STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **LXVII**, Number 1, 2022
DOI: 10.24193/subbi.2022.1.02

MULTIPLE TYPES OF AI AND THEIR PERFORMANCE IN VIDEO GAMES

IULIAN PRĂJESCU AND ALINA DELIA CĂLIN

Abstract. In this article, we present a comparative study of Artificial Intelligence training methods, in the context of a racing video game. The algorithms Proximal Policy Policy Optimization (PPO), Generative Adversarial Imitation Learning (GAIL) and Behavioral Cloning (BC), present in the Machine Learning Agents (ML-Agents) toolkit have been used in several scenarios. We measured their learning capability and performance in terms of speed, correct level traversal, number of training steps required and we explored ways to improve their performance. These algorithms prove to be suitable for racing games and the toolkit is highly accessible within the ML-Agents toolkit.

1. Introduction

From their inception in the 1950s, video game started to evolve and to become more and more complex in terms of better graphics, interaction controllers and game mechanics, audio and visual feedback, progressing at the same pace with the technology of the time and sometimes even pushing technology forward, becoming the beautiful pieces of art we think about today. The industry has been dominated by a small number of companies that established specific practices around the development and distribution of video games. Strategy video games have found an important role due to their effect on improving hand–eye coordination and visual-motor skills [14].

One of the most important steps in this evolution is marked by adding Artificial Intelligence (AI) methods, which simulates the presence of other players or characters, increasing the immersive experience of the game. This aims at designing agents capable of playing video games without human intervention [12], often called non-player characters. Thus, the efficiency of an AI agent in a game is generally evaluated by human experience [9]. The importance of

Received by the editors: 23 September 2021.

2010 *Mathematics Subject Classification.* 91A10, 68T05.

1998 *CR Categories and Descriptors.* I.2.1 [**Artificial intelligence**]: Applications and Expert Systems – *Games*; K.8.0 [**Personal computing**]: General – *Gaming*.

Key words and phrases. racing game, PPO, GAIL, behavioral cloning, AI in games.

AI tools in games is not limited to the game experience, but provides a rich research ground for studying and experimenting how humans interact with AI agents [21].

However, there is a gap between academic and industrial approach of game AI that needs addressing. The basic AI algorithms usually used in games (such as ad hoc authoring, tree search, evolutionary computation, and machine learning) do not rise to the current demands, meaning that new methods and techniques are needed [4].

In this paper we aim to encourage a more sophisticated use of AI in industry (such as neural networks) [20], by presenting the new tools available (like the Unity ML-Agents Toolkit, Pytorch) and specific case scenarios where algorithms can be used successfully. Thus, this paper focuses on determining the best method to train AI agents for a specific type of game: car racing simulation games, by presenting a specific video game context. The importance of this type of game is not only recreational, but given the realistic environment, it is used to develop driving skills in a safe environment. Using intelligent methods to simulate the required challenges of the environment and drive progress by competition with non-human agents is the key to success. Three types of AI agents are compared in simulating car driving agents, using the game development platform Unity [8] and the machine-learning agents module based on the PyTorch technology [11]: Proximal Policy Optimization (PPO), Behavioral cloning (BC) and Generative Adversarial Imitation Learning (GAIL) algorithms [18].

2. Background

Some examples of related work in the field would include the idea of a unified video game AI middleware [15], which was created by The International Game Developers Association (IGDA) by launching an Artificial Intelligence Interface Standards Committee (AIISC) in 2002, which had the goal of creating a standard AI interface for reusing and outsourcing AI code [15]. In Berndt et al. [1], was proposed an Open AI Standard Interface Specification (OASIS), which aimed at making the integration of AI in video games easier. This kind of game AI middleware can now be found in multiple video game engines [15], such as CryEngine, Havok, Unreal Engine and Unity, these game engines aiming to provide realistic agents and virtual environments.

In relation to racing games, recent interest has been present in the literature with the most focus on algorithms such as PPO for vehicles in mixed and full-autonomy traffic [13, 17], GAIL for modelling a human driver [2, 10], or BC for robust autonomous vehicles with end-to-end imitation learning [16].

2.1. **The Unity ML-Agents Toolkit.** This Unity toolkit is an open source project that consists of two elements: the ML-Agents software development kit (for creating environments within the Unity Editor and with the associated C# scripts) and a Python package (to help interfacing with the environments created). ML-Agents presents three components: (1) the Agent, responsible with collecting observations and taking actions; (2) the Brain, responsible with making decisions for the linked Agents containing matching observation and action space configurations; (3) the Academy, responsible for managing the learning environment by keeping track of the steps performed by the Agents, setting the target simulation speed and frame rate, and resetting parameters for eventual configuration changes during run-time.

