

## Article

# Efficient Difficulty Level Balancing in Match-3 Puzzle Games: A Comparative Study of Proximal Policy Optimization and Soft Actor-Critic Algorithms

Byounggwon Kim and Jungyoon Kim \*

Department of Game Media, College of Future Industry, Gachon University,  
Seongnam-si 13120, Republic of Korea; qldrnjs2001@gachon.ac.kr

\* Correspondence: kjoyoon@gachon.ac.kr

**Abstract:** Match-3 puzzle games have garnered significant popularity across all age groups due to their simplicity, non-violent nature, and concise gameplay. However, the development of captivating and well-balanced stages in match-3 puzzle games remains a challenging task for game developers. This study aims to identify the optimal algorithm for reinforcement learning to streamline the level balancing verification process in match-3 games by comparison with Soft Actor-Critic (SAC) and Proximal Policy Optimization (PPO) algorithms. By training the agent with these two algorithms, the paper investigated which approach yields more efficient and effective difficulty level balancing test results. After the comparative analysis of cumulative rewards and entropy, the findings illustrate that the SAC algorithm is the optimal choice for creating an efficient agent capable of handling difficulty level balancing for stages in a match-3 puzzle game. This is because the superior learning performance and higher stability demonstrated by the SAC algorithm are more important in terms of stage difficulty balancing in match-3 gameplay. This study expects to contribute to the development of improved level balancing techniques in match-3 puzzle games besides enhancing the overall gaming experience for players.

**Keywords:** match-3 puzzle game; balancing test; reinforcement learning; PPO; SAC



**Citation:** Kim, B.; Kim, J. Efficient Difficulty Level Balancing in Match-3 Puzzle Games: A Comparative Study of Proximal Policy Optimization and Soft Actor-Critic Algorithms.

*Electronics* **2023**, *12*, 4456. <https://doi.org/10.3390/electronics12214456>

Academic Editor: Stefanos Kollias

Received: 5 October 2023

Revised: 24 October 2023

Accepted: 27 October 2023

Published: 30 October 2023



**Copyright:** © 2023 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

## 1. Introduction

With the proliferation of smartphones, the accessibility of gaming has significantly broadened its reach to a wider audience, and the level of awareness has improved considerably. Among these gaming genres, match-3 puzzle games have gained substantial popularity. Match-3 games are a popular genre of puzzle games that involve matching three or more tiles of the same color or type. They appeal to individuals of all age groups because these games have relatively simple rules, a lack of association with violence and sensationalism, and brief gameplay duration [1]. Distinguished by their rapid content consumption, these games focus on a singular objective: to clear the stage. Therefore, it is important to keep updating stages quickly and with well-designed levels to attract new users and retain the existing player base. To create a game stage with the appropriate level of difficulty, manually generating the right combination of blocks can be a time-consuming and challenging task for a human. This is particularly evident in a match-3 game where the difficulty level must be carefully controlled by generating blocks one at a time. Thus, the ability to swiftly create and test the difficulty level of multiple stages becomes a must-have for developers specializing in match-3 puzzle games [2].

Artificial intelligence (AI) holds one of the most important fields in the game industry [3], and research has been conducted in various domains including AI for autoplaying non-player characters (NPCs), data analysis, and automated testing. In particular, AI advancements in NPCs and level design are of great interest to the game industry as they directly influence the immersive gaming experience [4,5]. Consequently, research on how to

enable AI to make precise and efficient situational judgments is steadily progressing [6–8]. In particular, the automatic generation of game stages with different difficulty levels was a focus of interest in applying AI. This includes the generation of stages for combat games [9], 2-D platform games [10,11], and even the creation of entirely new games [12].

In the context of the match-3 puzzle game genre, some research is underway to create automated AI that can aid in designing intricate stage levels. The Monte Carlo Tree Search (MCTS) is a search algorithm that has been applied to various games including match-3 games. It can be used to guide AI in making strategic decisions and improving its performance over time [13]. Genetic algorithms (GAs) have been also used for automatic generation of match-3 game stages. A genetic algorithm is a problem-solving method that emulates the natural evolutionary process. It represents solutions using data structures, assesses their fitness through a fitness function, and iteratively evolves multiple solutions to improve fitness. In implementing a match-3 game with a game level balancing concept, a genetic algorithm was employed to provide the computer with a probability of success and discover a gene sequence with the desired mean and standard deviation [14].

Many match-3 puzzle games primarily employ the MCTS algorithm for implementing automatic gameplay and level balancing tests. MCTS offers the advantage of automatically adapting to new elements introduced into the game by organizing the game state into a tree structure. Nonetheless, it suffers from the disadvantage of exponential tree growth when dealing with a large number of cases, thus resulting in increased search times [15]. Thus, due to the relatively slow search speed of MCTS, it has become a common practice to combine MCTS with deep neural networks (DNNs) or convolutional neural networks (CNNs) [7]. Then, recent research has paid attention to reinforcement learning (RL) for developing autoplay AI to test the difficulty levels in a stage because RL can offer a relatively short learning time and high accuracy. Among RL methods, Proximal Policy Optimization (PPO) has been studied to train AI agents in stage construction and level balancing verification in match-3 puzzle games [7,16]. For the development of an autoplay AI, the game needs to undergo training using a suitable RL algorithm. However, little research has been conducted to find the most appropriate algorithm in learning the match-3 game genre. In this regard, the purpose of the study is to make a comparison between the two most commonly used reinforcement learning algorithms, PPO and the Soft Actor-Critic (SAC), to establish criteria for assessing algorithms with superior performance for match-3 puzzle games. Both the PPO and SAC algorithms are model-free reinforcement learning approaches, so they have the capability to learn in diverse environments without requiring a predetermined model. Furthermore, these algorithms are adaptable for various gaming environments. Notably, the PPO algorithm is the most widely utilized among RL algorithms and the SAC algorithm, a more recent addition to the field, is well recognized for its incorporation of entropy measurement into the reward policy. Hence, this paper tests and compares these two algorithms to find the most effective one for achieving difficulty level balancing tests in the match-3 games.

In the case of match-3 games, it is difficult to learn appropriate behaviors based on the random behavior of the agent alone, so an imitation learning method was added to partially imitate human behavior, and we proceeded with a total of four learning methods. After learning all four methods in the two algorithms, the resulting performance was compared and analyzed to determine the criteria for selecting the appropriate algorithm to learn the match-3 puzzle game.

