Proceedings of the 2023 International Conference on Machine Learning and Automation
DOI: 10.54254/2755-2721/43/20230846

A model for automatic agent generation based on machine learning in simulation games

Siyuan Wang

School of Advanced Technology, Xi'an Jiaotong-Liverpool University, Suzhou, 215123, China

1255109893@139.com

Abstract. With the continuous promotion of Artificial Intelligence (AI) technology in games, researchers are generally willing to pay attention to the effect of AI on game picture quality and man-machine games. However, how to generate a large number of non-player characters (NPC) in a game, especially in a simulation game, is still a problem. Even if some people use machine learning to discover solutions, most answers tend to be limited to copying finite-state machines established by programmers. This paper explores techniques, such as machine learning and maze algorithms, for 2D in-game environments to construct a model that can quickly generate NPCs with 'personality' and adapt to the environment when developing a game. In the process of model construction, the temporal difference algorithm and the depth-first algorithm are used to customize the decision structure for the NPC. At the same time, a simple node-style map generation algorithm is added to restrain NPC. Moreover, an optional neural network model gives the NPC a memory that can be updated. This model successfully realizes the rapid construction of NPC in developing some simple 2D games. In addition, NPCs have a fair amount of intelligence. It can be used during game development or while the game is running and generate a certain number of NPCs in an unfamiliar environment with differences in behavior.

Keywords: machine learning, non-player characters, automatic generation.

1. Introduction

Since the first simulation (SIM) games were released in 2000, they have become one of the most realistic and magical games. This type of game puts aside the limitations of focusing on fantasy, competition, action, and other exquisite elements. Instead, return to the original goal of video games, which is to simulate everything about human beings by computers [1]. Although elements such as environment, interaction, etc., are essential in a simulated world, when it comes to player engagement, the most basic but important factor is the computer-simulated human or humanoid.

In recent years, advances in artificial intelligence (AI) have brought some changes to the game industry, and people are interested in exploring the role of AI in video games. Algorithms such as automatic map generation and natural language processing are changing rapidly, and new games use AI on a large scale, such as Spelunky and No Man's Sky [2]. Improvements related to NPCs are also essential. For example, the generation and game of robots in real-time Strategy (RTS) games are the product of the integration of reinforcement learning and supervised learning. These algorithms of self-improving agents in the environment took the agility and intelligence of non-player characters (NPC) to

DOI: 10.54254/2755-2721/43/20230846

a new level [2, 3]. However, NPCs, widely used in ubiquitous SIM games, are generated by people making finite state machines (FSM) for them, and the robots are repeatedly triggered according to certain conditions and then interact with each other on certain routes. This is very rigid and requires a great number of effort to customize the rules for each NPC. We need to use AI to make improvements [4]. Especially reinforcement learning and supervised learning directions. For example, over the past few years, several authors have improved the results and stability of the original Deep Q-Networks. Mnih et al., in 2016, successfully applied neural networks to actor-critic RL [5, 6].

Overall, these improvements aim to make the gameplay experience more realistic and the construction of agents smoother. For games that require many NPCs, such as simulation games, it is important to quickly supply the environment with several NPCs and make them look like real people when the player interacts [7].

This article builds a similar environment based on simulation games and uses artificial intelligence technology to operate NPC. We strive to discard formulating complete finite-state machines. Instead, the blank robot is placed into the environment, and the reward and punishment mechanisms are set to simulate the impact of the natural environment on actual humans. Reinforcement learning allows the robot components to have their behavior logic, and supervised learning ensures the robot's survival. Finally, implement a model where NPC decide their own lives.

In the sessions, we will first state the principle and structure of the constructed model, then introduce the process in the experimental operation, and finally discuss the results and draw a conclusion.

2. Methodology

In general, the core of this paper is the fast generation of NPC. Then, the steps to study the generative model of NPC are divided into the reinforcement learning part for training blank NPCs, the supervised learning part for providing memory to NPCs, and the environment construction and adaptation part for testing results. The following methodology details the NPC model construction and the checking process.

2.1. Setting up the NCP status and the required environment

Before building the algorithm, we need to build some framework for the ungenerated NPCs. One framework gives the NPC some "motivation." That is, we set a state for the NPC, and the NPC's state changes as it moves through the environment and interacts with it. Figure 1 shows an example state.

