Optimizing Stage Construction and Level Balancing of Match-3 Puzzle Game with PPO Algorithm Machine Learning

ByoungGwon Kim and JungYoon Kim *

Department of Game Media, College of Future Industry, Gachon University, Seongnma-si 13120, Gyeonggi-do, Republic of Korea; qldmje2001@gamil.com
* Correspondence: kjyoon@gachon.ac.kr

Abstract: Although the global market has witnessed a proliferation of diverse match-3 puzzle games, achieving success in this competitive market remains challenging. The crucial factors that determine the success of match-3 puzzle games are the creation of numerous engaging stages and precise level balancing. The purpose of this study is to propose a match-3 puzzle game system that aims at identifying the most effective algorithm for training artificial intelligence agents in stage construction and level balancing verification. To validate the systems’ usefulness, this paper conducted experiments with the Proximal Policy Optimization (PPO) algorithm and obtained cumulative reward and entropy value graphs. Consequently, it has been confirmed that the system can be employed to compare learning outcomes for each algorithm and identify the optimal algorithm suitable for match-3 puzzle games. The use of machine learning technology in match-3 puzzle games holds the promise of revolutionizing game development and leading to the creation of more captivating and rewarding gaming experiences for players.

Keywords: match-3 puzzle game; stage construction; level balancing; artificial intelligence; machine learning

1. Introduction

With the widespread adoption of smartphones, match-3 puzzle games have become a highly popular genre of mobile games that are enjoyed by people of all ages. For example, the globally renowned game Candy Crush Saga, a classic match-3 puzzle game, has been captivating players for nearly a decade and continues to hold the top position as the highest grossing game in the puzzle category since its mobile launch [1,2].

The success of match-3 puzzle games can be attributed to their ease of monetization even if the game quality is not exceptionally high. However, in order to stand out from the competition and appeal to players, continuous and rapid updates with well-designed stages are crucial [3]. For instance, Candy Crush Saga currently features more than 10,000 stages, with weekly updates every Wednesday. The reason for this continuous expansion is that players are progressing through stages at a rate that outpaces the weekly updates. Thus, developers need to focus on creating a large number of well-created level designs and carefully organizing them. This process involves thorough testing of various levels while also conducting level balancing tests.

Recent studies have delved into the integration of artificial intelligence (AI) within the realm of digital game development. AI algorithms can be used for procedural content generation, which can automatically create game levels, characters, and environments to save time and effort [3–6]. AI can assist in designing levels by analyzing data on player behavior, preferences, and interactions to optimize the game’s level design. In addition, it can be used for various aspects such as automatic play, NPC (Non-Player Character) AI, data analysis, and automatic testing. NPC AI involves algorithm-controlling characters that act intelligently within the game world, and automatic play is an AI-driven gameplay
that is performed without human intervention. In particular, automatic play AI for level design are attracting attention from the game industry because they directly affect the players’ immersion and the capacity for AI to make precise and efficient situational decisions [7].

In the match-3 puzzle game genre, autoplay AI plays a pivotal role in further enhancing sophisticated level design in the match-3 puzzle game genre [8]. In the pursuit of developing an AI system for automated gameplay, the significance of machine learning (ML) cannot be overstated. It is essential for developers to carefully select a reinforcement learning algorithm that is most suitable for the specific game [9]. Nevertheless, determining the optimal algorithm can be a challenging task due to the diverse types of algorithms available in reinforcement learning. In fact, Monte Carlo Tree Search (MCTS) is primarily employed to develop an AI system for automatic gameplay assistance in level design [10]. It is an iterative algorithm that constructs a tree by simulating gameplay and evaluates the nodes (actions) of the tree based on the game’s outcomes in order to select only the most valuable actions [11]. MCTS offers the advantage of automatically adapting to new elements introduced into the game, as it organizes the game state into a tree. However, it comes with the disadvantage that the size of the tree grows exponentially when the number of cases is large, resulting in increased search time [12]. Due to these limitations, there have been instances of developing autoplay AI through reinforcement learning with relatively short learning times and high accuracy. However, there have been few reported cases in the gaming industry due to the lack of technology and support for machine learning [13].

