Automating agent decisions in a virtual environment

Uicoabă Alexandru
Facultatea de Electronică, Telecomunicații și Tehnologii Informaționale
Universitatea Politehnica Timișoara
Timișoara, România
alexandru.uicoaba@student.upt.ro

***Abstract*—This paper proposes an analysis of how smart agents are implemented in virtual environments. Video games are meant to introduce the user to a virtual world. The diversity of these virtual worlds is huge, whether we refer to board games, action, racing, or strategy games, with the ability to play in single-player or multi-player mode. Following this analysis, an experiment was performed in which a kart agent learns to drive in a virtual environment, using several machine learning algorithms. Following the results, it is observed that the agent's performance is given both by the chosen machine learning algorithm and by the virtual environment.**

Keywords—artificial intelligence, smart agents, video games, machine learning, neural networks, reinforcement learning

I. Introduction

Video games have a long history, from the first games developed on the **Atari** console to the well-known **Pac-Man**, **Super Mario,** and **Sonic**, to contemporary ones, such as **Cyberpunk 2077.**

Intelligent agents are computer systems that are placed in various virtual environments, in this case, virtual worlds, which are capable of autonomous actions to achieve well-defined objectives [1]. They aim to solve problems in the game, which could be defeating an opponent in battle or navigating a maze [2].

Virtual worlds within video games are a playground for a wide variety of algorithms to try new things. Depending on the results obtained, these ideas can be transferred and applied in real life. The goal of artificial intelligence is to create intelligent agents who can make the best decisions, both in virtual environments and in the real world [3].

To train these agents, existing games can be used, or games can be created from scratch, in a multitude of programming languages, to create the environment we need.

In the last decade, more and more work has been done in this direction, due to the technological advancement we are in, being advanced hardware components for the use of developed algorithms that need high computing power. Thus, there have been games in which the graphics are getting closer to reality.

So, through smart agents, the emphasis is on getting a more natural behaviour of the characters in the games, by using fuzzy logic and neural networks. In this way, we get characters that behave naturally and fit into the game's setting more pleasantly [4].

Machine learning algorithms

Machine learning is the field of study that provides the ability of computers to learn without being explicitly programmed. Below is a breakdown of the agents and whether they are trained as a result of human intervention [5].

Unsupervised machine learning methods are particularly useful in descriptive tasks, as they aim to find relationships in a data structure without having a measured result. This category of machine learning is called unsupervised because it lacks a response variable that can supervise the analysis [6].

Supervised machine learning methods are used to describe predictive tasks because they are intended to predict and classify a particular outcome of interest (if a particular person is prone to certain diseases based on medical information). Supervised learning has been applied to large data structures because in order to achieve good accuracy we need to "feed" the ML model with a large set of data in the learning cycle [5].

In this ML category, the data is divided into several sets. The first set is the training set that is used to build the model. The second set of data is the test set, which determines the performance of the model by calculating several indicators, such as accuracy [5].

In **Reinforcement Learning**, a software agent makes observations and takes actions within the environment, and in return it receives rewards. Its objective is to learn to act in a way that will maximize its expected rewards over time [6].

It focuses on the episodic setting of reinforcement learning, where the agent's experience is broken down into a series of episodes. In each episode, the agent's initial state is randomly sampled from a distribution, and the interaction proceeds until the environment reaches a terminal state. The goal of episodic reinforcement learning is to maximize the expectation of total reward per episode and to achieve a high level of performance in as few episodes as possible [7].

Related work

To choose machine learning algorithms, several scientific articles were analyzed. For the most part, the chosen algorithm was learning by reinforcement, either simple[8][9] or in combination with other algorithms, such as the Markov Decision Making [10]. Another Reinforcement Learning algorithm is that of deep reinforcement learning, also used in several articles [11][12][13].

Zpepei Wei and his team have obtained an autonomous agent capable of playing the well-known game Snake. The training of the agent was done by using the Convolutional Neural Network (CNN) with a variety of Q-learning. This option aims to avoid situations where rewards are rare and come late. Experiments have shown that through this model, the agent has even managed to exceed the level reached by people [14].

Following the analysis made on several works in which agents were trained using various machine learning algorithms, the following questions were reached:

Question 1 - What types of algorithms are more efficient?

Question 2 - What data does the algorithm need to learn?

Question 3 - Is a trained machine learning agent better than a real person?

II. Methodology

This section will present the implementation of the experiment that aims to train an agent in a virtual environment, both by using ML unsupervised algorithms and by using reinforcement learning.

The agent must learn how to drive in a virtual circuit, similar to the NASCAR one (Fig. 1). His goal is to go through the three checkpoints placed on the circuit in the shortest possible time. The distribution of the three checkpoints (with purple) can be seen below.