For the NPC to be trained, one of the purposes of its construction is to fill the environment, so the basis of its activity is the interaction with the environment [8]. So, another framework is to provide an environment for the NPCs to be trained, and the environment can give the NPCs feedback. We have created a simple way to construct a flat environment from input coordinates and the relationships between different coordinates. Where coordinates represent the intersection points of the map. Coordinates and connection relationships form paths. This environment is not the same as the cell environment in the maze. Still, rather a more specific point and line environment and can be read by reinforcement learning algorithms and supervised learning algorithms. The relationship between the environment and the two algorithms is shown in Figure 2.

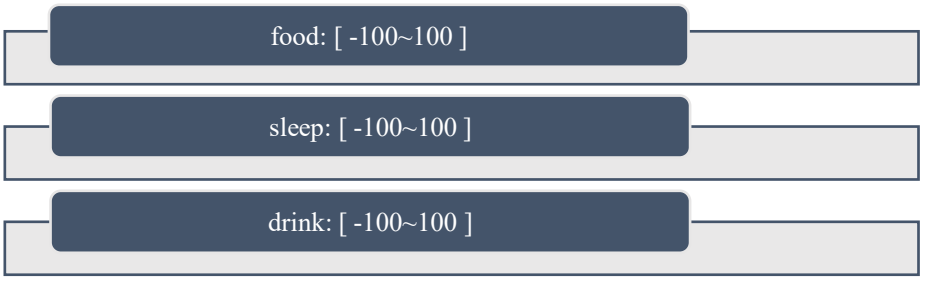


Figure 1. Display of NPC status.

DOI: 10.54254/2755-2721/43/20230846

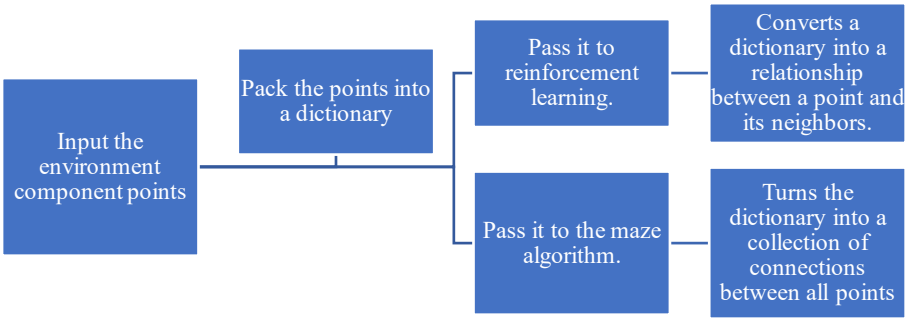


Figure 2. Relationship between the environment and the two algorithms.

2.2. Improvement based on reinforcement learning

After the preparatory work is done, the central part of the model is created. To get NPC out of the limitation of finite state machines, we use reinforcement learning to build NPC's decision model. The basis of the algorithm is the general time-difference algorithm. For this algorithm, in action, agents learn through constant trial and error as they explore the environment, making educated guesses based on feedback from their last action, and create a Q-table to store the value of each possible action at each step [9,10]. However, when we tried to build the model, we found that a simple temporal difference algorithm for maze-like single-width paths, such as paths with coordinates (0,0) to (5,0), required a lot of preparation work to enter the detailed path for the algorithm. Including in what coordinates is the intersection and what coordinates is the wall. So, while using Q-tables to collect environmental feedback and calculate values. We designed a point algorithm to return the direction an NPC can move when located at a certain coordinate without hitting a wall or going off the map. This point algorithm is added to the temporal difference algorithm wherever necessary to determine in which direction the NPC can move. The pseudocode of the temporal difference algorithm and the point algorithm are shown in Table 1 and Figure 3, respectively.

Table 1. Point algorithm.

Point (NPC coordinates) Initial point ←The coordinates of the nodes adjacent to the coordinates of the NPC Loop for each step of an initial point: Calculate the relative distance between the current coordinate of NPC and each point in the initial point. Returns the relative distance between each neighboring point and the current point

DOI: 10.54254/2755-2721/43/20230846

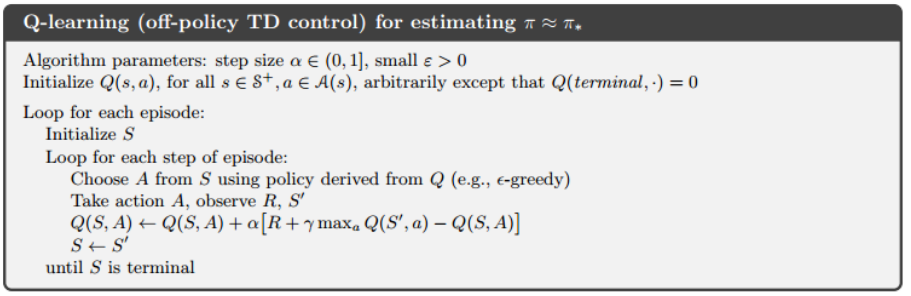


Figure 3. Traditional temporal difference algorithm.

