Journal of Theoretical and Applied Information Technology
15th August 2024. Vol.102. No. 15

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

REPLICATING VIDEO GAME PLAYERS' BEHAVIOR THROUGH DEEP REINFORCEMENT LEARNING ALGORITHMS

HAFSA GHARBI[1], MR. LOTFI ELAACHAK[2], AND MR. ABDELHADI FENNAN[3]

[1,2,3]Computer Science & Smart Systems Laboratory, Data & Intelligent Systems Team, FSTT, Abdelmalek Essaadi University, Tetouan, Morocco

E-mail: [1]hafsa.gharbi@etu.uae.ac.ma, [2]lelaachak@uae.ac.ma, [3]afennan@uae.ac.ma

ABSTRACT

This paper addresses the challenge of imitating the behavior of video game players using Deep Reinforcement Learning algorithms. By training intelligent agents with algorithms such as Proximal Policy Optimization (PPO), Behavioral Cloning (BC), and Generative Adversarial Imitation Learning (GAIL), we enable these agents to learn from their interactions with the game environment, optimizing their actions based on rewards and punishments. Experimental evaluations across various video games demonstrate that these trained agents can successfully mimic human player behavior in complex situations. This capability offers significant opportunities for creating challenging non-player characters (NPCs), designing adaptive difficulty levels, and enhancing the overall gaming experience. Our findings suggest that integrating Reinforcement Learning techniques allows game developers to provide more realistic and immersive gameplay, effectively bridging the gap between Artificial Intelligence and both video games and serious games.

Keywords: *Deep Reinforcement Learning, Proximal Policy Optimization, Behavioral cloning, Generative Adversarial Imitation Learning, video game.*

1. INTRODUCTION

The video game market is projected to experience exponential growth, reaching a remarkable $396.20 billion in 2023 [1]. Furthermore, rapid advancements in Artificial Intelligence and Machine Learning are creating new opportunities to enhance the gaming experience by simulating the behaviors of both video game players and serious game learners.

During the last few years, researchers [15, 16] have explored the application of various deep reinforcement learning algorithms, including deep Q-networks (DQN), policy gradients, proximal optimization policy (POO), Behavioral Cloning (BC), Generative Adversarial Imitation Learning (GAIL), and actor-critic methods (A2C), to imitate player behavior. These algorithms enable agents to learn from interactions with the game environment (2D/3D), optimizing their actions based on rewards and penalties received. The trained agents aim to mimic the decision-making processes, adaptive strategies, and skill levels of human players, thereby creating a more realistic and immersive gaming experience.

The importance of this issue lies in the growing demand for more engaging and dynamic gaming experiences. As games become more sophisticated, the need for intelligent, adaptive, and realistic non-player characters (NPCs) becomes critical. Traditional game design often relies on scripted behaviors that lack the complexity and adaptability of human players, leading to predictable and less challenging gameplay. This limitation not only affects player satisfaction but also hinders the potential for serious games to be used effectively in educational and training contexts.

In the imminent future, the seamless integration of intelligent agents into advanced serious games will revolutionize experiential learning. These sophisticated agents will have the remarkable capability to understand and learn complex behaviors through systematic experimentation. Acting as dynamic mentors, they will guide students through tackling multifaceted challenges by meticulously demonstrating step-by-step problem-solving procedures.

Addressing this challenge is crucial for advancing the field of game design and enhancing the utility of serious games. By developing and refining reinforcement learning algorithms that can

accurately imitate human behavior, we can create NPCs that provide more engaging and challenging interactions. This, in turn, can lead to better learning outcomes in educational settings and more satisfying experiences for recreational players.

In this research perspective, this paper aims to contribute to the existing literature on imitating the behavior of video game players using reinforcement learning algorithms. We will explore and evaluate the effectiveness of different algorithms such as POO, BC, and GAIL in imitating player behavior in various gaming scenarios and environments. By examining the performance and limitations of these algorithms, we seek to provide insights into their applicability for creating more realistic and challenging NPCs, designing adaptive difficulty levels, and ultimately enhancing the overall gaming experience.

The rest of this paper is organized as follows. Section 2 provides a comprehensive review of the related literature on imitating player behavior in video games using reinforcement learning. Section 3 describes the methodology and experimental setup employed in this research. Section 4 presents the results and analysis of our experiments using several metrics of evaluation. Finally, Section 5 concludes the paper with a summary of the findings, implications for future research, and potential applications in the field.

2. STATE OF THE ART

Imitation Learning In Video Game Environments

Numerous research studies have explored the potential of reinforcement learning algorithms to replicate player behavior in video games. For instance, Johnson and Miikkulainen [14] employed the PPO algorithm to train agents that can mimic expert players across various game genres. Their findings revealed that these agents exhibited behaviors similar to those of expert players, enhancing the overall gaming experience by providing more challenging and engaging gameplay.

In another study, Ho and Ermon [15] applied the GAIL algorithm to imitate the strategies of expert players in the popular game Dota 2. This approach showcased the capability of capturing complex player behaviors effectively.

Matheus et al. [17] used two machine learning techniques, namely a reinforcement learning approach and an Artificial Neural Network (ANN), in a fighting game to enable the agent/fighter to emulate human players. They incorporated a special reward function in the reinforcement learning approach, allowing the agent to exhibit specific human-like behaviors.

Daniel et al. [18] conducted research on simulating human behavior in video games using machine learning algorithms and emulating predefined behaviors. The experiments yielded promising results, demonstrating the potential of this approach.