As of now, there exists no one-size-fits-all algorithm that guarantees superior results for learning the intricacies of the match-3 game genre. To address the issue, the objective of this study is to design and implement a system that can identify the most appropriate reinforcement learning algorithm for the match-3 puzzle game genre. Furthermore, the study seeks to validate the system’s proficiency in conducting balancing tests to ensure its effectiveness by showcasing the Proximal Policy Optimization (PPO) algorithm. For this, the study established a specific system for training machine learning models in the environment of Unity 3D Engine version 2020.1.15f1 and Tensor Flow version 2.4.0. Unity’s ML SDK is the most commonly used among game developers and is an open-source library for dataflow programming in machine learning. To facilitate communication between Unity 3D and Tensor Flow, Unity ML-Agents were configured with release 20. This created an experimental environment where reinforcement learning algorithms from Unity ML could be trained and evaluated within Unity 3D and Tensor Flow. To verify the proposed system, this paper used the PPO algorithm to check whether the system can be utilized to find an optimized algorithm that performs well in the match-3 puzzle game.

2. Relevant Studies

2.1. Machine Learning

Artificial intelligence refers to a technological domain where computer programs are designed to emulate human learning, reasoning, perception, and natural language understanding. Within the diverse landscape of AI, machine learning (ML) stands out as a branch that endows computers with cognitive capabilities, such as thinking and learning [14].

Machine learning revolves around the study of algorithms that enable computers to learn autonomously. Its primary focus lies in developing computer algorithms that enhance AI through learning from experience. Two fundamental aspects of machine learning are representation and generalization. Representation deals with how data is structured and evaluated, while generalization pertains to the processing of unfamiliar data. This field finds its roots in computational learning theory and boasts numerous practical
applications. Machine learning includes various types of algorithms including supervised learning, unsupervised learning, and reinforcement learning (RL).

Reinforcement learning is a process in which an agent constantly interacts with its environment to gather information and determine the optimal policy that maximizes its reward. This is achieved by selecting actions that assess the environment and lead to the highest possible reward [15]. To be specific, reinforcement learning is composed of two fundamental elements: an agent and an environment. The agent observes the environment, chooses an action based on a policy, and then carries out the selected action. As a consequence of its action, the agent receives feedback in the form of the next state and reward. Then, the agent modifies its policy to better align with the environment in order to maximize its reward. This process is repeated to ultimately obtain an optimized policy. The structure of reinforcement learning is shown in Figure 1 [16]. In this study, reinforcement learning algorithms were applied to train an agent to play match-3 puzzle games.

![Figure 1. Structure of reinforcement learning.](image)

### 2.2. Proximal Policy Optimization (PPO) Algorithm

The PPO algorithm, introduced by the Open AI team in 2017, is a notable member of the Policy Gradient (PG) techniques. PG techniques utilize artificial neural networks in reinforcement learning to directly compute the action values associated with optimal behaviors [17]. The core concept revolves around identifying the optimal policy that maximizes cumulative rewards by training an artificial neural network. This cumulative reward serves as the objective function to be optimized by the policy neural network with a focus on refining the weight values within the neural network. In simulation environments, PG-based reinforcement learning primarily concentrates on agent control. Numerous studies in the field of reinforcement learning have demonstrated that PG algorithms can facilitate quicker learning compared to traditional methods [18]. Moreover, a recently proposed actor-critic technique incorporates PG for actor learning to ensure rapid convergence. The PPO algorithm embodies an approach where an agent alternates between data sampling from its interactions with the environment and optimizing a surrogate objective function using stochastic gradient ascent [19]. In the realm of digital games, the PPO algorithm has shown impressive performance when compared to other reinforcement learning algorithms. For instance, efforts have been made to employ the PPO algorithm in generating reinforcement learning agent models for tower defense games [20]. In a virtual flight environment created using the Unity engine, the PPO algorithm demonstrated the agent’s ability to compute a flight trajectory reaching the target, even in the face of various changes in the external environment [21]. In this paper, the PPO algorithm was used to validate the effectiveness of the proposed system and its practical utility for optimizing stage construction and level balancing in match-3 puzzle games.