In Figure 3, Q is a three-dimensional matrix. The first and second dimension is the coordinates of the points around the center of a point, and the third dimension is the four possible directions of action at this point: forward, backward, left, and right.

At this point, the NPC can act in the environment at its discretion. We could specify that it stops after a certain number of steps and waits for further instructions, representing the NPC's day.

Since we assume that the NPC thinks of a real person, reinforcement learning can only motivate it to act within the environment. After the NPC completes its' day 'activities, we need it to return to its starting coordinate, home. The model uses the depth-first algorithm of the maze algorithm and modifies it to be suitable for a more general environment. A typical depth-first algorithm traverses its surroundings and assigns a score to each direction, picking the cell with the highest score to move [11]. This algorithm can find the shortest path from the beginning to the end of the maze [12]. We regard the stop position after the reinforcement learning movement as the starting point, and the NPC departure zone as the endpoint. To adapt to a more random environment consisting not of cells but of points and lines, we modified the Return algorithm, as shown in Table 2.

Table 2. Return algorithm.

Return (Initial point, Stop point) Read environment dictionary Represents points in a dictionary as contiguous relationships. Loop for a few times: Depth-first maze model, find a better solution.

2.3. Integrate NPC logic using a neural network model.

The main body of the third part is the neural network model. Such models are applied to supervised learning. It is a loosely computational system inspired by the neural networks that make up animal brains. It belongs to the nonlinear statistical data modeling tool. It is good at modeling the complex relationship between various inputs and outputs for pattern recognition, such as the behavior logic library of NPC that we built [13,14].

The purpose of supervised learning is to train the NPC's behavior logic library and give the NPC 'personality.' After the database training is completed. When an emergency occurs in the same environment, the supervised learning model will score the feedback received by the NPC and the consequences of its actions, updating the database (i.e., memory).

For the data to be trained, the source is that the NPC uses supervised learning to act within the environment for a certain amount of time after we acquire the behavior. Each row of the data consists of the direction the NPC moved for each step of the day. The label for each row is the state in which the NPC lost the least on that day. Since the data we collected is not linear, we chose neural network model training. The data structure is not very large, so there is no need to consider very large levels when

DOI: 10.54254/2755-2721/43/20230846

choosing hidden layers with neurons in each layer. We validated 1 to 20 hidden layers, each with neurons from 5 to 80 in step 5, traversing and trying to fit the data. The result is shown in Figure 4.

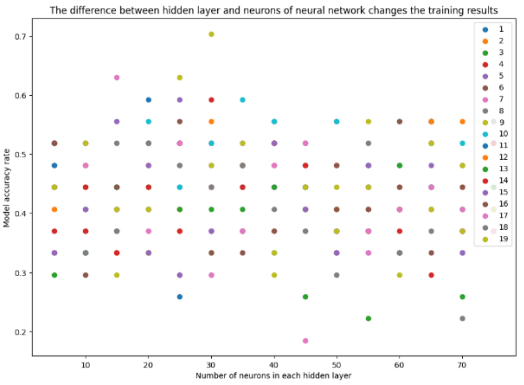


Figure 4. The difference between the hidden layer and neurons of the neural network changes the training results.

In Figure. 4, the horizontal coordinate is the number of neurons in the hidden layer, the points in different colors represent the number of hidden layers, and the vertical coordinate is the model accuracy. Therefore, we can find many hidden layers and neurons for model training. Since using 7 hidden layers, with 30 neurons in each layer, a success rate of about 0.7 can be obtained.

3. Results and Discussion

At this point, we can assemble a complete model based on the methodology. The first is to set the environment and state. Then, the reinforcement learning algorithm trains a certain number of NPCS, stores the behavior logic of NPCS in the database, and trains the neural network model according to this data. Later, we can directly access the NPC's memory when using an NPC. If a sudden external force changes the NPC's behavior, it can self-evaluate the changed behavior according to the neural network model and update the memory. To show the effect of using the model, we set up a test. We tried several different environments, as well as the training speed and visualization results of the NPC in the environment. The result is shown in Figure 5.