The Python Unity ML-Agents Trainers Package provided in this toolkit communicates with Unity by using the included UnityEnvironment class, by the use of a gRPC communication protocol, which utilises protobuf messages.

2.1.1. *Proximal Policy Optimization (PPO).* By trying to improve the already ample scene of reinforcement learning with neural network function approximators, the OpenAI team introduces a new family of policy gradient methods with the Proximal Policy Optimization Algorithms [18]. These new methods share some of the benefits brought by the trust region policy optimization (TRPO), but have the advantage of having better sample complexity (empirically). While TRPO uses a complex second-order method when confronted with the problem of trying to improve the step on a policy using the data it currently has without stepping too far as to cause a performance collapse, PPO uses a family of first-order methods which use some other algorithmic approaches to keep the new policies close to old.

The main deviations of PPO are the PPO-Penalty and the PPO-Clip. We will primarily focus on the PPO-Clip variant as it is the most commonly used and it is present in the ML-Agents toolkit used in this study. As opposed to PPO-Penalty, it does not have any constraint or a KL-divergence term in the objective, but instead relies on specialised clipping in the objective function to remove incentives for the new policy to get far from the old policy. The PPO algorithm uses fixed-length trajectory segments, where on each iteration, every of the N actors collect T timesteps of data in parallel, then constructs the surrogate loss on the NT timesteps of data and optimises them with minibatch Stochastic gradient descent (SGD) for a K number of epochs.

2.1.2. *Generative Adversarial Imitation Learning (GAIL).* Generative Adversarial Imitation Learning (GAIL) is an Inversive Reinforcement Learning algorithm, which, as the name suggests, uses a Generative Adversarial Network (GAN) to function. This algorithm can be also described as a model-free

imitation learning algorithm, and can yield a good performance for complex behaviours, particularly in big, high-dimensional environments. As presented in [6], GAN is a type of generative model, which brings a way to learn deep, hierarchical representations in a semi-supervised or unsupervised manner.

The GAN architecture consist of two different networks working together to learn from existing datasets. The first network is the generator, which has the role of generating new data by learning the distribution of the input dataset. The second network called the discriminator has the role of gathering the samples from the training data and classifying them either as generated by the generator or as real data. The Inversive reinforcement learning (IRL) methods were presented in the idea of helping the reinforcement learning agents to learn the experts policy and to get reward functions in order to explain the experts behaviors from their given trajectory [7].

2.1.3. *Behavioral Cloning (BC).* Behavioral Cloning (BC) represents a form of "Imitation Learning" which has the goal of creating a model of a human's behavior when trying to execute a difficult set of actions. The BC method is one of the most used approaches in regards to the imitation learning problem and has been proven powerful in the sense that it can very quickly imitate the demonstrator without needing to interact with the environment [19].

This method has been used in many different applications, from flying down a quadrotor on a forest trail [5], to autonomous driving [3]. BC is related with other methods of learning by imitation [31], such as GAIL [7], IRL and other methods that use data from human performance. The behavioral cloning algorithm used by the ML-Agents toolkit is one of Behavioral Cloning from Observation [19] and works in the following fashion: the algorithm needs to find a good imitation policy from a set of state-only demonstration trajectories. The extraction of the agent-specific part of the demonstrated state sequence and the forming of a set of demonstrated agent-specific state transitions, in order for the use of the agent-specific inverse dynamic model [19]. For each transition the algorithm computes the model-predicted distribution over demonstrator actions and uses the maximum-likelihood action as the inferred action. We then build the set of complete state-action pairs [19].

3. Case study

3.1. **The game.** The racing game environment we study is built in Unity, employing several levels (tracks), race configurations, and car models (see Figure 1). The player can compete against multiple AI cars trained and compared in this case study. The agents used different training methods such as PPO, GAIL, BC, and Soft Actor-Critic (SAC). After multiple training sessions, the

SAC models did not manage to train to the point of completing the level so the other methods were used forward.

FIGURE 1. Game circuit

3.1.1. *Training the AI.* After the training process, five models using three different training methods were obtained, one type using PPO, two types using GAIL and two types using BC. The training of the AIs was done using ML-Agents. In this framework the necessary components for training an agents include a virtual environment, the agent component present in the Unity project and a configuration file which holds all the variables and parameters of the neural network and training method used.

The specific settings for each training methods are as follows: for the PPO method the `trainer_type: ppo` and the `reward_signals` need to be `extrinsic`; for the GAIL methods one more module needs to be added, which is the `gail:` one with multiple specific parameters such as `demo_path` for showing the location of the demo file used in the training process, `strength` value representing how much the agent should copy the demonstration, `gamma`, `learning_rate`, `use_actions` and `use_vail`; finally for the BC method the `behavioral_cloning` module which adds the `demo_path` parameter and the `value` representing how much the agents should copy the demonstration and other BC specific parameters.