### 2.3. Unity 3D Engine

The Unity 3D Engine is a versatile game engine that empowers game developers to create a variety of content extending beyond just 2D or 3D games. Its capabilities include
animation, simulation, architecture, and more. One of the key strengths of Unity 3D Engine is its support for multi-platform play that enables deployment across mobile, PC, console, augmented reality and virtual reality devices. As a result, many games are developed and launched using Unity 3D Engine in the mobile game market. The engine's flexibility allows it to accommodate a wide range of game genres from those with straightforward controls to complex strategy games, physics-based puzzles, and multiplayer competitive experiences. Beyond gaming, Unity 3D Engine can be harnessed to rapidly develop simulation environments and extend its usage to various applications beyond the multi-platform capabilities previously mentioned [22–25].

2.4. Unity ML-Agents

Unity ML-Agents stand as a Software Development Kit (SDK) for machine learning designed to be integrated into projects developed with Unity 3D Engine. By utilizing the Unity ML SDK to configure machine learning, it is easier than other ML SDKs to use Deep Reinforcement Learning in games and simulations created with Unity Engine. It offers a Python API that enables swift configuration of the training environment for intelligent agents. The structure of Unity ML-Agents is depicted in Figure 2.

As shown in Figure 3, Unity ML-Agents are trained in the structured process. The agent selects an action, and in response, the environment offers feedback through a reward that can either increase or decrease the agent's reward, and the state of the learning environment. This architecture of Unity ML-Agents makes it possible to enhance the learning efficiency of the agent by initially providing rewards to encourage specific actions. Thus, the combination of Unity 3D Engine and Unity ML-Agents facilitates the seamless integration of machine learning capabilities into Unity projects. It enables developers to create intelligent agents that can learn and improve their performance through interactions with the environment.
Figure 3. Training process of Unity ML-Agents.

In Unity ML-Agents, the training environment is constructed by placing Game Objects and Components within a scene. This environment is designed based on either a Markov Decision Process or a Partially Observable Markov Decision Process, which forms the foundation of reinforcement learning tasks. Within this environment, observations, actions, and reward functions are defined and these can be specified within the agent’s components. It is worth noting that any structure with access to the agent’s components can manipulate the agent’s reward function and determine whether the agent reaches a concluding state.

Observations serve as elements that provide the agent with information about the state of the environment. They can take the form of vectors containing floating-point data types within the scene, referencing aspects like the number of cameras that output data. As a result, the components that make up an agent can define information corresponding to any number of observations.

Developers have the flexibility to define fixed or dynamic intervals in the environment. At these intervals, the agent can request brain decisions. Reward functions play a crucial role in providing learning cues to the agent and can be defined and modified during the simulation using the Unity Script System. The simulation can also reach a completion state either at the individual agent level or the entire environment level. This state can be activated via the Unity Script System or triggered when reaching a predefined maximum number of learning steps. When resetting the environment, developers have the ability to define reset environment parameters that can be accessed and manipulated from the Python API to gain further control over the learning process [23,25–28].

In this paper, the reason for utilizing Unity engine, Unity ML-Agents, and a connection to the Python API is their widespread adoption and easy accessibility among game developers. However, this does not imply a strict dependency on these platforms. The proposed algorithm testing system tailored to match-3 puzzle game was designed to be adaptable for implementation in other development tools or environments beyond the Unity engine. The system can be easily ported to other game engines since the codes can be reusable in different environments as long as the framework is supported. For instance, the proposed system utilizes a universal algorithm so it is independent of platform or system. Furthermore, most learning algorithms, including PPO and Soft Actor-Critic (SAC) that can be tested in the proposed system, are designed to work with a variety of environments. For the test, Unity 2020.1.15f1 and ML Release 10 was used.

3. Design and Implementation

3.1. Designing Match-3 Puzzle Game System

In this study, the match-3 puzzle game system was designed with six different block types on the board. Also, one general block and two special blocks were incorporated. Special blocks were defined as blocks that offer higher rewards compared to general blocks. This approach enables the agent to learn to prioritize acquiring special blocks. The
puzzle board is organized as an X×Y field, and the gameplay follows the typical match-3 puzzle game style with blocks being filled in from top to bottom.