Barros et al. [19] experimented with ANN to map the strategies of specific opponents. They trained their model online, utilizing a composite loss based on contrastive optimization, which proved effective for learning competitive and multiplayer games. Their experiments showed improved performance when the model played against offline opponents.

To replicate concurrent actions in 3D games, Harmer, Jack, et al. [20] introduced a novel architecture that efficiently allows multiple actions to be selected at each step. This architecture not only offers a 4x improvement in training time but also enhances performance by 2.5x compared to single action selection.

In a separate study, Harshit Sikchi et al [21] proposed a new framework for imitation learning that focuses on imitating a two-player ranking-based game between a policy and reward agent. In this framework, the reward agent learns to satisfy pairwise performance rankings between behaviors, while the policy agent aims to maximize this reward. According to the authors, this approach has successfully solved tasks in learning from observation that were previously considered unsolvable.

Despite these advancements, there are still significant challenges in developing intelligent agents that can fully replicate the nuanced behaviors of human players. These challenges include the need for more sophisticated models that can handle the complexity of real-time decision-making and the variability of human behavior across different game contexts.

Problem Statement: While current research has made strides in using reinforcement learning and imitation learning to emulate player behavior, there remains a gap in achieving truly adaptive and

realistic NPCs. This gap highlights the necessity for more advanced algorithms and frameworks that can better mimic the adaptive strategies and complex decision-making processes of human players. Addressing this issue is critical for enhancing the realism and engagement of video games, which is increasingly important as the gaming industry continues to grow and diversify.

3. THEORETICAL BACKGROUND

3.1. Reinforcement & Imitation Learning Algorithms

Reinforcement learning is a field of machine learning. It involves taking appropriate action to maximize benefits in a given situation. It is used by various software programs and machines to find the best possible behavior or path to follow in a given situation. Reinforcement learning differs from supervised learning in that, in supervised learning, the training data has the answer key; the model is therefore trained with the correct answer, whereas in reinforcement learning, there is no answer, but the reinforcement agent decides what to do. To perform the given task in the absence of a training dataset, it is necessary to learn from experience. The algorithm learns behavior based on observations. The actions taken by the algorithm in the environment produce feedback values that guide the learning process [2].

Proximal Policy Optimization

PPO algorithm [3,23] plays a pivotal role in training the decision function of a computer agent to successfully tackle challenging tasks. This innovative algorithm, conceived by artificial intelligence luminary John Schulman in 2017, represents a significant advancement in the field, providing a robust framework for enhancing the capabilities of computer agents through effective decision-making processes. PPO uses an on-policy approach to train a stochastic policy, meaning it explores task dynamics by sampling actions based on the latest version of its stochastic policy. The degree of randomness in action selection is influenced by initial conditions and the ongoing training process. As training progresses, the policy generally exhibits reduced randomness due to the update rule that encourages the exploitation of previously discovered rewards. However, this progression can potentially cause the policy to become stuck in local optima during the training process.

Advantage Actor-Critic

The A2C [4,24] algorithm presents a synchronous counterpart to the Asynchronous Advantage Actor-Critic (A3C) policy gradient method in reinforcement learning. In contrast to the asynchronous nature of A3C, A2C operates synchronously, employing a deterministic implementation that mandates each actor to conclude its segment of experience before initiating updates. This synchronous paradigm allows for a coordinated update procedure, wherein updates are averaged across all actors, ensuring a more stable and deterministic learning process. By synchronizing the updates, A2C mitigates potential issues associated with asynchronous updates, such as instability due to varied update frequencies and non-deterministic training trajectories. Consequently, A2C presents a promising alternative that maintains the benefits of actor-critic methods while addressing challenges posed by asynchronous implementations. Through empirical evaluation and comparative analysis, this study elucidates the efficacy of A2C as a synchronous and deterministic reinforcement learning algorithm, highlighting its utility in various environments and domains.

Behavioral Cloning

The BC algorithm [5,6] is a notable technique within the domain of reinforcement learning. It has demonstrated considerable success in tackling practical control problems across a diverse spectrum, from playing video games to navigating complex environments. By leveraging expert demonstrations or expert policies, BC learns to mimic the behavior exhibited by the expert in a given task. This approach has proven effective in scenarios where expert data is readily available, enabling the algorithm to quickly acquire high-quality policies through imitation learning. Notably, Behavioral Cloning has showcased its versatility and efficacy in real-world applications, exhibiting promising results in areas such as autonomous driving, robotic manipulation, and computer vision tasks. As research continues to evolve, further refinements and extensions to the Behavioral Cloning algorithm hold the potential to advance its applicability and performance across a broader array of challenging control problems.

Generative Adversarial Imitation Learning

GAIL [15] represents a cutting-edge approach within the realm of imitation learning and reinforcement learning. Unlike traditional Behavioral Cloning methods, GAIL introduces a

novel paradigm by incorporating adversarial training techniques. The algorithm learns a policy by concurrently training it alongside a discriminator network, which is tasked with discerning expert trajectories from those generated by the learned policy. Through this adversarial process, the policy network seeks to generate trajectories that are indistinguishable from expert demonstrations, effectively leveraging the discriminative feedback to refine its own behavior. This adversarial framework imbues GAIL with the ability to learn from sparse or unstructured expert data, making it particularly well-suited for scenarios where access to expert demonstrations is limited or costly. As a result, GAIL has garnered significant attention and demonstrated remarkable efficacy across a wide range of applications, including robotic manipulation, autonomous driving, and natural language processing tasks.