3.2. **Experiments.**

3.2.1. *Training PPO.* The parameters of the configuration file were adjusted to train multiple agents using trial and error in order to increase the performance of the agents. The best configuration identified (Test 33), which consisted of 40 agents, used a `batch_size` of 120, the `learning_rate` of 0.0003

and `strength` of the `extrinsic` `reward_signals` of 1(the maximum recommended value). This configuration demonstrated very good performance in accurately parsing the circuit, with the maximum speed reaching the value of 27. The agents started training with the environment reward at -1.477 and after training for over 5 million steps they reached the value of 0.7687, being the highest value achieved with this method, and it had a growth value of 2.2457. The evolution of the training is shown in Figure 2.

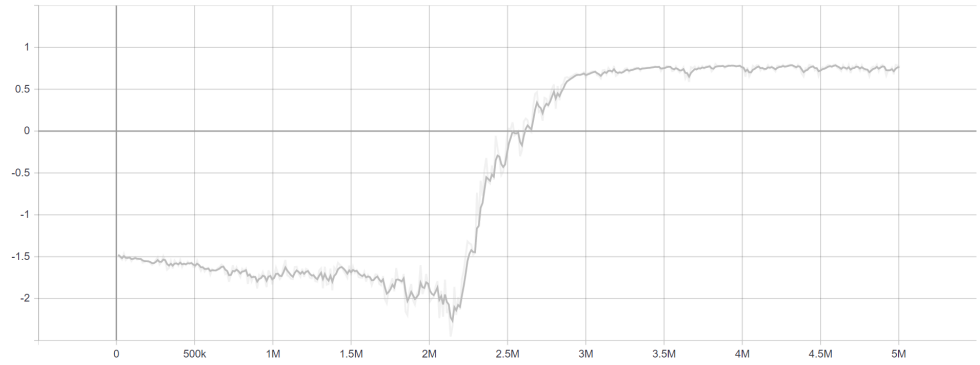

FIGURE 2. Environment cumulative reward of the PPO method

In terms of traversing the level, the final model takes shortcuts by cutting corners and getting off the track portion of the level in order to complete the level as fast as possible. This is a good thing in the context of finishing first, but ultimately decreases the value of the model for not traversing the level correctly.

3.2.2. *Training GAIL.* For the GAIL training method two types of demos were used, one made by a human player, and one made using the PPO trained method, where one demonstration of traversing the track and one agent using the PPO Test 33 brain were recorded. This was done in order to determine if there is a difference in performance between these two kinds of demos.

For both models trained using the GAIL method, along with the 40 agents used, the extrinsic `reward_signals` module was utilised, with the same values as the PPO method, in collaboration with the `gail` module, which included the `learning_rate` of 0.0003, the `encoding_size` of 128 and the gail `strength` of 0.1. After adding the `gail` module, the agents started to learn and the cumulative reward started increasing alongside the performance on the racing track.

The performance of the GAIL method using a player demo reached the speed value of 21, being a very good one for the gaming context, achieved after a bit over 5.5 million steps, and after starting with the environmental cumulative reward of -1.486, it reached the value of 0.725, having a growth value of 2.211. The evolution of this method can be seen in Figure 3.

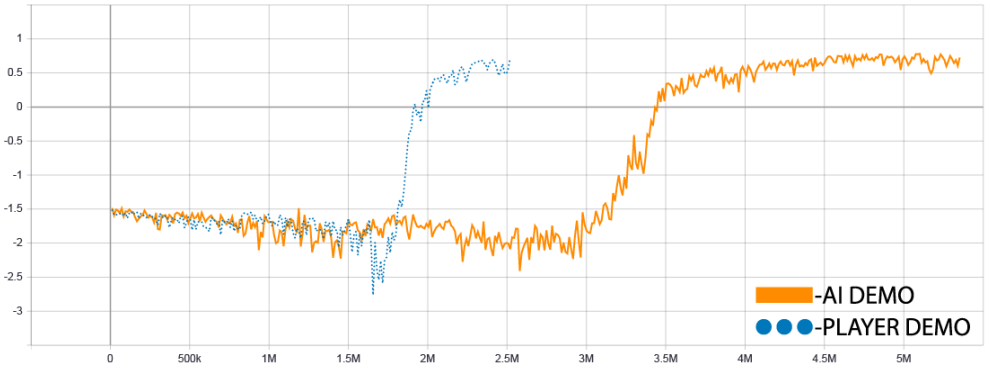

FIGURE 3. Environment cumulative reward of the GAIL method using a player demo versus using an AI demo

The performance of the GAIL method using the AI demo reached the speed value of 17 after just over 2.5 million steps. Starting with the cumulative reward of -1.486 and reaching the value of 0.679, it had an approximated growth value of 2.165, very close to the player made demo method described above and also the PPO model, used for creating the demo after which this model learned. We will examine the evolution of the training session present in Figure 3.