The remainder of this paper is constructed as follows. Section 2 provides a discussion on the relevant studies including RL and Unity ML Software Development Kit (SDK). Section 3 outlines the design and implementation of a match-3 game system where AI agents are trained to learn match-3 gameplay. Section 4 describes the training of the agent for a match-3 puzzle game with two different algorithms and four distinct training methods for each algorithm. The learning results and the discussion on their interpretation are provided in Sections 5 and 6. Finally, Section 7 concludes the study.

## 2. Relevant Studies

### 2.1. Monte Carlo Tree Search (MCTS)

The algorithm previously employed for level designing in match-3 games was MCTS (Monte Carlo Tree Search). MCTS is a popular algorithm that has been successfully applied in various games like chess, Go, and poker [17–19]. It is an iterative process that involves constructing a tree through simulations and assessing the nodes (actions) in the tree based on game outcomes to select the most valuable actions. The simplicity of the MCTS algorithm makes it suitable for game applications, and it can be adapted to games with high branching factors by focusing the tree search on promising subtrees, potentially outperforming traditional algorithms. However, a drawback is that as the number of scenarios increases, the search process becomes exponentially longer, leading to a substantial increase in computational requirements.

To address this challenge, methods like deep neural networks (DNN) and convolutional neural networks (CNN) have been utilized as alternative algorithms for learning in match-3 games. Nonetheless, MCTS still necessitates learning for each unique in-game situation, especially in environments where block positions and orientations change randomly [7]. Additionally, whenever new blocks or rules are introduced, previously acquired knowledge becomes obsolete. Even if CNNs are employed, their primary limitation is the requirement of an extensive dataset of gameplay data [13].

### 2.2. Reinforcement Learning (RL)

Within the domain of AI machine learning, RL (Reinforcement Learning) stands out as a significant field. In RL, an agent engages in continuous interactions with its environment to acquire knowledge and discover the optimal policy that maximizes its rewards [20]. The classification system of RL is divided into agents. An agent in RL has three main components: Policy, Value Function, and Model. First, the Policy represents the behavior pattern of the agent. It determines the action to be taken in a given state and it serves as a guide for decision making during the learning process. Second, the Value Function serves as a prediction mechanism for the reward value that is expected to be received after executing a particular action in a specific state. It constitutes a weighted sum of all future reward values that the agent anticipates obtaining when taking that specific action from the current state. Third, the Model is the agent's predictive understanding of the environment by estimating the subsequent state and reward values. It can be divided into a State Model and a Reward Model.

The first condition that distinguishes RL algorithms is the presence or absence of a Model for the environment. Having a Model enables planning as it allows the agent to anticipate changes in the environment that results from its actions. Thus, the agent can efficiently plan and execute optimal behavior beforehand and it will lead to more efficient actions.

The second condition is about how to use the Value Function and Policy. In the cases where the Value Function is entirely understood, deriving the optimal Policy becomes a straightforward process by choosing the action with the highest value within each state. This is called an Implicit Policy and agents that solely learn the Value Function with an Implicit Policy are known as (1) Value-based agents. On the other hand, if the Policy is already known to be optimal, the Value Function becomes unnecessary as it acts as an intermediate calculation for Policy creation. Agents that exclusively learn policies without Value Functions are called (2) Policy-based agents. Value-based agents have the advantage of utilizing data more efficiently while Policy-based agents offer more reliable learning outcomes as they directly optimize for the desired objectives. Moreover, agents exist that combine both Value Functions and Policies, known as (3) Actor-Critic agents [21].

### 2.3. Soft Actor-Critic (SAC) Algorithm and Proximal Policy Optimization (PPO) Algorithm

The Soft Actor-Critic (SAC) algorithm, formerly known as Soft Q-Learning, is a method that incorporates an entropy measure of the Policy into the reward in order to enhance

the efficiency of the agent's exploration. By doing so, the algorithm encourages the agent to learn a Policy that enables proficient task performance while maintaining a level of randomness in its actions. Employing a framework of maximum entropy reinforcement learning, the SAC is integrated with a non-policy Actor-Critic model.

The Proximal Policy Optimization (PPO) algorithm draws inspiration from the Trust Region Policy Optimization (TRPO) algorithm. Both algorithms employ techniques like target network and experience replay to increase algorithm stability and limit aggressive updates when updating network weights. This is achieved by selecting or tuning the suitable hyperparameters for the gradient used in the weight update. Conservative policy iteration updates use a proportional combination of old and new weights to update. Additionally, the conservative policy iteration update is utilized, which involves a proportional combination of old and new weights for updates [22].

This paper compares the PPO algorithm, which has been extensively studied in the context of traditional match-3 games, with the more recently researched SAC algorithm. The PPO algorithm is a Policy-based method that builds upon the lineage of Policy Optimization. It aims to maximize the advantages of Policy Gradient while addressing the limitations of the TRPO algorithm. On the other hand, the SAC algorithm is designed for Actor-Critic agents and incorporates elements of both Policy Optimization and Q-Learning. It introduces an entropy measurement technique to enhance exploration efficiency. In this paper, both the SAC and PPO algorithms were selected because they model free RL, so these algorithms can adapt and learn in various environments without requiring a predefined model [23]. Consequently, they are well suited to a wide range of game environments [24].

#### 2.4. Unity ML-Agents

Unity ML-Agents (Release 10) represents an ML (Machine Learning) Software Development Kit (SDK) designed to function within the Unity 3D Engine environment. By utilizing the Unity ML SDK, configuring ML becomes more straightforward compared to other ML SDKs. It enables the implementation of deep reinforcement learning, evolutionary strategies, and other techniques in games and simulations created with the Unity Engine. Additionally, the SDK provides a Python API to facilitate the rapid configuration of training environments for intelligent agents.