3.2. Similarity Measurement

In the realm of data analysis and machine learning, the measurement of similarity between data points serves as a fundamental component for various tasks, including clustering, classification, and information retrieval. The assessment of similarity involves quantifying the likeness or resemblance between pairs of data instances based on their respective features or attributes. Numerous methods have been developed to compute similarity, ranging from simple metrics such as Euclidean distance [7], Euclidean distance with normalization [7], Dice coefficient [8], cosine similarity[9] ,Dynamic Time Warping (DTW) [10], Manhattan distance [11], Pearson correlation [12] and Mahalanobis distance [13] to more complex techniques like kernel functions and graph-based similarity measures. Each method offers unique advantages and is suited to specific data types and application domains.

4. METHOD

4.1. Implementing Agent in a Stochastic 2D Environment with one action: Flappy Bird

4.1.1. Flappy bird game introduction

Flappy Bird [22] (see Figure.1) is a popular 2D game that challenges players to navigate a small bird through a series of obstacles. The objective is simple: keep the bird flying for as long as possible by tapping the screen or pressing a key to make it flap its wings. The bird automatically descends, and the player must time their taps to make it ascend and pass through openings in pipes. However, colliding with any obstacle or the ground ends the game. Flappy Bird's addictive gameplay, coupled with its simple yet challenging mechanics, has captivated players worldwide.

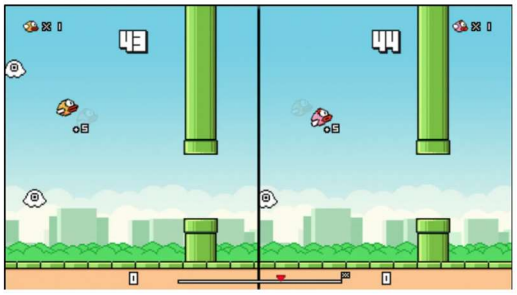

Figure 1: Flappy Bird games by DotGears Studios. Source [20]

4.1.2. PPO training process of agent (Flappy Bird)

The process consists of training BirdAgent using PPO and A2C algorithms. In the PPO training process for Flappy Bird, we follow a series of steps to train six different brains with varying configurations. Table 1 below provides a summary of the hyper-parameters used to train BirdAgent using PPO algorithm.

Table 1: Hyper-parameter values used in training Flappy bird using PPO algorithm

		Brain_1	Brain_2	Brain_3	Brain_4	Brain_5	Brain_6
Hyper-parameters	trainer_type	PPO	PPO	PPO	PPO	PPO	PPO
	batch_size	1000	64	64	64	64	64
	buffer_size	1000	10000	10000	10000	10000	10000

	learning_rate	0.0003	0.0003	0.0003	0.0003	0.0003	0.0003
	beta	0.001	0.001	0.001	0.001	0.001	0.001
	epsilon	0.2	0.7	0.7	0.7	0.7	0.7
	lambda	0.99	0.99	0.99	0.99	0.99	0.99
	num_epoch	3	4	4	4	4	4
	learning_rate _schedule	Linear	Linear	Linear	Linear	Linear	Linear
	normalize	False	False	False	False	False	False
Network-settings	hidden_units	256	256	256	256	256	256
	num_layers	2	2	2	2	2	2
	vis_encode_ type	Simple	Simple	Simple	Simple	Simple	Simple
Reward signals	gamma	0.99	0.995	0.995	0.995	0.995	0.995
	strength	1.0	1.0	1.0	1.0	1.0	1.0
	keep_checkp oints	5	5	5	5	5	5
	max_steps	50000	100000	100000	500000	500000	1M
	time_horizon	1000	1000	1000	1000	1000	1000
	summary_fre q	2500	5000	5000	25000	25000	50000
	Number of envs	1	1	3	1	3	6

As shown in Table 1, we train the first brain using the default configuration. Next, we fine-tune the hyper-parameters of the second brain's configuration file within the suggested range. Then, we increase the number of environment instances to three for the third brain to introduce parallelism.
The fourth brain increases the maximum number of training steps while keeping the other parameters the same as the second brain. Similarly, the fifth brain uses the same configuration as the fourth but with three environment instances. Finally, the sixth brain undergoes training with 1 million training steps and six environment instances. Each step involves running the reinforcement learning agents-learn command with specific parameters to execute the training process and optimize the performance of the AI agents in Flappy Bird.

4.1.3. A2C Training Process of Agent (Flappy Bird)

For the training process of the A2C algorithm in Flappy Bird, a series of steps is followed to train five distinct brains, each with varying configurations. The information in Table 2 below outlines the hyper-parameters utilized in the configuration file of each brain for the A2C algorithm.

To initiate the A2C training process, we start by using the default configuration file of the

initial brain. The training begins with a single environment. Next, we tune the hyper-parameters of the second brain's configuration file by adjusting the values within the suggested range. This tuning process also uses a single environment. For the third brain, we increase the number of environment instances to three, allowing parallel training within the same scene. In the fourth and fifth brains, we modify the maximum number of training steps to 500,000 while maintaining the parameters from the second brain, using either one environment or three environments, respectively.