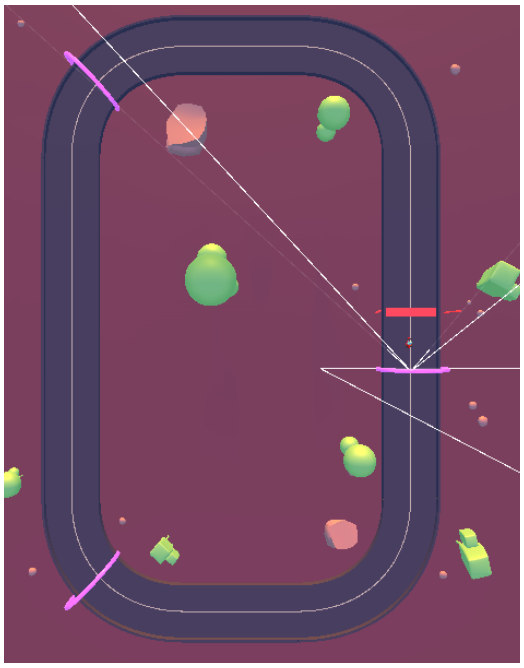
Fig. 1. Track Layout

Collecting data from the virtual environment

The first step is to collect the necessary data of the agent, in this case, the kart, which is necessary for training. As a medium, the microgame kart from Unity was used [14], which provides the circuit together with the kart. To be able to train the ML models, the following data is collected through a script: the positions on the three axes, the distance to the object from the 5 sensors, the direction of movement of the kart, the state of the game, and the time in the game.

Access to this information is done by calling a get endpoint of an API made with the Flask API. At each frame of the game, the kart information is updated by a POST call from the unity to the water environment. Another way to access data is through the *csv* files that are generated for each game.

The data flow, both for the scenario in which the kart learning is performed is done through supervised machine learning methods, and through reinforcement learning are presented in diagrams 2 (Fig. 2) and 3 (Fig. 3).

III. Implementation

Unsupervised ML

A first experiment that aims to train the agent to drive in the virtual environment is through supervised algorithms. In this sense, two algorithms will be used, namely Random Forrest and Decision Tree.

Since we are talking about supervised machine learning algorithms, they need a set of data to train on. So, a human player simulated driving the kart for several episodes, the kart data was saved in CSV files, one file for each episode.

After combining the CSV files into one, the data cleaning operations were performed. The next step was to choose the features for the ML algorithms. From the point of view of dividing the data between training data and validation data, a ratio of 7 to 3 was used.

After the machine learning model has been trained, each frame of the game transmits the data from the environment through the POST endpoint to the script in Python, the data is taken by the ML model, whether we are talking about Random Forrest or Decision Tree. That decision is sent back to the unity environment as a response to the same endpoint, after which a mapping is done to the values in the Unity for kart control.

The above steps are repeated until the game reaches a finished state, such as when the kart reaches the finish line, or when the time expires.

Its output, which represents the decision that the kart will take in the next step, is an integer value from one to fifteen, representing all possible combinations for the four commands of the kart (forward, backward, left, and right).

Reinforcement Learning

The second experiment involves learning the agent to drive (on the same circuit used for the first experiment) using a reinforcement learning algorithm. The Reinforcement learning part will be done with the help of the OpenAI gym toolkit. This type of algorithm is based on making decisions and observing their effect.

At the time *t*, a certain decision is made. At time *t + 1*, it is checked whether the decision was a good one or not. Depending on this, the agent receives a reward or a penalty. The goal is for the agent to receive as many rewards as possible so that they can perform as well as possible in that environment.

In this scenario, in addition to the endpoint for communication between the python script and the Unity environment, two additional endpoints are used. One is used by the reinforcement learning environment to retrieve the latest data from the game, and the other is to send actions provided by the model to the link script, which are then sent to the game.

OpenAi Gym was used to create the learning environment using Reinforcement Learning. It has several game scenarios that can be used, or as in this case, can be created from scratch. To achieve the environment, it is necessary to define several specialized functions.

The first is the __*init*__, which has the initialization part of the game. Another important function is the __*reset*__ function, which is called when a learning sequence is completed. This can be done either if the kart reaches the finish line or if the time expires.