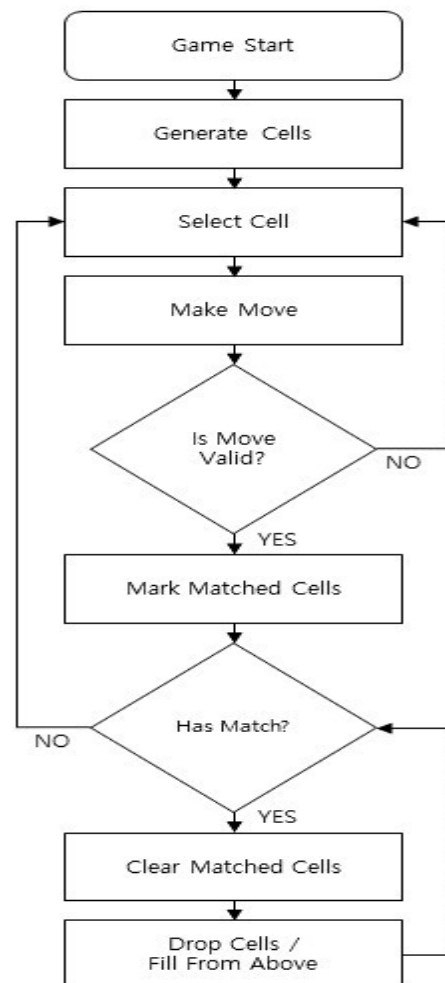
In Unity ML-Agents, Observations play a crucial role as they provide the agent with essential information about the state of the environment. These Observations can take the form of vectors represented as floating-point numbers within a scene including references and the number of rendered camera outputs. Consequently, game developers can define Observations that correspond to any numeric information accessible to the components of the agent. Also, the agent can request brain decisions either at fixed or dynamic intervals as determined by the environment developer. The reward functions, which provide crucial learning cues to the agents, can be defined or modified at any point during the simulation using the Unity Scripting System. Additionally, the simulation can reach a completion state at either the individual agent level or for the entire environment. This happens when invoked via a Unity script or when a predefined maximum number of steps is reached. When resetting the environment, developers have the flexibility to define a set of reset environment parameters that are accessible and manipulable from the Python API. This allows for fine-tuning and customizing the simulation to meet specific requirements [25–28].

### 3. Design and Implementation

#### 3.1. Match-3 Puzzle Game Design

For the comparative analysis of the algorithms proposed in this study, the block types were categorized into six types and a game was designed with one regular block and two special blocks. Special blocks were set as blocks that could yield higher rewards than regular blocks, so that the agent can learn to prioritize the acquisition of special blocks. The puzzle board consists of an  $X * Y$  field, and the gameplay follows the standard match-3 puzzle game rules in which missing blocks are filled from the top to the bottom.

Figure 1 shows the design flowchart of the match-3 puzzle game to be implemented. The game starts with the organization of cells on the board. The agent then selects a cell and makes a swipe move, after which the validity of the move is assessed. In the case of the move being invalid, the agent is prompted to select the cell again. On the contrary, if the move is valid, the matched cells are identified and their type uniformity is checked. If the cells are not of the same type, the agent is required to select the cells once more. On the other hand, if the cells match in type, a new cell is generated at the empty location. The process continues to check for matching cells and if they are found to be of the same type, the sequence repeats. However, if none of the matching cells share the same type, the flow loops back to cell selection for the agent to continue the game.

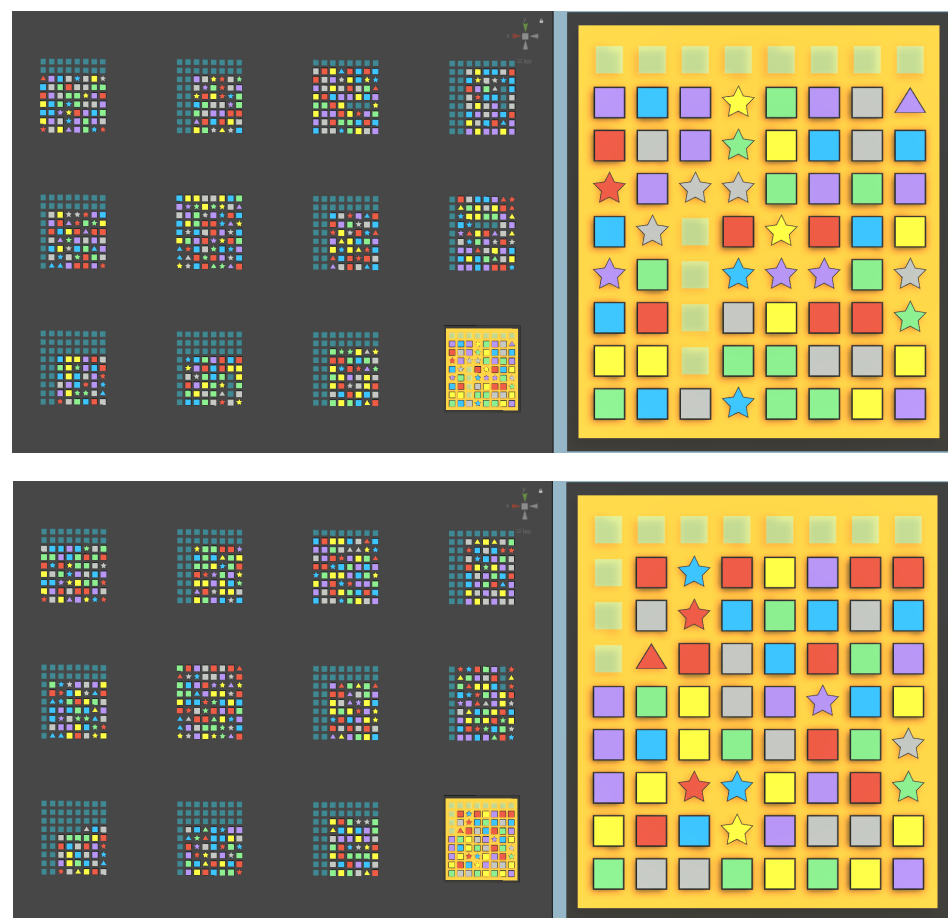


**Figure 1.** Flowchart of the match-3 puzzle game design.

### 3.2. Implementing a Match-3 Puzzle Game

To facilitate training with Unity ML-Agents, the game was implemented following the specified configuration. Multiple game levels were created within a single scene as shown in Figure 2. This design allows for the simultaneous training of numerous agents in order to enhance the efficiency of the training process. The game board structured with blocks featuring six different colors. A core rule is to align three or more of identical colors in a row to facilitate their removal from the board. Essentially, the boards take a square shape but when an additional rule awards extra points when players successfully match triangle and star-shaped blocks to eliminate them.