Table 2: Hyper-parameter values used in training Flappy bird using A2C algorithm

		Brain_1	Brain_2	Brain_3	Brain_4	Brain_5
	trainer_type	A2C	A2C	A2C	A2C	A2C
Hyper-parameters	batch_size	1000	64	64	64	64
	buffer_size	1000	10000	10000	10000	10000
	learning_rate	0.0003	0.0008	0.0008	0.0008	0.0008
	beta	0.001	0.01	0.01	0.01	0.01
	lambda	0.99	0.99	0.99	0.99	0.99
	num_epoch	1	1	1	1	1
	learning_rate_ schedule	Linear	Linear	Linear	Linear	Linear
	normalize	True	True	True	True	True
Network-settings	hidden_units	256	256	256	256	256
	num_layers	2	2	2	2	2
	vis_encode_type	Simple	Simple	Simple	Simple	Simple
Reward signals	gamma	0.99	0.995	0.995	0.995	0.995
	strength	1.0	1.0	1.0	1.0	1.0
	keep_checkpoints	5	5	5	5	5
	max_steps	50000	100000	100000	500000	500000
	time_horizon	1000	1000	1000	1000	1000
	summary_freq	2500	5000	5000	25000	25000
	Number of envs	1	1	3	1	3

4.1.4. Behavioral Cloning with PPO Training Process of Agent (Flappy Bird)

Imitation learning involves learning from demonstrations provided by human experts, while

reinforcement learning focuses on learning through trial and error interactions with the environment. By combining BC and PPO, we leverage the strengths of both algorithms to improve the agent's performance and enhance its decision-making capabilities.

Two demonstrations of human players were recorded and used to test different combinations of hyper-parameters, the demo called Demo_1 achieved the best scores by the player, while Demo_2 obtained remarkably less scores.
During the testing of the Behavioral Cloning (BC) algorithm, different values of lambda were evaluated to determine their impact on the highest rewards achieved. Lambda is a parameter used in BC to control the trade-off between the expert demonstrations and the agent's own policy.

The results showed that varying lambda had a noticeable effect on the highest rewards obtained. Both demos achieved higher rewards when lambda was set to 0.95 compared to when it was set to 0.9.

Increasing lambda to 0.95 resulted in improved performance, indicating that giving more weight to the agent's own policy during training was beneficial. This suggests that relying more on the agent's exploration and learning from its own experiences, rather than solely relying on the expert demonstrations, led to better rewards.

Based on these findings, it is recommended to use a lambda value of 0.95 for the BC algorithm. This choice allows for a greater emphasis on the agent's own policy, promoting learning and exploration, and resulting in improved performance and higher rewards.

Here is the final hyper-parameters values (Table 3) used to train agents with BC and PPO combination:

Table 3: Hyper-parameter values used in training Flappy bird using BC Based PPO algorithm

Hyper-parameter	Value	Hyper-parameter	Value
batch_size	256	Extrinsic:gamma	0.99
buffer_size	10000	Extrinsic:strength	1.0
learning_rate	0.0005	keep_checkpoints	5
beta	0.005	time_horizon	1000
epsilon	0.3	behavioral_cloning: steps	0
lambd	0.95	behavioral_cloning: strength	0.3
num_epoch	5	behavioral_cloning: samples_per_update	0
learning_rate_schedule	linear	-	-
normalize	false	-	-
hidden_units	128	-	-
num_layers	2	-	-

4.1.5. GAIL with PPO Training Process of Agent (Flappy Bird)

Testing imitation learning algorithms, such as GAIL, in combination with PPO as a reinforcement learning method, can be a powerful approach in training intelligent agents. This combination leverages the strengths of both algorithms to improve the agent's performance and learning capabilities, involving the evaluation of the agent's performance trained with GAIL and PPO on various tasks or environments. The testing phase measures the agent's ability to imitate the expert's behavior, adapt to different scenarios, and achieve high rewards.

During the testing of the GAIL algorithm (Table 4), the parameter "use_vail" was adjusted to assess its impact on the achieved rewards. Both Demo_1 and Demo_2 were evaluated with true and false values for this parameter. When "use_vail" was set to true, it introduced a variational bottleneck within the GAIL discriminator, forcing the discriminator to learn a more general representation and reducing its tendency to be overly proficient at discriminating. This approach aimed to enhance the stability of learning. However, the results showed that setting "use_vail" to true led to slightly lower rewards compared to when it was set to false.

On the other hand, when "use_vail" was set to false, allowing the discriminator to focus solely on learning the task at hand, higher rewards were achieved for both demos. This suggests that omitting the variational bottleneck and enabling the discriminator to solely discriminate between demonstrated and actual behavior without additional constraints resulted in better rewards.

Therefore, it is recommended to set "use_vail" to false to optimize rewards when utilizing the GAIL algorithm. By doing so, the discriminator can solely focus on discriminating between the demonstrated and actual behavior, leading to improved performance in terms of reward attainment.

Table 4: Hyper-parameter values used in training Flappy bird using GAIL Based PPO algorithm

Hyper-parameter	Value	Hyper-parameter	Value
batch_size	256	Extrinsic: gamma	0.99
buffer_size	10000	Extrinsic: strength	1.0
learning_rate	0.0005	keep_checkpoints	5
beta	0.005	time_horizon	1000
epsilon	0.3	gail: gamma	0.99
lambda	0.95	gail: strength	0.05(0.3)
num_epoch	5	gail: learning_rate	0.0003
learning_rate_schedule	linear	gail: use_actions	false
normalize	false	gail : use_vail	false
hidden_units	128	Gail: hidden_units	64
num_layers	2	Gail: num_layers	2

5. RESULTS & DISCUSSION

5.1. Behavioral Cloning with PPO

Imitation learning involves acquiring skills by observing human experts, whereas reinforcement learning relies on trial-and-error interactions with the environment to learn. By integrating BC with PPO, we can harness the advantages of both methods to boost the agent's performance and refine its decision-making abilities. In our study, we recorded two demonstrations of human players to test various combinations of hyper-parameters. The demonstration labeled Demo_1 featured the highest scores achieved by the player, while Demo_2 resulted in significantly lower scores.

