times, the bot just accelerated the car without steering left or right and bumped into another car or the side of the road. It kept doing the same action till it reached the finish line or ran out of time. The bot had lost and then tried to include a steering into the game. Then a brake and so on. After hours of learning the game, the game bot was able to perfect the game in every manner and was able to race the car first to the finish line with no hits or bumps. The bot has become a professional Artificial Intelligence gamer for that particular game.

REFERENCES

1. Muhammad Firmansyah Kasim Department of Physics University of Oxford Ox ford OX1 3RH, United Kingdom "Playing the Game of Congklak with Reinforcement Learning", 8th International Conference on Information Technology and Electrical Engineering (ICITEE), Yogyakarta, Indonesia, 2016.
2. Paulo Bruno S. Serafim, Yuri Lenon B. Nogueira, Creto A. Vidal, Joaquim B. Cavalcante Neto Department of Computing Federal University of Cear´ a Fortaleza, Brazil "OntheDevelopmentofanAutonomousAgentfora3DFirst-PersonShooter Game Using Deep Reinforcement Learning ", Brazilian Symposium on Computer Games and Digital Entertainment, 2017.
3. Etienne Perot Valeo, Maximilian Jaritz Valeo/ Inria, Marin To roman off Valeo, Raoul de Charette Inria, "End-to-EndDrivinginaRealisticRacingGamewithDeepReinforcementLearning", IEEE Conference on Computer Vision and Pattern Recognition Workshops, 2017.
4. Keiji Kamei, Yuuki Kakizoe, "An Approach to the Development of a Game Agent based on SOM and Reinforcement Learning", 5th IIAI International Congress on Advanced Applied Informatics, 2016.
5. Ian Good fellow and Yoshua Bengio and Aaron Courville "Deep Learning," An MIT Book Press, 2016.
6. Michael Nielsen "Neural Networks and Deep Learning"

Published By:
Blue Eyes Intelligence Engineering
& Sciences Publication