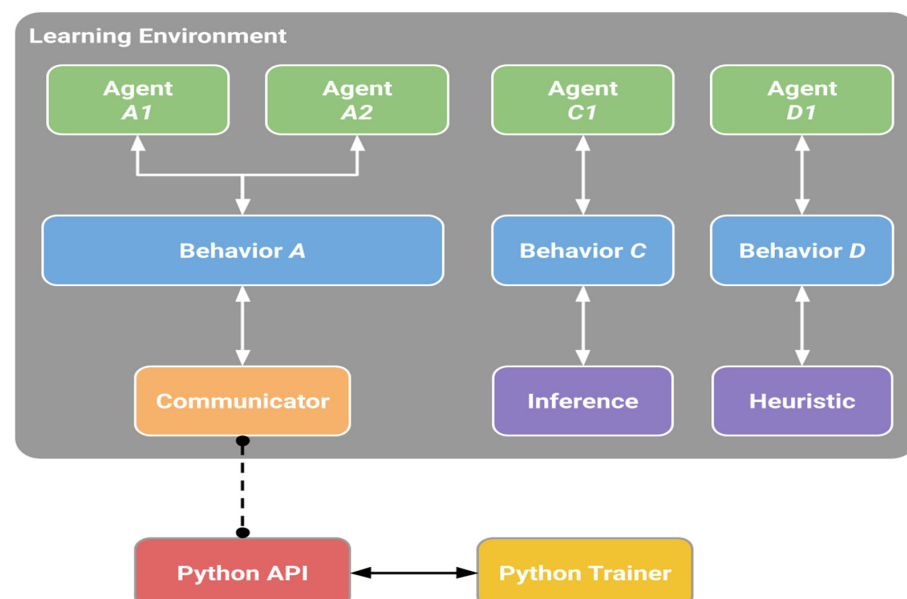




**Figure 2.** Screenshot of match-3 puzzle game implementation.

### 3.3. Unity ML-Agents and TensorFlow

Within Unity ML-Agents, interaction with the Python API is streamlined via an External Communicator. The Python API in this case is TensorFlow, a widely used open-source machine learning library crafted by Google. As shown in Figure 3, ML-Agents makes use of TensorFlow for performing reinforcement learning tasks.



**Figure 3.** Unity ML-Agents and Python API (TensorFlow).

TensorBoard is a powerful visualization tool in TensorFlow that enables users to visualize the loss values and other relevant data during the learning process of ML-Agents. In this study, TensorBoard was used to graphically visualize the actual training result data and enable a comparative analysis of the learning outcomes from the two algorithms of the Soft Actor-Critic (SAC) and the Proximal Policy Optimization (PPO).

## 4. Experiment

### 4.1. Learning Environment and Four Learning Methods

In this paper, the experimental environment was configured as shown in Table 1.

**Table 1.** Learning environment.

| Section                | Details            |
|------------------------|--------------------|
| CPU                    | Intel Core i7-9700 |
| GPU                    | GeForce GTX 1660   |
| MEMORY                 | 32 GB              |
| Unity Engine Version   | Unity 2020.1.15f1  |
| Tensorflow Version     | 2.2.2              |
| Unity ML-Agent Version | ML Release 10      |

The Intel Core i7-9700 is manufactured by Intel Corporation, which is located in Santa Clara, California, US. The GeForce GTX 1660 is a graphics card manufactured by NVIDIA Corporation, which is located in Santa Clara, California, US. After applying Unity ML-Agents to the designed match-3 puzzle game system, the study proceeded with four learning methods for each algorithm based on 5,000,000 trials to compare the performance of the two algorithms. It is difficult for randomly acting agents to effectively learn and understand the rules of match-3 puzzle games. Therefore, four learning methods were trained simultaneously to highlight differences and conduct a thorough comparison.

As shown in Table 2, four different learning methods were applied to learn the two algorithms. First, observation learning constitutes an approach where knowledge acquisition transpires through the observation of agents in random actions. It is further divided into two sub-methods: Vector Observation, which observes vectors, and Visual Observation, which focuses on visual elements such as colors. Next, heuristic learning emerges as an imitation learning method wherein an agent's conduct is governed not by randomness but is influenced by a code that reflects a degree of human intuition. The Greedy Heuristic method refines and applies human intention more accurately and minimizes unnecessary actions as much as possible. The simpler version of this method is referred to as the Simple Heuristic method.

**Table 2.** Learning methods.

| Machine Learning | SAC Algorithm             | PPO Algorithm      |
|------------------|---------------------------|--------------------|
| Observation      | Vector Observation (VE-O) | Vector Observation |
|                  | Visual Observation (VI-O) | Visual Observation |
| Heuristic        | Simple Heuristic (S-H)    | Simple Heuristic   |
|                  | Greedy Heuristic (G-H)    | Greedy Heuristic   |

### 4.2. Setting Trainer

To enable simultaneous training of the four learning methods, hyperparameter values were set for both the Proximal Policy Optimization (PPO) and Soft Actor-Critic (SAC) algorithms. Hyperparameters are values that designers define to customize the model according to their requirements. If no specific values are set, default values defined by the system will be applied. Customizing these hyperparameters is essential for optimizing the training process and achieving desired learning outcomes for each learning method. The configuration file containing these parameter settings is referred to as a trainer.

## 5. Results

The learning result data of the two reinforcement learning algorithms were categorized based on time and the number of iterations. The results were visualized into two types of graphics to evaluate and understand the performance of the algorithms in the context of the match-3 puzzle game environment. Comparing the Soft Actor-Critic (SAC) and Proximal Policy Optimization (PPO) algorithms based on cumulative reward and entropy is a common approach in the field of reinforcement learning. When comparing SAC and PPO, analyzing their cumulative rewards helps to understand which algorithm is more effective at solving a particular task. Higher cumulative rewards typically indicate better performance in terms of task completion or goal achievement.

Entropy, in the context of reinforcement learning, measures the level of disorder or uncertainty in a given dataset or system. It is a metric that quantifies the amount of information contained in a dataset and is commonly used to evaluate the quality of a model and its ability to make accurate predictions. Claude E. Shannon introduced Equations (1) and (2) to mathematically express the connection between probability and heterogeneity. These equations serve as fundamental tools in information theory.

$$H(X) = -\sum (p_i * \log_2 p_i) \quad (1)$$

$$Entropy(p) = -\sum_{i=1}^N p_i * \log_2 p_i \quad (2)$$

In information theory, entropy quantifies the uncertainty associated with a random variable. The formula for information entropy, Equation (3), is defined as follows, with the random variable  $X = \{x_1, x_2, x_3, \dots, x_m\}$ .





$$H(X) = \sum_{i=1}^m p(x_i) \cdot \log \frac{1}{p(x_i)} = -\sum_{i=1}^m p(x_i) \cdot \log p(x_i) \quad (3)$$

In this paper, Equation (3) is employed as the entropy formula to calculate the complexity measure of the system [29]. The greater the complexity of the system and the more unknown states it encompasses, the higher the information entropy becomes. Higher entropy values indicate a heterogeneous dataset with many different classes, while lower entropy indicates a pure and homogeneous subset of the data. Decision tree models use entropy to determine the best splits to make informed decisions and build accurate predictive models. When comparing SAC and PPO, the resulting entropy graph serves as a measure of the randomness in the model's decisions [30]. In a successful training process, the entropy value should exhibit a gradual and consistent decrease. If it decreases too rapidly, it may indicate that the model has become overly deterministic, and in such cases, increasing the value of the hyperparameter 'beta' is recommended. Beta controls the trade-off between maximizing the expected cumulative reward and maximizing entropy, influencing the model's level of randomness. In summary, the cumulative reward graph within the learning results served as an indicator of the player-agent's learning efficiency while the entropy graph provided insights into the stability of the learning outcomes. The cumulative reward graph represents the agent's total reward value over training iterations, and higher values indicate better learning performance. On the other hand, the entropy graph illustrates the entropy value for each training run. A lower entropy value signifies more stable learning results.