Table 5: Highest rewards value over number of steps

Steps	50k	100k	500k
Demo_1	2.565	2.565	1.136
Demo_2	8.582	25.195	25.398

When testing the BC algorithm, the highest rewards earned by two demos were recorded over different numbers of steps (50k, 100k, and 500k) "Table 5". Surprisingly, the demo that performed the best in terms of highest scores during game-play (demo_1) did not fare well in terms of the highest average rewards earned during training. The comparison of graphs on TensorBoard revealed that the peak values were observed at 100k steps for both players. Consequently, the decision was made to use 100k steps as a fixed hyper-parameter and fine-tune the remaining parameters.

To explain why the best demo performed poorly in terms of highest average rewards, there are a few possibilities to consider. One possibility is that the bad demo, despite not achieving a high score, better represented the behavior that the agent needed to learn. Another possibility is that the hyper-parameters used in the training process were not appropriately tuned for the good demo, resulting in ineffective learning from it. By acknowledging these observations and insights, adjustments can be made to further improve the training process and enhance the agent's performance.

Table 6: Highest rewards over different values of strength of behavioral cloning

Strength	0.1	0.3	0.5	0.8	1.0
Demo_1	18.254	8.013	2.565	0.823	-0.908
Demo_2	23.581	20.188	25.195	7.736	4.221

During testing, different values of the strength parameter in the Behavioral Cloning algorithm were evaluated "Table 6 ". The highest average rewards were obtained when using lower strengths, specifically 0.1 and 0.3, for both the bad and good demonstrations. It was observed that lower strength values resulted in better performance, potentially because the agent had more opportunities to explore and learn from its own experiences. This is particularly relevant if the expert demonstrations were not flawless or representative of all possible situations the agent could encounter. Furthermore, lower strength values can help mitigate the risk of overfitting to the expert demonstrations, which can adversely affect performance on unseen data. The strength parameter in behavioral cloning determines the weight assigned to expert demonstrations versus the agent's own policy during training. A higher strength value indicates a greater reliance on expert demonstrations, limiting the agent's exploration and independent learning. Conversely, a lower strength value encourages the agent to rely more on its own policy, fostering exploration and potentially facilitating better learning.

Given these findings, the decision was made to use a strength of 0.3 as a fixed hyper-parameter and further fine-tune other parameters in subsequent iterations of the training process.

Table 7: Highest rewards as values over different values of batch size

Batch size	64	128	256	512
Demo_1	8.013	7.827	12.045	14.797
Demo_2	20.188	20.652	17.577	11.090

During the experimental phase, the impact of different batch sizes on the BC algorithm was evaluated. Higher batch sizes "Table 7" were found to yield higher average rewards and showed improved statistics on TensorBoard compared to

lower batch sizes. Based on these results, a batch size of 256 was selected as the optimal choice.

A larger batch size can enhance the stability of the training process and contribute to better overall performance. By processing more samples in each iteration, the agent can benefit from a larger and more diverse set of experiences, potentially leading to more effective learning and improved results. Therefore, the decision was made to utilize a batch size of 256 for the subsequent stages of the training process.

Table 8: Highest rewards over different values of learning rate

	0.001	**0.0005**	**0.0003**	**0.0001**	**0.00005**
Demo_1	7.444	18.115	12.045	13.885	7.663
Demo_2	21.821	23.191	17.577	10.143	1.216

Throughout the testing stage, different values of the learning rate were evaluated for the Behavioral Cloning algorithm "Table 8". The highest rewards were achieved with learning rates of 0.0005, 0.0003, and 0.0001 for both Demo_1 and Demo_2. These learning rates resulted in better average rewards compared to the other values tested. Although the performance was promising with learning rates of 0.0007 and 0.0009, it was decided to establish 0.0005 as the standard value for this experiment. This choice is based on the consistent success achieved with this learning rate across both demos and is expected to provide a good balance between effective learning and stability.

By selecting a standard learning rate, it becomes easier to compare and evaluate the performance of the algorithm across different experiments and variations. The chosen learning rate of 0.0005 will serve as the baseline for further tuning and refinement of the BC algorithm.

Table 9: Highest rewards over hidden units and number of layers

Hidden units and number of layers	**1/64**	**2/64**	**2/128**	**2/256**	**3/256**
Demo_1	11.325	11.112	19.456	18.115	9.582
Demo_2	8.904	17.668	22.209	23.191	22.114

During performance testing, different configurations of hidden units and number of layers were evaluated for the Behavioral Cloning algorithm "Table 9". The highest rewards were obtained with the following configurations: 2 layers with 128 hidden units, and 2 layers with 256 hidden units. These configurations resulted in better rewards compared to using 3 layers with 256 hidden units.

The lower rewards obtained with the 3-layer, 256-unit configuration suggest that increasing the number of layers and hidden units may have led to over-fitting. Overfitting occurs when the model becomes too specialized in the training data and performs poorly on new, unseen data. By reducing the number of layers and hidden units, it is possible to prevent overfitting and improve the model's ability to generalize to new data.

To further refine the BC algorithm, it is suggested to explore configurations such as 3 layers with 128 hidden units or even 4 layers with 128 hidden units. These configurations have the potential to provide more precision and better performance by finding an optimal balance between complexity and generalization.