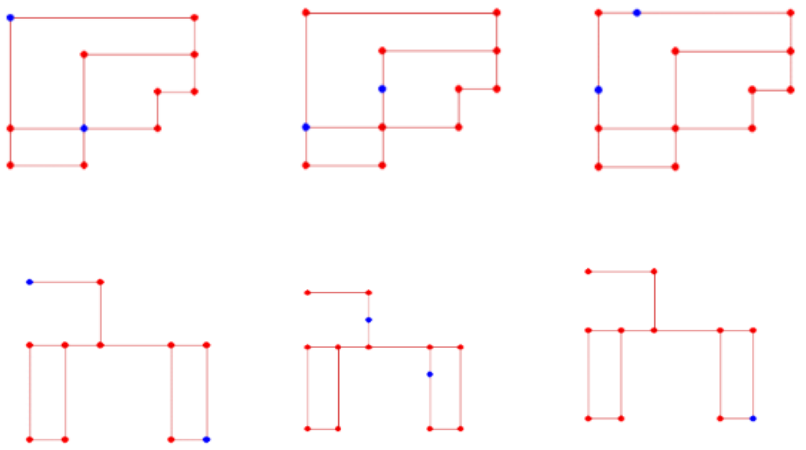
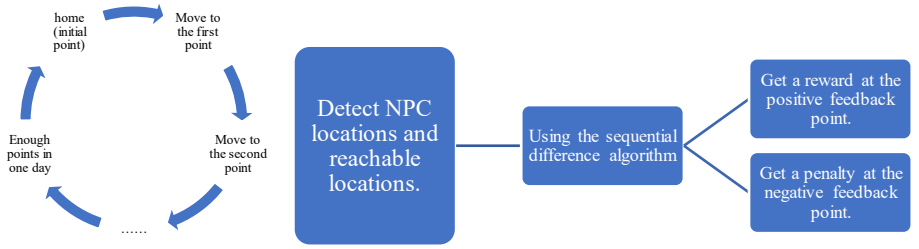


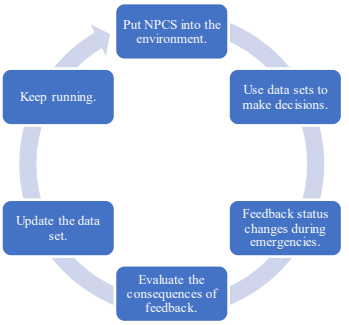
Figure 5. Trial run of multiple NPCS in an unfamiliar environment.

DOI: 10.54254/2755-2721/43/20230846

Parts a and b in Figure 5 are two tests. They generate two NPCS to learn on a map of 11 nodes and try to act. Figure 5 shows the state of the trained NPC at runtime. As shown by the NPC training time calculated by the computer, the two NPC training times in a and b are 53.8279186s and 51.4309817s. In other words, on a home computer, it takes less than a minute to iterate two NPCS with independent behaviors in an environment composed of ten large nodes and countless small nodes. Finally. Show the complete flow of NPC training and use. This is shown in Figure 6. Where 6a is the process of NPC learning in the environment when reinforcement learning is used. 6b is the process of the learned NPC living and dealing with the emergency under the decision of the database fitted by the neural network.



(a). Training in environment



(b). Use the trained NPC.

Figure 6. Flowchart of NPC training and use.

4. Conclusion

This paper mainly studies how to automatically generate intelligent NPC in a certain environment and make the agent more like a real person while ensuring that many NPCS can be generated. After several months of exploration, we built a generative model of NPC using reinforcement learning and supervised learning algorithms. Users can input their nodes to form a two-dimensional environment and then define the initial point to let multiple NPCS learn the environment. The learned NPC will store the logic as a memory. Memory is trained to be thought. And the mind can modify the memory when there is an unexpected situation. After specific tests, the model can successfully generate NPC. It can also have better training efficiency than ordinary home computers. However, our model leaves a lot to be desired. The first is the lack of generality of the model. The NPC generation algorithm can only run in a 2D lattice environment that meets the requirements. The second is its efficiency, leading to better neural network architectures when using supervised learning to fit the data. These are the results and shortcomings of the model we built. There is an excellent potential for improvement, and it is hoped that these defects can be remedied in future research.