### 5.1. Results of SAC Algorithm

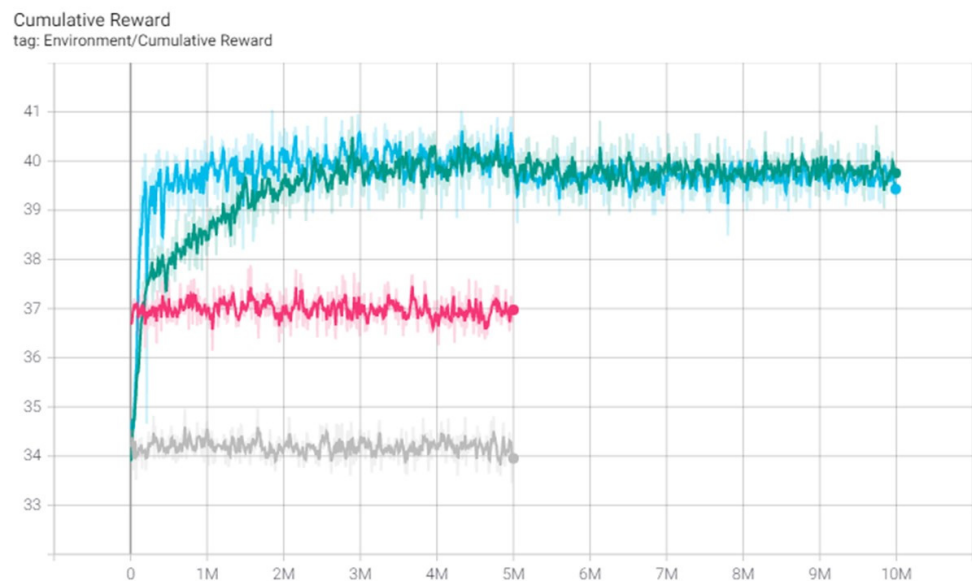
As shown in Figure 4, the training result graphs are color-coded according to the training method. The No.1 graph (sky blue) represents training with Visual Observation (VI-O), the No.2 graph (pink) with Greedy Heuristic (G-H), the No.3 (green) graph with Vector Observation (VE-O), and the No.4 (grey) graph with Simple Heuristic (S-H).



-  1. SAC Algorithm Visual Observation Training
-  2. SAC Algorithm Greedy Heuristic Training
-  3. SAC Algorithm Vector Observation Training
-  4. SAC Algorithm Simple Heuristic Training

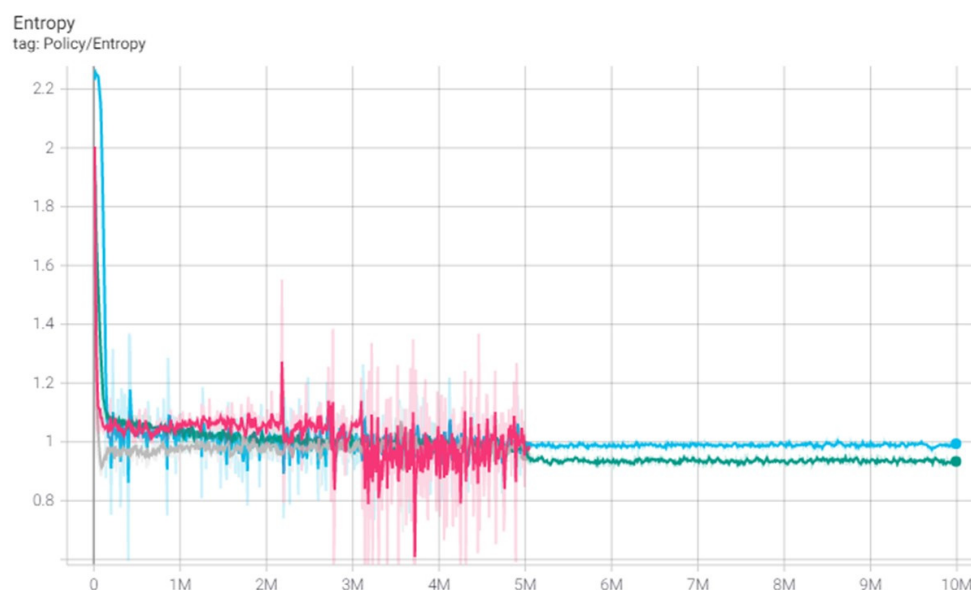
**Figure 4.** SAC algorithm training results—classification by color.

As shown in Figure 5, it illustrates the cumulative reward per training number when training with the SAC algorithm. The No.1 (blue) and No.3 (green) graphs, representing the observation learning method, demonstrate a continuous increase in cumulative reward value with repeated training. The maximum value exceeds 40, starting from an initial value of 34. However, the No.2 (pink) and No.4 (grey) graphs, corresponding to the heuristic learning method, do not exhibit any significant increase in cumulative reward value with repeated training. The No.3 graph (green, S-H) maintains a value around 34 while the No.2 graph (pink, G-H) hovers around 37. Furthermore, both the No.1 (sky blue) and No.3 (green) graphs, representing the observation learning method, do not show any further increase in the cumulative reward value beyond 3,000,000 training runs.



**Figure 5.** SAC algorithm training results—cumulative reward graph.

Figure 6 shows the entropy values of the results obtained from training with the SAC algorithm. All four learning methods rapidly converged from an initial value of 2.2 to a value of 1 and they maintained this value around 1 throughout the learning process. However, in the No.2 graph (red, G-H), the value's deviation became more pronounced after 3,000,000 runs. This indicates that the learning results for the G-H method are less stable compared to the other learning methods.