When comparing the two GAIL methods we can see that they have similar results but also big differences. Starting with the similarities, they both have very close growth values and the learning process is very similar, both with the cumulative reward slowly decreasing until the second half of the session where they started to reach their maximum value very fast, then very slightly increasing until the end of the training session. Considering this, the second method, using the AI made demo, learned twice as fast as the first one, but ultimately had a smaller speed performance.

With all this said, we can see that using the player made demo was better than the AI made one, even after considering the inefficiency in time.

3.2.3. *Training BC.* Just like the GAIL agents, the BC method was used to create two types of AI using the same demos as before to determine if the type of demo affects the performance of the AIs and what differences can be

found. The best configuration found so far for this method used 40 agents and the same values for the `extrinsic` `reward_signal` as the PPO and GAIL methods, with the exception of the `batch_size` which was increased to 512 and the behavioral cloning module was added which included a `strength` of 0.1. The best session of the BC method using a player demo was the fourth one with the maximum speed of 6. This result was achieved after 6 million steps, starting with the environment cumulative reward of -5.534 and reaching the value of -4.053 by the end of the training session. The growth value of this method was 1.481 and we can see the evolution of the agent in Figure 4.

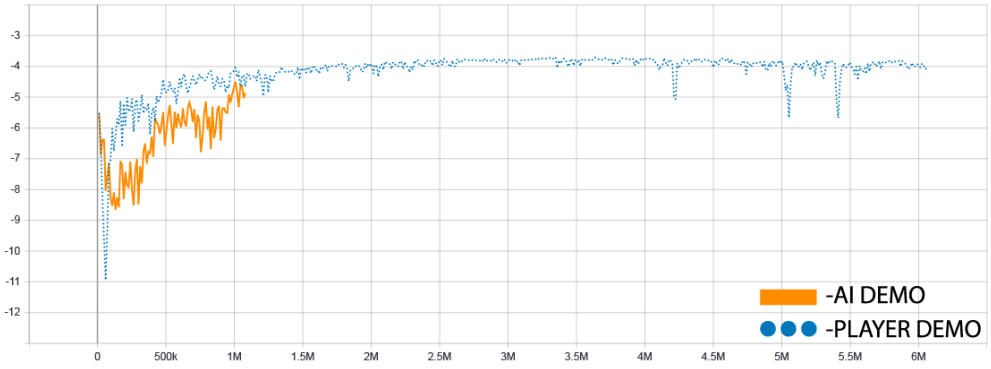

FIGURE 4. Environment cumulative reward of the BC method using a player demo versus using an AI demo

Unfortunately, when traversing the environment, this model also takes shortcuts, going off the track part and onto the surrounding environment, losing value.

As the aforementioned GAIL agent that used an AI made demo, this one also uses the same demo made from the performance of the PPO trained AI. Just like the agents trained with a player demo, the AI demo trained agents achieved a speed performance value of 6 but in this case, the training session was much shorter, ending after just over 1 million steps. Within this period, the agents grew the environment cumulative reward from -5.531 to the value of -4.863, having the final growth value of 0.668.This method had the poorest growth while training and we can visualise it in Figure 4.

Even though both models using the BC training method had the same maximum speed value, the difference between these two methods is the one of training session length and efficiency, the agents using the AI made demo reached the same performance almost 6 times faster than the one trained with the player made demo even though the latter had a bigger growth value.

Overall, we can say that using an AI made demo is better than using a human player demo for the BC training method.

3.3. **Results.** When comparing all five models trained for this experiment, we can see exactly how different the training methods perform and which one has the best performance.

The methods with the fastest growth of their reward value are the BC methods (Player demo red, AI demo dark blue) reaching values close to maximum in just 1 million steps, after that, the GAIL method using an AI made demo (green) is in third place, followed by the PPO method (orange) and finally the GAIL method using a player made demo (light blue).

For the criteria of correctness while traversing the level, all five models have a bad performance, taking shortcuts and cutting corners through the level by going off the track onto the surrounding environment (leading to incorrect level traversal). This fact will not be taken into consideration in the current comparison.

All results are compared in Table 1 below, in terms of speed of circuit traversal, growth value (based on starting and ending reward) and number of steps involved in the training.