By carefully selecting the number of layers and hidden units, the BC algorithm can achieve improved results by effectively capturing the underlying patterns and dynamics of the expert demonstrations while maintaining the ability to generalize to new situations.

5.2. GAIL with PPO

Evaluating imitation learning algorithms like GAIL, combined with PPO as a reinforcement learning method, can effectively train intelligent agents. This approach capitalizes on the strengths of both algorithms to enhance the agent's performance and learning capabilities. The assessment involves testing the agent trained with GAIL and PPO across various tasks or environments, measuring its ability to mimic expert behavior, adapt to different scenarios, and achieve high rewards.

Table 10: Highest rewards as value over GAIL strength

Strength	0.01	0.05	0.1	0.3	1.0
Demo_1	7.186	14.639	11.447	16.586	24.537
Demo_2	6.591	9.458	22.409	16.693	24.620

While testing the GAIL algorithm, the highest rewards were achieved at different GAIL strengths "Table 10". For both the Demo_1 and Demo_2, the rewards varied across the different GAIL strengths tested.

It was observed that keeping the GAIL strength below approximately 0.1, as suggested in the documentation, yielded better results. This indicates that a lower GAIL strength allows the trained agent to prioritize receiving extrinsic rewards rather than strictly copying the demonstrations.

By striking a balance between imitation and exploration, the agent can learn from the demonstrations while also adapting its policy to maximize the extrinsic rewards in the given environment. Therefore, it is recommended to use a GAIL strength of 0.05 for this particular scenario, based on the testing results.

Table 11: Highest rewards as value over ANN Architecture

	1_64	2_64	1_128	2_128	1_256
Demo_1	4.283	13.919	10.376	14.639	11.568
Demo_2	8.096	16.440	13.981	9.458	11.842

Also during the testing of this algorithm, the highest rewards were observed for different Neural Network architectures "Table 11". Both demos demonstrated varying rewards across the different Neural Network architectures tested.

In accordance with the documentation, it is recommended to choose a Neural Network architecture that strikes a balance between compressing the original observation and effectively differentiating between demonstrated and actual behavior. The chosen architecture should be small enough to encourage compression but not too small to hinder the discrimination process.

Based on the testing results, the 2/64 architecture yielded the highest reward. Therefore, it is advisable to select the 2/64 Neural Network architecture for this particular scenario, as it aligns with the recommended guidelines in the documentation.

Table 12: Highest reward as values over GAIL learning rate

	0.001	0.0005	0.0003	0.0001	0.00005
Demo_1	14.241	6.360	13.919	18.364	7.819
Demo_2	9.046	3.122	16.440	11.900	4.451

As part of the GAIL's algorithm testing, different learning rates were evaluated to determine their impact on the rewards achieved "Table 12". The Demo_1 and Demo_2 exhibited varying rewards across the different learning rates tested.

Based on the testing results, it was observed that a learning rate of 0.0003 or a value close to it consistently yielded the highest rewards for both demos. Therefore, for future experiments and implementations, it is recommended to use a learning rate of 0.0003 as it has shown to be effective in maximizing the rewards achieved by the GAIL algorithm.

5.3. Evaluating the integration of IL algorithms

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Evaluating applied imitation learning algorithms using different metrics is crucial for assessing the similarity between an agent's game-play and the player's game-play on which it was trained. It provides valuable insights into the effectiveness and performance of the imitation learning process. By employing various metrics, we can obtain a comprehensive understanding of how well the agent has learned to imitate the player's behavior and game-play style.

In this experiment, we tried to record 3 types of different play styles (in order to simulate different players), the first play style is centered around floating in the lowest area of the pipes, followed by mid lane area and high area, while recording the demos from the rounds game-plays, we train 3 bots according to those play styles and save actions and observations and using those to perform evaluations.

Table 13: Action Vectors Comparison is between brains and their associated demos

Metrics	euclidian_ distance_ score	euclidian_d istance_ wNorm_ score	cosine_ distance_ score	dice_ coefficient_ score	values_ similarity_ short_ score	difference
Player/round1 /brain1	19.51	47.81	50.0	92.09	94.28	2.19
Player/round2 /brain2	12.84	42.28	52.09	96.46	90.24	6.22
Player/round3 /brain3	10.54	42.81	50.0	96.42	91.98	4.44
Player/round1 /brain2	17.91	45.51	50.0	91.17	91.42	0.25
Player/round1 /brain3	19.07	47.76	50.0	91.77	94.29	2.52
Player/round2 /brain1	14.28	46.60	53.34	94.38	93.56	0.82
Player/round2 /brain3	13.65	45.92	51.60	93.90	93.42	0.48
Player/round3 /brain1	10.81	43.0015	51.28	97.08	92.004	5,076
Player/round3 /brain2	12.84	44.96	51.61	98.26	92.58	5.68

Action vectors

As we notice in Table 13, the most representative metrics are dice_coefficient and values_similarity.

The pairs that are similar have a higher value for the first value (dice_coefficient_score) and a lower value for the second value (values_similarity_short_score), while the pairs that are dissimilar have a lower value for the first value and a higher value for the second value.

In contrast, there is no clear similarity pattern among the second set of pairs. Some pairs have a higher first value and some have a higher second value. This suggests that for these pairs, the similarity between the player and the bot actions is not consistent across both similarity measures.

However, since the values are close, it is difficult to make a definitive conclusion and other factors should be taken into consideration as well.