**Figure 6.** SAC algorithm training results—entropy graph.

### 5.2. Results of PPO Algorithm

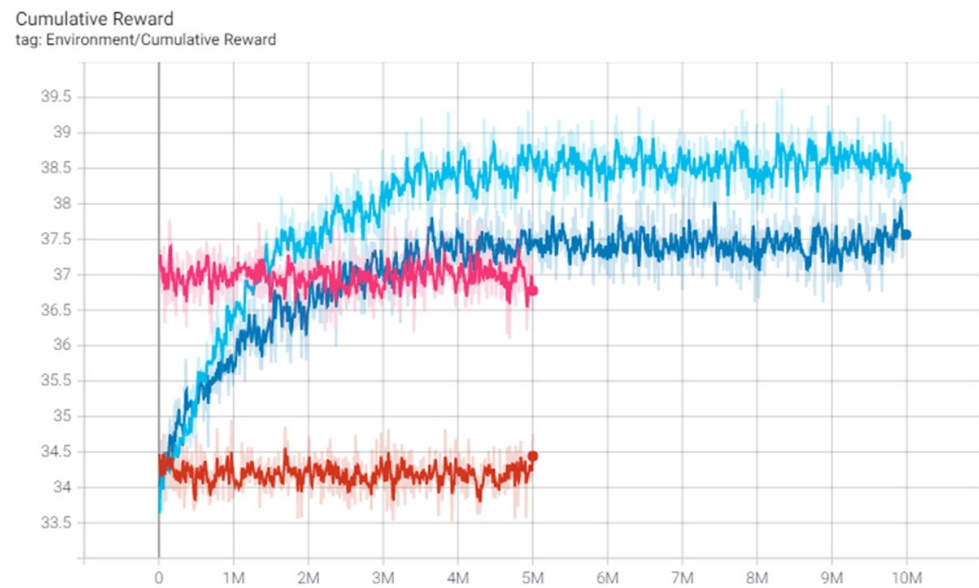
In Figure 7, the training result graph for the PPO algorithm is color-coded according to the training method. The No.1 graph (blue) represents training with VI-O, the No.2 graph (red) with S-H, the No.3 graph (sky blue) with VE-O, and the No.4 graph (pink) with G-H.

- ☒ 1. PPO Algorithm Visual Observation Training
- ☒ 2. PPO Algorithm Simple Heuristic Training
- ☒ 3. PPO Algorithm Vector Observation Training
- ☒ 4. PPO Algorithm Greedy Heuristic Training

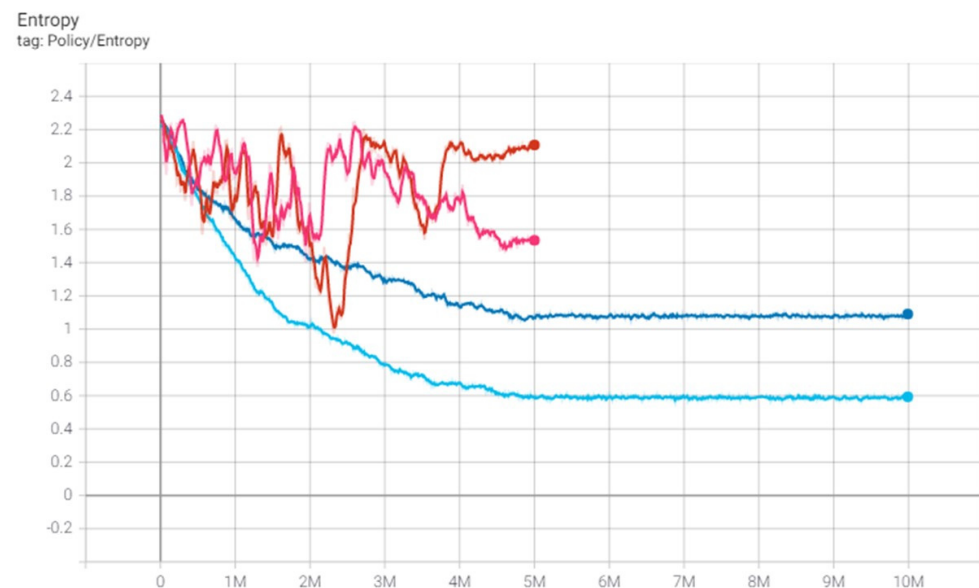
**Figure 7.** PPO algorithm training results—classification by color.

As shown in Figure 8, in the No.3 graph (sky blue, VE-O), a steady increase in the cumulative reward value is observed as training is repeated. The maximum value surpasses 38.5, starting from an initial value of 34. The No.1 graph (blue, VI-O) also shows an increase in the cumulative reward value, rising from an initial value of approximately 34 to a maximum value exceeding 37.5. On the other hand, the No.2 (red) and No.4 (pink) graphs, representing the heuristic learning method, do not exhibit a significant increase in the cumulative reward as observed in the PPO algorithm results. The No.2 graph (red, S-H) maintains a value around 34, while the No.4 graph (pink, G-H) maintains a value around 37.

Figure 9 illustrates the entropy values of the results learned by the PPO algorithm. The No.1 graph (blue, VI-O) shows a slow convergence to 1. On the other hand, the No.2 graph (red, S-H) exhibits a faster convergence to a value of 0.6, which is lower than 1, for the VE-O learning method. All four graphs, numbered from 1 to 4, initially start with a value of approximately 2.2. Analyzing the No.1 and No.3 graphs (sky blue, Observation), a steady decrease can be seen in the entropy value over time. In contrast, the No.2 and No.4 graphs (pink, Heuristic) show a larger deviation between the entropy values of 2.2 and 1.



**Figure 8.** PPO algorithm training results—cumulative reward graph.



**Figure 9.** PPO algorithm training results—entropy graph.

## 6. Discussion

As shown in Table 3, the training result data were organized for each algorithm and training method by the cumulative reward, entropy, and training time per number of training iterations. This study tried 1,000,000 and 5,000,000 runs as the baseline and compared the early and late training periods. A comparison was made between the early and late training periods.

In the first analysis of cumulative reward results, it is evident that for both the Soft Actor-Critic (SAC) and Proximal Policy Optimization (PPO) algorithms, the heuristic learning method does not significantly enhance the agent's performance. In contrast, the performance gradually improves with repeated training using the observation learning method. To be more specific, within the observational learning approach, the Visual Observation (VI-O) method yielded a performance gain of +0.94 for the SAC algorithm, while the PPO algorithm benefited more from the Vector Observation (VE-O) method with a +1.9 gain.