Table 1. Comparison of all five initial AIs specifications

	Speed	**Starting reward**	**Ending reward**	**Growth value**	**No. of steps**
PPO	27	-1.477	0.7687	2.2457	5 mil
GAIL& player demo	21	-1.486	0.725	2.211	5.5 mil
GAIL & AI demo	17	-1.486	0.679	2.165	2.5 mil
BC & player demo	6	-5.534	-4.053	1.481	6 mil
BC & AI demo	6	-5.531	-4.863	0.668	1 mil

This table shows us all the properties of each model in the order of which they were trained. Coincidentally, the order also represents the performance order of the models. The PPO model had the best performance of all the trained AIs with the biggest speed and growth values. The next best performance is of the GAIL method with both models having a good performance and as stated in subchapter 3.2.2, the model using a player made demo had a better performance than the one using the AI made demo in both speed and growth value.

The method with the least performance is the BC one, with both models reaching the low speed value of 6 and having suboptimal growth values compared to the other two methods. Even though the BC model using the AI agent demo has the smaller growth value than the one trained with a a human

player demo, it managed to train about 6 times faster reaching the same speed performance, therefore we conclude that it is a much better model.

3.4. **Improvements.**

3.4.1. *Improving PPO.* For this training session, the parameters `batch_size` and `learning_rate` were increased to 2048 and 0.0005 respectively and 20 agents were used, which at the beginning of the session had a cumulative environment reward of -1.562 and after just 5 million steps, reached the reward value of 0.7193, having an approximate growth value of 2.2813 and an average speed value of 24.

In Figure 5, we can see how the second version (blue) started the training session very close to the first one and oscillated until the 1.7 million steps mark, compared to the 2 million steps mark of the first version (grey). After that point, it slowly started to learn, oscillating until the 3.5 million steps mark where it reached its maximum potential and until the end of the episode maintained its value close to the maximum like the first version.

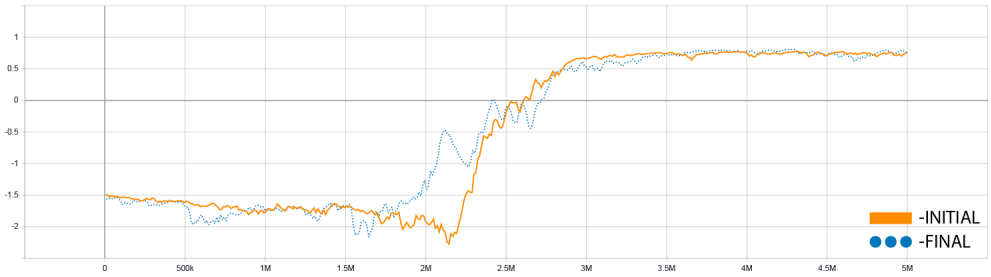

FIGURE 5. Environment cumulative reward of the PPO method initial versus final model

When comparing the first model with the improved one, the improved one has a slower speed value, 24 versus 27, but a slightly bigger growth value 2.2813 versus 2.2457. While traversing the level, the improved model has a better understanding of the environment, maintaining its traversing pattern almost exclusively on the track part of the level, compared to the first model which cuts corners in order to complete the level faster. This adds more value to the improved model, making it more realistic and better suited for this genre of video games.

3.4.2. *Improving GAIL.* Compared to the first models trained with the GAIL method, the improved ones used 20 agents, new demos and had the same configuration with only a slight increase in the gail `strength`, having the value of 0.15. The performance of the improved GAIL model using a player made

demo is very good, reaching the speed value of 21 after 5 million steps and after starting with the cumulative environment reward of -1.486, it reached the value of 0.8046, having an approximate growth value of 2.2906.

In Figure 6 we can see the training evolution of the model, compared to the previous version.

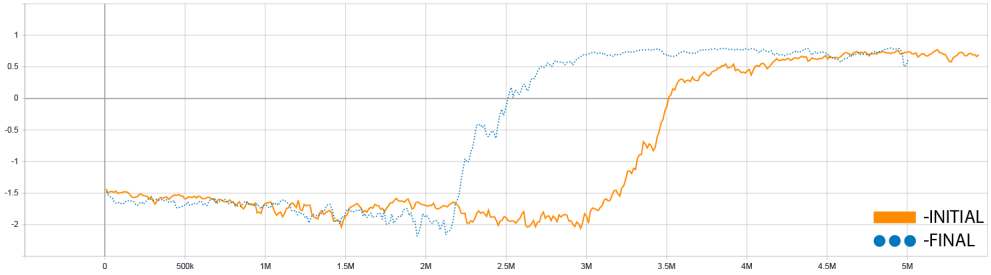

FIGURE 6. Environment cumulative reward of the GAIL method using a player demo initial versus final model

As we can see, the improved model (orange) learned faster due to the increase in the learning rate, from around the 2.1 million steps mark compared to the 3 millions steps mark of the first version (blue). It reached its maximum potential around the 3 million steps mark and from there, maintaining its value close to its maximum until the end of the session.