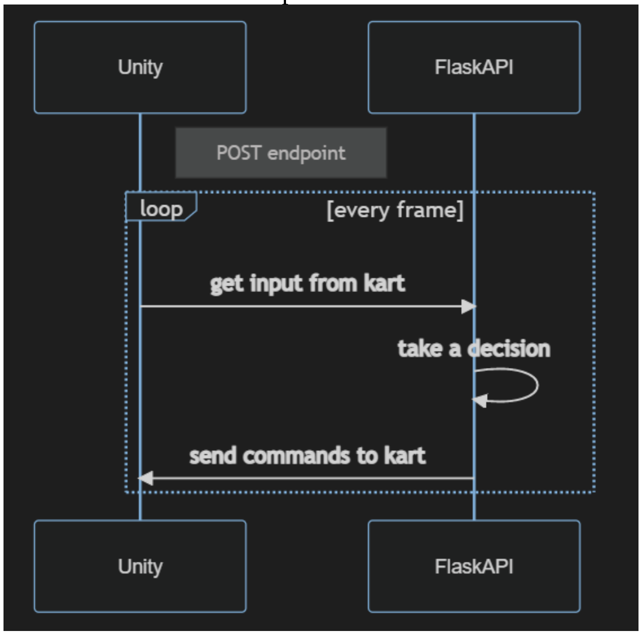

Fig. 2. Supervised ML data flow

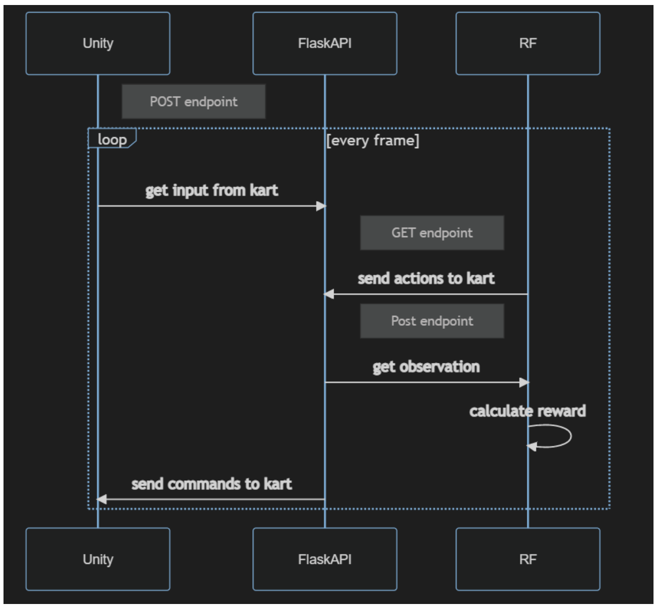

Fig. 3. Reinforcement Learning data flow

In the __*step*__ function, the decision is made and the reward is calculated. Here, too, the information from the kart that is taken into account in the next step is received and the actions taken by it are sent to unity.

In this case, several factors are taken into account in calculating the reward. The first factor is the direction of travel of the kart. if he moves in front, according to the direction of the circuit tour, he will receive a reward (+100), otherwise, he will be penalized (-5000). Another criterion depends on the value of the 5 obstacle detection sensors. The input data from the unity environment is updated every frame of the game. If the kart approaches an object (currently on the parapet of the circuit), it will receive a penalty that increases as it approaches the obstacle in an almost exponential manner(from -50 to -5000 for each sensor).

IV. RESULTS

After conducting the two experiments and collecting the related data, in case the route is known, the following results were obtained.

In Table 1 it can be seen the times obtained for collecting the 3 checkpoints in the various implementations). In this comparison, the time obtained by a real person, karting through the Random Forrest and Decision Tree algorithm after five repetitions. The accuracy obtained after testing the algorithms used was about 83 percent was obtained compared to the decision taken by the human in both cases. As can be seen in Fig. 1, the differences between the three scenarios do not exist. This is also because the circuit is quite simple, with an oval configuration, similar to those in the NASCAR.

For the reinforcement learning part, the time taken is not close to that of the other algorithms, because the kart learns from scratch how to behave, compared to other algorithms in which the kart makes a decision similar to that of the person in a similar situation. After running the reinforcement learning for 9 hours, approximately 540 episodes were done. The agent managed to complete the goal in a couple of them.

Discussions

This experiment shows that if the route in which the kart is to be learned is known, the use of prediction algorithms is more optimal. On the other hand, if the route is not known, or if it is changed or generated randomly, it is preferable to use reinforcement learning algorithms.

In the future, the complexity of the experiment can be continued in several directions. One direction would be to switch from a single agent to a multi agent mode and observe how they interact with each other. Another direction would be to use several circuit tours, and to observe the differences in time and route between them. One last direction could be to increase the complexity of the circuit and change it in a random way, and to observe how the agent adapts to the change.

Table 1. Human vs Decision Tree vs Random Forrest

	Human	Random Forrest	Decision Tree
Lap 1	27 s	29s	29s
Lap 2	27s	28s	28s
Lap 3	28s	27s	28s
Lap 4	26s	27s	29s
Lap 5	26s	28s	28s

Limitation

A first limitation that emerges from the first experiment, in which supervised machine learning techniques were used, is that the prediction will be a good one only if the circuit is a known one, and previously a player drove in an optimal way. If the circuit is randomly generated, this type of approach can no longer be used.