State vectors

Using state vectors instead of action vectors for measuring similarity between a player's game-play and a bot's game-play in the Flappy Bird game can work better for several reasons.

Firstly, state vectors capture a more comprehensive representation of the game state at any given moment. They contain information such as the bird's position, the distance to obstacles, and the current score, which collectively define the current state of the game. By comparing the similarity of state vectors, we can assess how closely the bot's game-play aligns with the player's game-play in terms of overall progress, obstacle avoidance, and general game strategy.

Secondly, Flappy Bird is a game where timing and reaction play a crucial role. Actions alone may not provide enough context to accurately measure similarity. The same action at different moments in the game can yield different outcomes based on the current state. By considering state vectors, we can capture the temporal dynamics and contextual information necessary for evaluating similarity accurately.

Table 14: State Vectors Comparison is between brains and their associated demos

	cosine_similarity	euclidian_distance	dtw_distance	dice_coefficient	cosine	euclidean	manhattan	manhattan2	pearson_corr	mahalanobis_dist
Player_round1/brain1	67.22	72.62	19.91	2.56	65.84	36.30	44.30	43.85	72.86	35.32
Player_round2/brain2	79.22	73.94	19.9	1.89	77.52	35.89	45.27	43.60	80.25	34.08
Player_round3/brain3	79.73	72.611	22.612	1.76	80.519	35.74	43.89	43.147	81.06	36.39
Player_round1/brain2	75.90	71.68	15.89	2.78	69.92	41.13	53.06	51.89	78.62	29.50
Player_round1/brain3	70.09	67.09	13.44	1.51	71.09	41.93	54.47	53.49	73.39	25.82
Player_round2/brain1	73.67	72.48	18.42	1.76	73.75	35.95	43.99	44.08	76.53	34.36
Player_round2/brain3	81.98	75.135	21.89	1.36	79.30	34.98	44.72	42.28	83.42	37.52
Player_round3/brain1	75.07	68.24	13.67	2.54	75.45	40.90	52.47	52.15	77.22	29.29
Player_round3/brain2	81.70	74.62	21.07	2.79	78.90	37.19	46.21	45.39	82.44	35.52

Based on Tables 13 and 14, which display the similarity values between various brains and the demos they were trained on using different metrics, we can derive some general conclusions.

In the first table, where the comparison is between brains and their associated demos, we observe varying similarity values across different metrics. Cosine similarity, Pearson correlation, and Mahalanobis distance consistently yield high similarity scores, indicating a strong resemblance between the brains and their corresponding demos. On the other hand, metrics such as Euclidean distance, DTW distance, and dice coefficient tend to provide lower similarity values, suggesting some dissimilarity or deviations in behavior.

In the second table, which compares different brains with different demos, the goal is to identify patterns and determine which metric is most useful in distinguishing between similar and dissimilar pairs. Cosine similarity, Pearson correlation, and Mahalanobis distance again emerge as reliable metrics, consistently assigning higher values to pairs that exhibit similarity in game-play. Euclidean distance, DTW distance, and dice coefficient continue to provide lower values, indicating dissimilarity between pairs. This suggests that the selected metrics are effective in capturing the similarity between different brains and demos, providing valuable insights into the extent of imitation learning.

When comparing our findings with existing literature, we see that previous studies have also highlighted the effectiveness of Cosine similarity and Pearson correlation in capturing behavioral similarities [9, 12]. However, our results extend these findings by demonstrating that Mahalanobis distance can be equally reliable in this context, which has been less emphasized in earlier works. Conversely, the lower performance of Euclidean distance and DTW distance aligns with the observations made by Ho and Ermon [15] in their study on Dota 2, where these metrics were found to be less effective in capturing nuanced player behaviors.

Overall, these tables demonstrate the importance of employing multiple metrics to evaluate similarity in imitation learning. Each metric captures different aspects of similarity, and the consistent patterns observed across metrics validate the effectiveness of the chosen evaluation methods. By considering these metrics collectively, we can gain a comprehensive understanding of the degree to which the trained brains mimic the behavior of the demos. This information is valuable for assessing the quality of the imitation learning process, identifying areas for improvement, and making informed decisions in further refining the algorithms.

Future research should address the following open issues:

- Improving the robustness of similarity metrics across different game genres to ensure generalizability.
- Investigating the impact of more diverse training datasets on the performance and adaptability of the trained agents.
- Exploring advanced reinforcement learning techniques that can mitigate the limitations observed with Euclidean and DTW distances.
- Enhancing the interpretability of the learned behaviors to better understand the decision-making processes of the agents.

6. CONCLUSION

This paper delves into the application of deep reinforcement learning algorithms to imitate video game player behavior. By training intelligent agents with algorithms like Proximal Policy Optimization, Behavioral Cloning, and Generative Adversarial Imitation Learning, we empower them to learn from game environment interactions and optimize actions based on rewards and penalties. Through experiments across various video games, our research showcases these agents' potential to mimic human player behavior in complex scenarios, thereby enabling the creation of challenging non-player characters, adaptive difficulty levels, and enriched gaming experiences.

Our results align with and build upon existing studies, demonstrating the robust capabilities of these algorithms in various gaming contexts. Compared to the work of Johnson and Miikkulainen [14], who focused on PPO, our integration of GAIL and BC has shown additional strengths in adaptive behavior simulation. Furthermore, our findings corroborate the effectiveness of GAIL observed by Ho and Ermon [15], while also highlighting new insights into the utility of Mahalanobis distance for behavior comparison.