**Table 3.** SAC vs. PPO algorithm learning results comparison.

| Learning Method           | No of Learning | SAC                |               |                | PPO                |               |                |
|---------------------------|----------------|--------------------|---------------|----------------|--------------------|---------------|----------------|
|                           |                | Cumulative Rewards | Learning Time | Entropy (Nats) | Cumulative Rewards | Learning Time | Entropy (Nats) |
| Visual Observation (VI-O) | 1 m            | 39.07              | 3 h 47 m      | 1.09           | 35.78              | 1 h 42 m      | 1.64           |
|                           | 5 m            | 40.01              | 19 h 34 m     | 0.99           | 37.48              | 8 h 36 m      | 1.07           |
| Vector Observation (VE-O) | 1 m            | 38.70              | 3 h 47 m      | 1.01           | 36.64              | 1 h 42 m      | 1.41           |
|                           | 5 m            | 39.91              | 19 h 34 m     | 0.96           | 38.54              | 8 h 36 m      | 0.58           |
| Simple Heuristic (S-H)    | 1 m            | 34.69              | 7 h 34 m      | 1.01           | 33.86              | 3 h 24 m      | 1.82           |
|                           | 5 m            | 33.90              | 1 d 12 h      | 0.95           | 34.76              | 16 h 45 m     | 2.11           |
| Greedy Heuristic (G-H)    | 1 m            | 37.04              | 7 h 35 m      | 1.03           | 37.26              | 3 h 24 m      | 1.94           |
|                           | 5 m            | 37.03              | 1 d 12 h      | 1.02           | 35.50              | 16 h 45 m     | 1.53           |

Secondly, in terms of learning time, the SAC algorithm required an average of 18 min per training session in observation learning, as opposed to 9 min with the heuristic learning method. These results indicated that the observation learning method was twice as efficient. Conversely, the PPO algorithm averaged 9 min per training session with the heuristic learning method and only 4 min with the observation learning method. Thus, the observation learning method was two times faster than the heuristic learning method. When comparing the SAC and PPO algorithms, the PPO algorithm was approximately twice as fast.

Thirdly, the analysis of entropy values suggests that when seeking rapid learning outcomes, the PPO algorithm's VE-O method is optimal. On the contrary, for achieving the best learning performance with a slower learning pace, the SAC algorithm's VE-O learning method is recommended.

In conclusion, this paper reveals that, for the purpose of implementing stage difficulty balancing in a match-3 puzzle game, the SAC algorithm (VI-O) offers the best learning performance, despite the PPO algorithm exhibiting twice the learning speed. This conclusion is based on the fact that the superior learning performance and higher stability demonstrated by the SAC algorithm are more important in terms of stage difficulty balancing in match-3 gameplay.

## 7. Conclusions

This study compared two reinforcement learning algorithms, Soft Actor-Critic (SAC) and Proximal Policy Optimization (PPO), to identify the optimal algorithm for training an agent to construct stages and balance difficulty levels in match-3 games. Through a comparative analysis of the SAC and PPO algorithms, four different learning methods were also applied including heuristic and observation learning methods.

Upon comparing the results of accumulated rewards, learning time, and entropy, we observed that the PPO algorithm exhibited the fastest learning speed. Nevertheless, when meticulously playing a match-3 puzzle game and assessing stage difficulty for the purpose of achieving balance, the higher accumulated reward value and stability of entropy takes precedence over learning speed. Consequently, this paper found that the reinforcement learning approach utilizing the SAC algorithm delivers the most optimal performance.

By employing the suggested reinforcement learning methods in this study, game developers can reduce the labor and time required for these tasks, thus resulting in more efficient game development processes. This also can contribute to creating more engaging and enjoyable gaming experiences for players and further advance the field of reinforcement learning in the context of game development.

A limitation of this paper is that the training was performed using only simple bullock movements, so the training took place in a simplified environment compared to actual match-3 games. Currently released match-3 puzzle games feature complex map structures, stage-clearing missions, special blocks, block counts, game items, and obstacles. These elements are used to adjust level balancing and difficulty. Therefore, learning in a simplified

match-3 puzzle game environment may yield different results when the environment includes these additional factors. One emerging topic in the match-3 game and AI industry is the use of machine learning algorithms to create more personalized game experiences for players. This involves using data on the player's behavior and preferences to adjust the game mechanics and level design to better suit the player's individual needs and preferences. Thus, further research can be conducted with personized level balancing as well as the development of a tool to create stages with customizable stage difficulty.

**Author Contributions:** Conceptualization, B.K. and J.K.; methodology, B.K.; software, B.K.; validation, B.K. and J.K.; formal analysis, B.K.; data curation, B.K.; writing—original draft preparation, B.K.; writing—review and editing, J.K.; visualization, B.K. and J.K.; supervision, J.K.; project administration, J.K.; funding acquisition, J.K. All authors have read and agreed to the published version of the manuscript.

**Funding:** This research was supported by Culture, Sports and Tourism R&D Program through the Korea Creative Content Agency grant funded by the Ministry of Culture, Sports and Tourism in 2023 (Project Name: Cultural Technology Specialist Training and Project for Metaverse Game, Project Number: RS-2023-00227648), Contribution Rate: 100%.

**Data Availability Statement:** The data presented in this study are available on request from the corresponding author.

**Acknowledgments:** This work was also supported by the Gachon University Research Fund of 2020 (GCU-202008460010). The authors thank Delicious Games Inc. for their assistance with data collection and statistical analysis. And this work was supported by Culture, Sports and Tourism R&D Program through the Korea Creative Content Agency grant funded by the Ministry of Culture, Sports and Tourism in 2023 (Project Name: Cultural Technology Specialist Training and Project for Metaverse Game, Project Number: RS-2023-00227648), Contribution Rate: 100%.