The second experiment shows that the agent needs a lot of time and a lot of episodes to perform better and better.

V. Conclusion

In this paper, we want to automate the learning process of an agent in a virtual environment. For this, two experiments were performed, and the results showed that for this case the most recommended is the supervised machine learning approach.

For the experiment that uses reinforcement learning, with the time limit in which the agent must learn at this time, the number of episodes will be further increased to see if the training time has a role in the performance obtained by the agent.

Reference

[1] L. Padgham and M. Winikoff, *Developing Intelligent Agent Systems: A Practical Guide*, 1st ed. Wiley, 2004. doi: 10.1002/0470861223.

[2] F. Safadi, R. Fonteneau, and D. Ernst, ‘Artificial intelligence in video games: Towards a unified framework’, *Int. J. Comput. Games Technol.*, vol. 2015, 2015.

[3] D. A. Chentsov and S. A. Belyaev, ‘Monte Carlo Tree Search Modification for Computer Games’, in *2020 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus)*, Jan. 2020, pp. 252–255. doi: 10.1109/EIConRus49466.2020.9039281.

[4] A. Webster, ‘Designing League of Legends’ stunning holographic Worlds opening ceremony’, *The Verge*, Nov. 11, 2019. https://www.theverge.com/2019/11/11/20959206/league-of-legends-worlds-2019-opening-ceremony-holograms-holonet (accessed Nov. 14, 2020).

[5] A. Géron, *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*, 2nd Edition. O’Reilly Media, Inc., 2019.

[6] T. Jiang, J. L. Gradus, and A. J. Rosellini, ‘Supervised Machine Learning: A Brief Primer’, *Behav. Ther.*, vol. 51, no. 5, pp. 675–687, Sep. 2020, doi: 10.1016/j.beth.2020.05.002.

[7] G. Brockman *et al.*, ‘OpenAI Gym’. arXiv, Jun. 05, 2016. Accessed: Jun. 03, 2022. [Online]. Available: http://arxiv.org/abs/1606.01540

[8] Z. Zhang, D. Wang, D. Zhao, Q. Han, and T. Song, ‘A Gradient-Based Reinforcement Learning Algorithm for Multiple Cooperative Agents’, *IEEE Access*, vol. 6, pp. 70223–70235, 2018, doi: 10.1109/ACCESS.2018.2878853.

[9] I. Fathy, M. Aref, O. Enayet, and A. Al-Ogail, ‘Intelligent online case-based planning agent model for real-time strategy games’, in *2010 10th International Conference on Intelligent Systems Design and Applications*, Nov. 2010, pp. 445–450. doi: 10.1109/ISDA.2010.5687225.

[10] D. Daylamani-Zad, L. B. Graham, and I. T. Paraskevopoulos, ‘Swarm intelligence for autonomous cooperative agents in battles for real-time strategy games’, in *2017 9th International Conference on Virtual Worlds and Games for Serious Applications (VS-Games)*, Sep. 2017, pp. 39–46. doi: 10.1109/VS-GAMES.2017.8055809.

[11] D. Daylamani-Zad and M. C. Angelides, ‘Altruism and Selfishness in Believable Game Agents: Deep Reinforcement Learning in Modified Dictator Games’, *IEEE Trans. Games*, pp. 1–1, 2020, doi: 10.1109/TG.2020.2989636.

[12] M. P. P. Faria, R. M. S. Julia, and L. B. P. Tomaz, ‘Evaluating the Performance of the Deep Active Imitation Learning Algorithm in the Dynamic Environment of FIFA Player Agents’, in *2019 18th IEEE International Conference On Machine Learning And Applications (ICMLA)*, Dec. 2019, pp. 228–233. doi: 10.1109/ICMLA.2019.00043.

[13] Y. Takano, S. Ito, T. Harada, and R. Thawonmas, ‘Utilizing Multiple Agents for Decision Making in a Fighting Game’, in *2018 IEEE 7th Global Conference on Consumer Electronics (GCCE)*, Oct. 2018, pp. 594–595. doi: 10.1109/GCCE.2018.8574675.

[14] Z. Wei, D. Wang, M. Zhang, A. Tan, C. Miao, and Y. Zhou, ‘Autonomous Agents in Snake Game via Deep Reinforcement Learning’, in *2018 IEEE International Conference on Agents (ICA)*, Jul. 2018, pp. 20–25. doi: 10.1109/AGENTS.2018.8460004.

[15] N. A. Mas’udi, E. M. A. Jonemaro, M. A. Akbar, and T. Afirianto, ‘Development of Non-Player Character for 3D Kart Racing Game Using Decision Tree’, *Fountain Inform. J.*, vol. 6, no. 2, pp. 51–60, 2021.