Both the first and the improved models have relatively the same performance, with average speed of 21, but the improved model has a slightly bigger growth value. When comparing the models while traversing the level, the improved one has a better understanding of the environment, traversing the level almost exclusively on the track part. This again adds more value to the improved model, making it more realistic and better suited for this genre of video game.

The performance of the improved GAIL model using an AI made demo is very good, reaching the speed value of 21 after 5 million steps and after starting with the cumulative environment reward of -1.527, it reached the value of 0.7133, having an approximate growth value of 2.2403.

In Figure 7 we can see the training evolution of the model, compared to the initial experiment. When comparing the two versions, the improved one has an approximately 23% increase in speed performance, going from 17 to 21, and it has a bigger growth value. While traversing the level, both the models take shortcuts, going off the track part and on to the surrounding environment, so there is no significant improvement in this department.

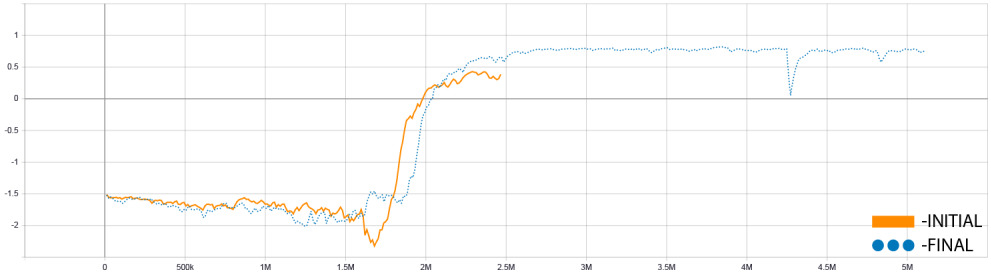

FIGURE 7. Environment cumulative reward of the GAIL method using an AI demo initial versus final model

3.4.3. *Improving BC.* Regarding the improved BC configuration, the only differences consisted of using 20 agents, just like the improved PPO and GAIL models, and using new demos from which the agents learned.

The performance of the improved BC model using a player made demo did not increase in terms of speed, having the average speed of 6, like the model before it. In terms of environmental cumulative reward, it started with the value of -1.499 and had a maximum value of 0.4614 with an approximate growth value of 1.9604.

As we can see from Figure 8, the improved version of this method started to slowly learn until reaching close to its maximum potential at the 1 million steps mark. From there until the 2.5 million steps mark held its value very steady, but after that it had an unpredictable behaviour and slowly decreased until the end of the training session.

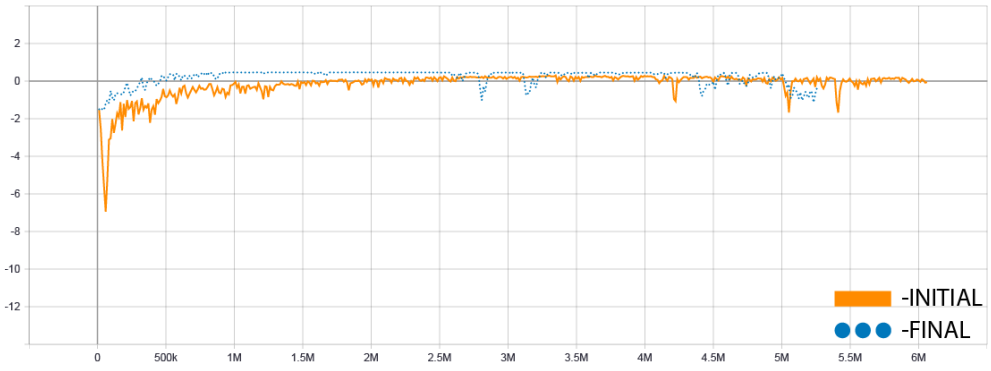

FIGURE 8. Environment cumulative reward of the BC method using a player demo initial versus final model

When compared to the previous model, the improved one has the same average speed but a bigger growth value, 1.9604 versus 1.481. While traversing the level, the improved model learned to take fewer shortcuts, by cutting less corners and staying more on the track part of the level, this in turn increases the value of the model.

The performance of the improved BC model using an AI made demo did not increase in terms of speed, having the average speed of 6, like the model before it. In terms of environmental cumulative reward, the model started with the value of -1.541 and had a maximum value of 0.5802 after 5 million steps with an approximate growth value of 2.1212.