**Conflicts of Interest:** The authors declare no conflict of interest.

## References

1. Gevolution. Available online: <http://www.gevolution.co.kr/rank/> (accessed on 13 August 2023).
2. Coulom, R. Efficient Selectivity and Backup Operations in MonteCarlo Tree Search. In *Computer and Games*; Springer: London, UK, 2006; Volume 4630, pp. 72–83.
3. Cha, M.H. Smart Commerce Case Study Using 4th Industrial Revolution Technology-AI, VR. *JDAEM* **2020**, *7*, 395–407. [\[CrossRef\]](#)
4. Kopel, M.; Hajas, T. Implementing AI for Non-player Characters in 3D Video Games. In Proceedings of the Intelligent Information and Database Systems: 10th Asian Conference, ACIIDS 2018, Dong Hoi City, Vietnam, 19–21 March 2018. [\[CrossRef\]](#)
5. Compton, K.; Mateas, M. Procedural Level Design for Platform Games. In Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, Marina Del Rey, CA, USA, 20–23 June 2006. [\[CrossRef\]](#)
6. Kim, S.J. NPC Battle AI Using Genetic Algorithm and Neural Network in MMORPG. Master's Thesis, Hoseo University, Asan, Republic of Korea, 2007.
7. Park, D.G.; Lee, W.B. Design and Implementation of Reinforcement Learning Agent Using PPO Algorithm for Match 3 Gameplay. *JCIT* **2021**, *11*, 1–6. [\[CrossRef\]](#)
8. An, H.Y.; Kim, J.Y. Design of a Hyper-Casual Futsal Mobile Game Using a Machine-Learned AI Agent-Player. *Appl. Sci.* **2023**, *13*, 2017. [\[CrossRef\]](#)
9. Kang, S.J.; Shin, S.H.; Cho, S.H. A Game Level Design Technique Using the eGenetic Algorithms. *J. Korea Comput. Graph. Soc.* **2009**, *15*, 13–21.
10. Sorenson, N.; Pasquier, P.; DiPaola, S. A Generic Approach to Challenge Modeling for the Procedural Creation of Video Game Level. *IEEE Trans. Comput. Intell. AI Games* **2011**, *3*, 229–244. [\[CrossRef\]](#)
11. Smith, G.; Whitehead, J.; Mateas, M.; Treanor, M.; March, J.; Cha, M. Launchpad: A Rhythm-Based Level Generator for 2-D Platformers. *IEEE Trans. Comput. Intell. AI Games* **2011**, *3*, 1–16. [\[CrossRef\]](#)
12. Browne, C.; Maire, F. Evolutionary Game Design. *IEEE Trans. Comput. Intell. AI Games* **2010**, *2*, 1–16. [\[CrossRef\]](#)
13. Shin, Y.C.; Kim, J.W.; Kim, Y.B. Playtesting in Match 3 Game Using Strategic Plays via Reinforcement Learning. *IEEE Access* **2020**, *8*, 51593–51600. [\[CrossRef\]](#)
14. Park, I.H.; Oh, K.S. Automatic Generation of Match-3 Game Levels using Genetic Algorithm. *J. Korea Game Soc.* **2019**, *19*, 25–32. [\[CrossRef\]](#)
15. Silver, D.; Huang, A.; Maddison, C.J.; Guez, A.; Sifre, L.; Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvan, V.; Lanctot, M.; et al. Mastering the game of Go with deep neural networks and tree search. *Nature* **2016**, *529*, 484–489. [\[CrossRef\]](#)

16. Kim, B.G.; Kim, J.Y. Optimizing Stage Construction and Level Balancing of Match-3 Puzzle Game with PPO Algorithm Machine Learning. *Electronics* **2023**, *12*, 4098. [CrossRef]
17. Hsueh, C.H.; Wu, I.C.; Tseng, W.J.; Yen, S.J.; Chen, J.C. An analysis for strength improvement of an MCTS-based program playing Chinese dark chess. *Theor. Comput. Sci.* **2016**, *644*, 63–75. [CrossRef]
18. Lee, B.D. The most promising first moves on small Go boards, based on pure Monte-Carlo Tree Search. *J. Korea Game Soc.* **2018**, *18*, 59–67. [CrossRef]
19. Arrington, R.; Langley, C.; Bogaerts, S. Using Domain Knowledge to Improve Monte-Carlo Tree Search Performance in Parameterized Poker Squares. In Proceedings of the AAAI Conference on Artificial Intelligence, Phoenix, AZ, USA, 12–17 February 2016.
20. Jung, D.Y. Automatic Road Traffic Collection Model based on Deep Learning Image Recognition. *JNCIST* **2022**, *11*, 465–475. [CrossRef]
21. Introduction to Reinforcement Learning. Available online: <https://deepmind.com/learning-resources/-introduction-reinforcement-learning-david-silver> (accessed on 13 August 2023).
22. Muzahid, A.J.; Kamarulzaman, S.F.; Rahman, M.A. Comparison of PPO and SAC Algorithms towards Decision Making Strategies for Collision Avoidance Among Multiple Autonomous Vehicles. In Proceedings of the 2021 International Conference on Software Engineering & Computer Systems and 4th International Conference on Computational Science and Information Management (ICSECS-ICOCSIM), Pekan, Malaysia, 24–26 August 2021. [CrossRef]
23. [RLKorea] Unity ML-Agents Presentation. Available online: <https://www.slideshare.net/KyushikMin1/rlkorea-unity-mlagents> (accessed on 13 August 2023).
24. Kim, D.H.; Jung, H.J. Comparison Reinforcement Learning Algorithms used Game AI. In Proceedings of the KISS Fall Conference 2021, Gunsan, Republic of Korea, 28–30 October 2021.
25. 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* **2018**, arXiv:1809.02627. [CrossRef]
26. Lai, J.; Chen, X.-L.; Zhang, X.-Z. Training an Agent for Third-Person Shooter Game Using Unity ML-Agents. Available online: <https://dpi-journals.com/index.php/dtce/article/view/29442> (accessed on 1 October 2023).
27. Keehl, O.; Smith, A.M. Monster Carlo: An MCTS-based Framework for machine Playtesting Unity Games. In Proceedings of the 2018 IEEE Conference on Computational Intelligence and Games, Maastricht, The Netherlands, 14–17 August 2018.
28. Johansen, M.; Pichlamair, M.; Risi, S. Video Game Description Language Environment for Unity machine Learning Agents. In Proceedings of the 2019 IEEE Conference on Games, London, UK, 26 September 2019.
29. Ramakrishna, M.T.; Venkatesan, V.K.; Izonin, I.; Havryliuk, M.; Bhat, C.R. Homogeneous Adaboost Ensemble Machine Learning Algorithms with Reduced Entropy on Balanced Data. *Entropy* **2023**, *25*, 245. [CrossRef] [PubMed]
30. Chen, W.; Wong, K.; Long, S.; Sun, Z. Relative Entropy of Correct Proximal Policy Optimization Algorithms with Modified Penalty Factor in Complex Environment. *Entropy* **2022**, *24*, 440. [CrossRef] [PubMed]

**Disclaimer/Publisher’s Note:** The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.