This work not only advances our understanding of artificial intelligence in gaming but also paves the way for future innovations aimed at

crafting more lifelike and immersive gaming encounters.

Addressing the identified open issues in future research will be crucial for further advancements in this field, providing new directions for the development of more sophisticated and adaptable intelligent agents.

ACKNOWLEDGMENTS

We would like to extend our heartfelt thanks to Mr. Hassan Faham for his invaluable contributions and expert guidance throughout this research. His insights and dedication were pivotal in the successful completion of this study. We are deeply appreciative of the support and mentorship provided.

REFERENCES

[1] Statista. "GamesWorldwide.",Visited[www.statista.com /outlook/amo/media/games/worldwide].
[2] Leemon Baird. Residual algorithms: Reinforcement learning with function approximation. In Proceedings of the 12th International Conference on Machine Learning (ICML 1995), pages 30–37. Morgan Kaufmann, 1995.
[3] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," arXiv.org, https://arxiv.org/abs/1707.06347, arXiv:1707.06347 [cs.LG].
[4] Mnih, V., "Asynchronous Methods for Deep Reinforcement Learning", arXiv e-prints, 2016. doi:10.48550/arXiv.1602.01783.
[5] Dean A Pomerleau. Alvinn: An autonomous land vehicle in a neural network. In Advances in Neural Information Processing Systems, 1989.
[6] Michael Bain and Claude Sammut. A framework for behavioral cloning. In Machine Intelligence 15, 1995.
[7] Dokmanic, I., Parhizkar, R., Ranieri, J., and Vetterli, M., "Euclidean Distance Matrices: Essential theory, algorithms, and applications", IEEE Signal Processing Magazine, vol. 32, no. 6, pp. 12–30, 2015. doi:10.1109/MSP.2015.2398954.
[8] Dice LR. Measures of the amount of ecologic association between species. Ecology. 1945;26:297–302.
[9] M. Kusner, Y. Sun, N. Kolkin and K. Weinberger, "From Word Embed- dings To Document Distances", in Proceedings of the 32nd International Conference on Machine Learning, pp. 957-966, 2015.
[10] Bringmann, K., Fischer, N., van der Hoog, I., Kipouridis, E., Kociumaka, T., and Rotenberg, E., "Dynamic Dynamic Time Warping", arXiv e-prints, 2023. doi:10.48550/arXiv.2310.18128.
[11] Black, Paul E. "Manhattan distance". Dictionary of Algorithms and Data Structures.
[12] Spearman, C. (1904). The proof and measurement of association between two things, 15,72-101.The American Journal of Psychology, 100(3/4), special centennial issue (Autumn -Winter,1987), 441-471. DOI: 10.2307/1422689. https://www.jstor.org/stable/1422689.
[13] Mahalanobis, Prasanta Chandra (1936). "On the generalized distance in statistics" . Proceedings of the National Institute of Sciences of India. **2** (1): 49–55. Retrieved 2016-09-27.
[14] Johnson, M., & Miikkulainen, R. (2019). Adapting Deep Reinforcement Learning for Imitation Learning in Video Games. In Proceedings of the 14th International Conference on the Foundations of Digital Games (pp. 1-8).
[15] Ho, J., & Ermon, S. (2016). Generative Adversarial Imitation Learning. In Advances in Neural Information Processing Systems (pp. 4565-4573).
[16] Nair, A., Srinivasan, P., Blackwell, S., Alcicek, C., Fearon, R., De Maria, A. & Beattie, C. (2015). Massively parallel methods for deep reinforcement learning. In Proceedings of the 33rd International Conference on Machine Learning (Vol. 48, pp. 2927-2936).
[17] M. R. F. Mendonça, H. S. Bernardino and R. F. Neto, "Simulating Human Behavior in Fighting Games Using Reinforcement Learning and Artificial Neural Networks," 2015 14th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames), Piaui, Brazil, 2015, pp. 152-159, doi: 10.1109/SBGames.2015.25.
[18] D. de Almeida Rocha and J. Cesar Duarte, "Simulating Human Behaviour in Games using Machine Learning," 2019 18th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames), Rio de Janeiro, Brazil, 2019, pp. 163-172, doi: 10.1109/SBGames.2019.00030.
[19] Pablo Barros, Alessandra Sciutti, All by Myself: Learning individualized competitive behavior with a contrastive reinforcement learning optimization, Neural Networks, Volume 150, 2022, Pages 364-376, ISSN 0893-6080, https://doi.org/10.1016/j.neunet.2022.03.013.

[20] Harmer, Jack, et al. "Imitation learning with concurrent actions in 3d games." 2018 IEEE Conference on Computational Intelligence and Games (CIG). IEEE, 2018.
[21] Sikchi, H., Saran, A., Goo, W., & Niekum, S. (2023). A Ranking Game for Imitation Learning. Transactions on Machine Learning Research. https://openreview.net/forum?id=d3rHk4VAf0
[22] Nick Statt, "Flappy Bird returns", 2014, https://www.cnet.com/tech/gaming/flappy-bird- returns-yet-only-on-amazon-fire-tv/
[23] Schulman, John, et al. "Proximal policy optimization algorithms." arXiv preprint arXiv:1707.06347 (2017).
[24] Huang, Shengyi, Anssi Kanervisto, Antonin Raffin, Weixun Wang, Santiago Ontañón, and Rousslan Fernand Julien Dossa. "A2C is a special case of PPO." arXiv preprint arXiv:2205.09123 (2022).