In Figure 9 we can see how the improved model started to learn slowly, reaching its maximum potential at around the 1 million steps mark and keeping its value pretty consistent throughout the training session, until approximately around the 3.3 million steps mark where it started to raise and fall until the end of the session. Compared to the previous version it had a more stable learning rate, the first version having a very unpredictable learning pattern.

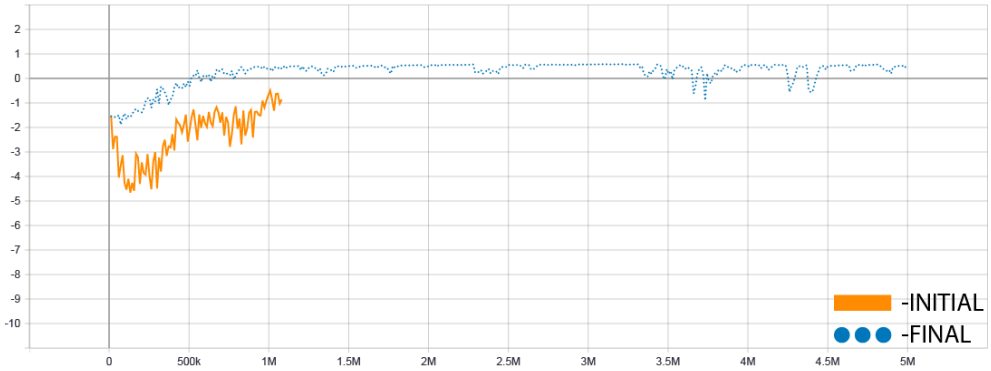

FIGURE 9. Environment cumulative reward of the BC method using an AI demo initial versus final model

When compared to the previous model, the improved one has the same average speed but a bigger growth value, 2.1212 versus 0.668. While traversing the level, the improved model follows along relatively the same pattern as the previous version, taking shortcuts, leaving the track part of the level and traversing the surrounding environment. Other than the increase in growth value, the model did not receive an increase in its value compared to the previous version.

3.5. **Final results.** When comparing the improved version of the five models trained for this experiment, we can see even more clearly how different the training methods perform and which one has the best performance.

The method with the fastest growth is the BC method (red for BC using a player demo and dark blue for BC using an AI demo), with both its models reaching their maximum potential by the 1 million steps mark. Following we have the GAIL model using an AI demo (brown), reaching its maximum at around the 2.5 million steps mark. Finally, we have the PPO (light blue) and GAIL using a player demo (orange) models, which both reached their maximum around the 3 million steps mark.

In the context of traversing the environment, the models of PPO, GAIL using a player demo and BC using a player demo have improved by staying more on the track and not taking shortcuts on their way to complete the level, with the BC model only having a slight improvement in this regard.

In Table 2, we have the values of the performance for all the five improved models (with 5 mil number of steps for each) with correct level traversal for some (does not cut through the environment like before). The best model was the one of the PPO method, which had the biggest speed value out of all five models, with the value 24. The PPO model is followed again by the GAIL models, both having the speed value of 21. Lastly, the BC models had again the poorest performance with the average speed value of 6.

Table 2. Comparison of all five improved AIs specifications

	Speed	**Starting reward**	**Ending reward**	**Growth value**	**Correct level traversal**
PPO	27	-1.562	0.7193	2.2813	Yes
GAIL& player demo	21	-1.486	0.8046	2.2906	Yes
GAIL & AI demo	21	-1.527	0.7133	2.2403	No
BC & player demo	6	-1.499	0.4614	1.9604	Yes
BC & AI demo	6	-1.541	0.5802	2.1212	No

The reason behind this ranking is that the PPO model had the best speed performance in both the original and improved model, with the increase in value coming from the improved model, which learned to traverse the environment more correctly. Next, we have the GAIL models, which had the same speed performance in the improved model, but ultimately the model using a player made demo learned to traverse the environment more correctly than the model using an AI made demo. For the final places, the BC method had the poorest performance of all five model, but the model using a player made

demo learned a little bit better to traverse the environment and ultimately this put it at an advantage compared to the one using an AI made demo.

4. Conclusions

The purpose of this article is firstly to present the power of the ML-Agents toolkit, which, as we have seen, is a very competent and accessible tool for training multiple types of intelligent agents using different training methods. This is thanks to the use of a high-level framework such as PyTorch, working in the background. This is a facilitating tool in the process of creating, training and adding artificial intelligence to video games, supporting the game development industry.