References

[1] Annart J. (2020) A brief cultural and industrial history of video games. La Revue Nouvelle.;1(1):56-69. doi:10.3917/rn.201.0056

[2] Risi, S. and Preuss, M. (2020) 'From Chess and Atari to StarCraft and Beyond: How Game AI is Driving the World of AI'. doi:10.1007/s13218-020-00647-w.

DOI: 10.54254/2755-2721/43/20230846

[3] N. A. Barriga, M. Stanescu, F. Besoain and M. Buro, (2019) "Improving RTS Game AI by Supervised Policy Learning, Tactical Search, and Deep Reinforcement Learning," in IEEE Computational Intelligence Magazine, vol. 14, no. 3, pp. 8-18, doi: 10.1109/MCI.2019.2919363.

[4] C. Hu et al., (2023)"Reinforcement Learning With Dual-Observation for General Video Game Playing," in IEEE Transactions on Games, vol. 15, no. 2, pp. 202-216, doi: 10.1109/TG.2022.3164242.

[5] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, et al., (2016) "Asynchronous methods for deep reinforcement learning", International Conference on Machine Learning, pp. 1928-1937.

[6] R. R. Torrado, P. Bontrager, J. Togelius, J. Liu and D. Perez-Liebana, (2018)"Deep Reinforcement Learning for General Video Game AI," 2018 IEEE Conference on Computational Intelligence and Games (CIG), Maastricht, Netherlands, pp. 1-8, doi: 10.1109/CIG.2018.8490422.

[7] P. Xu, Q. Yin, J. Zhang and K. Huang, (2022) "Deep Reinforcement Learning With Part-Aware Exploration Bonus in Video Games," in IEEE Transactions on Games, vol. 14, no. 4, pp. 644-653, doi: 10.1109/TG.2021.3134259.

[8] BUDIMAN, J. S.; LAURENSIA, M.; ARTHAYA, B. M. Maze Mapping Based Modified Depth First Search Algorithm Simulator for Agricultural Environment. 2021 International Confere nce on Mechatronics, Robotics and Systems Engineering (MoRSE), Mechatronics, Robotics and Systems Engineering (MoRSE), 2021 International Conference on, [s. l.], p. 1–6, 2021. DOI 10.1109/MoRSE50327.2021.9766000. Disponível em: https://search-ebscohost-com-s.elink.xjtlu.edu.cn:443/login.aspx?direct=true&db=edseee&AN=edseee.9766000&site=eds-live&scope=site. Acesso em: 8 jul. 2023.

[9] R. Sutton and A. Barto, (2018) Reinforcement Learning: An Introduction ser. Adaptive Computation and Machine Learning series, MIT Press.

[10] Li, H. et al. (2023) 'An Improved Artificial Bee Colony Algorithm With Q-Learning for Solving Permutation Flow-Shop Scheduling Problems', IEEE Transactions on Systems, Man, and Cybernetics: Systems, Systems, Man, and Cybernetics: Systems, IEEE Transactions on, IEEE Trans. Syst. Man Cybern, Syst, 53(5), pp. 2684–2693. doi:10.1109/TSMC.2022.3219380.

[11] Mahmud, M.S. et al. (2012) 'A Greedy Approach in Path Selection for DFS Based Maze-map Discovery Algorithm for an autonomous robot', 2012 15th International Conference on Computer and Information Technology (ICCIT), Computer and Information Technology (ICCIT), 2012 15th International Conference on, pp. 546–550. doi:10.1109/ICCITechn.2012.6509798.

[12] Uludağlı, M.Ç. and Oğuz, K. (2023) 'Non-player character decision-making in computer games,' Artificial Intelligence Review: An International Science and Engineering Journal, pp. 1–33. doi:10.1007/s10462-023-10491-7.

[13] Chen Y-Y, Lin Y-H, Kung C-C, Chung M-H, Yen I-H. (2019) Design and implementation of cloud analytics-assisted smart power meters considering advanced artificial intelligence as edge analytics in demand-side management for smart homes. Sensors; 19; 9: 2047. 2019Senso.19.2047C. 10.3390/s19092047. 31052502. 6539684

[14] Hamadani, A., Ganai, N.A. and Bashir, J. (2023) 'Artificial neural networks for data mining in animal sciences,' Bulletin of the National Research Centre, 47(1), pp. 1–8. doi:10.1186/s42269-023-01042-9.