Moreover, we have shown how different methods of AI perform compared to one another in the context of a racing video game and which would be the best option to choose when developing this type of video games. The PPO method, using reinforcement learning, had the best performance of all the trained models, followed by GAIL and BC respectively. The results found in this experiment may not be definitive, as there is always room for improvement and every training game environment is different, but they are a good reference point on how each of these methods performs. The implemented application also shows the simplicity and efficiency of the training process and it is a very good graphical representation of the results found in this experiment.

References

[1] Berndt, C., Watson, I., and Guesgen, H. Oasis: an open ai standard interface specification to support reasoning, representation and learning in computer games. In *IJCAI-05 Workshop on Reasoning, Representation, and Learning in Computer Games* (2005), Citeseer, pp. 19–24.

[2] Bhattacharyya, R., Wulfe, B., Phillips, D., Kuefler, A., Morton, J., Senanayake, R., and Kochenderfer, M. Modeling human driving behavior through generative adversarial imitation learning. *arXiv preprint arXiv:2006.06412* (2020).

[3] Bojarski, M., Del Testa, D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., Jackel, L. D., Monfort, M., Muller, U., Zhang, J., et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316* (2016).

[4] Fan, X., Wu, J., and Tian, L. A review of artificial intelligence for games. *Artificial Intelligence in China* (2020), 298–303.

[5] Giusti, A., Guzzi, J., Cireşan, D. C., He, F.-L., Rodríguez, J. P., Fontana, F., Faessler, M., Forster, C., Schmidhuber, J., Caro, G. D., Scaramuzza, D., and Gambardella, L. M. A machine learning approach to visual perception of forest trails for mobile robots. *IEEE Robotics and Automation Letters 1*, 2 (2016), 661–667.

[6] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. Generative adversarial nets. *Advances in neural information processing systems 27* (2014).

[7] Ho, J., and Ermon, S. Generative adversarial imitation learning. *Advances in neural information processing systems 29* (2016), 4565–4573.
[8] Juliani, A., Berges, V.-P., Teng, E., Cohen, A., Harper, J., Elion, C., Goy, C., Gao, Y., Henry, H., Mattar, M., et al. Unity: A general platform for intelligent agents. *arXiv preprint arXiv:1809.02627* (2018).
[9] Kreminski, M., Samuel, B., Melcer, E., and Wardrip-Fruin, N. Evaluating ai-based games through retellings. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment* (2019), vol. 15, pp. 45–51.
[10] Kuefler, A., Morton, J., Wheeler, T., and Kochenderfer, M. Imitating driver behavior with generative adversarial networks. In *2017 IEEE Intelligent Vehicles Symposium (IV)* (2017), IEEE, pp. 204–211.
[11] Nandy, A., and Biswas, M. Unity ml-agents. In *Neural Networks in Unity*. Springer, 2018, pp. 27–67.
[12] Perez-Liebana, D., Liu, J., Khalifa, A., Gaina, R. D., Togelius, J., and Lucas, S. M. General video game ai: A multitrack framework for evaluating agents, games, and content generation algorithms. *IEEE Transactions on Games 11*, 3 (2019), 195–214.
[13] Quang Tran, D., and Bae, S.-H. Proximal policy optimization through a deep reinforcement learning framework for multiple autonomous vehicles at a non-signalized intersection. *Applied Sciences 10*, 16 (2020), 5722.
[14] Rollings, A., and Adams, E. *Andrew Rollings and Ernest Adams on game design*. New Riders, 2003.
[15] Safadi, F., Fonteneau, R., and Ernst, D. Artificial intelligence in video games: Towards a unified framework. *International Journal of Computer Games Technology 2015* (2015).
[16] Samak, T. V., Samak, C. V., and Kandhasamy, S. Robust behavioral cloning for autonomous vehicles using end-to-end imitation learning. *arXiv preprint arXiv:2010.04767* (2020).
[17] Sander, R. Emergent autonomous racing via multi-agent proximal policy optimization. *Embodied Intelligence* (2020).
[18] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347* (2017).
[19] Torabi, F., Warnell, G., and Stone, P. Behavioral cloning from observation. *arXiv preprint arXiv:1805.01954* (2018).
[20] Yannakakis, G. N. Game ai revisited. In *Proceedings of the 9th conference on Computing Frontiers* (2012), pp. 285–292.
[21] Zhu, J., Villareale, J., Javvaji, N., Risi, S., Löwe, M., Weigelt, R., and Harteveld, C. Player-ai interaction: What neural network games reveal about ai as play. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems* (2021), pp. 1–17.

Babeş-Bolyai University, Faculty of Mathematics and Computer Science, Cluj-Napoca, Romania

Email address: alina.calin@ubbcluj.ro