Figure 4 illustrated a flowchart of the match-3 puzzle game system to be implemented. The designed match-3 puzzle game starts with a predetermined organization of cells on the board. The agent selects a cell and attempts a swipe movement to determine if it is a valid move. If the swipe movement is invalid, the agent selects another cell and tries again. However, if the swipe movement is valid, the matched cells are marked, and it is checked whether they are all of the same type. If the matched cells are not of the same type, the agent is prompted to select a different set of cells. On the other hand, if the matched cells are of the same type, they are processed accordingly. After the processing, a new cell is constructed at the empty location on the board. The process then checks for additional matching cells and repeats if they are of the same type. If none of the matching cells are of the same type, the process returns to cell selection. This game configuration ensures that the match-3 puzzle game follows the standard gameplay mechanics and provides a suitable environment for the machine learning algorithm to be evaluated and validated.

![Flowchart of the match-3 puzzle game](image)

**Figure 4.** Flowchart of the mach-3 puzzle game.

### 3.2. Implementing a Mach-3 Puzzle Game System

The proposed system in this paper was implemented based on the training configuration using Unity ML-Agents. As shown in Figure 5, the authoring screen provides an overview of the features offered by the Unity 3D Engine.
Figure 5. Unity 3D engines’ screenshot.

Based on the design, multiple game levels were implemented in a scene as shown in Figure 6. By creating multiple game levels within the scene, the developers can effectively conduct training sessions for a multitude of agents in a controlled and consistent environment. This approach facilitates the evaluation and comparison of different reinforcement learning algorithms and their performance in the match-3 puzzle game.

Figure 6. Screenshot of match-3 puzzle game implementation.
Figures 7 and 8, provided below, illustrate the class diagram of the match-3 game’s implementation. The implementation involves writing a class script that serves as a base class and inherits from additional classes to establish a connection between Unity ML-Agents and the match-3 game to facilitate the transfer of training data. As shown in Figure 7, the AbstractBoard class acts as a bridge between the ML-Agents and the match-3 game. In doing so, the ML-Agents perform the following tasks: (1) check a cell’s type value and whether it is a special block, (2) determine if the cell is movable, and (3) request the project to move the cell if it is movable. This is achieved by overriding the GetCellType(), IsMoveValid(), and MakeMove() functions as explained in Table 1. The AbstractBoard class keeps track of the number of rows, columns, and potential cell types on the board.

![Figure 7. Class diagram of match-3 puzzle game.](image7)

![Figure 8. Class diagram of match-3 puzzle game.](image8)
### Table 1. AbstractBoard class method.

<table>
<thead>
<tr>
<th>Function</th>
<th>Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetCellType()</td>
<td>- This function returns the cell types of the specified rows and columns on the game board.</td>
</tr>
<tr>
<td></td>
<td>- The returned value falls between 0 and a value containing NumCellTypes, i.e., 1.</td>
</tr>
<tr>
<td>GetSpecialType()</td>
<td>- When given the rows and columns of a cell, this function returns the special block types, if any.</td>
</tr>
<tr>
<td></td>
<td>- The value returned ranges between 0 and a value containing NumSpecialType.</td>
</tr>
<tr>
<td>IsMoveValid()</td>
<td>- This function is responsible for determining whether a particular move is allowed in the game.</td>
</tr>
<tr>
<td></td>
<td>- The validity of a move may differ based on the specific game rules. This function also includes the SimpleMoveValid() function, which handles basic match-3 puzzle game rules without special or unmovable pieces.</td>
</tr>
<tr>
<td>MakeMove()</td>
<td>- This function allows a given move to be executed.</td>
</tr>
<tr>
<td></td>
<td>- This function is considered true if the move is executed correctly.</td>
</tr>
<tr>
<td></td>
<td>- Be careful of requiring invalid moves during training.</td>
</tr>
<tr>
<td></td>
<td>- The move structure encapsulates the movement between two neighboring cells.</td>
</tr>
<tr>
<td></td>
<td>- Use Move.NumPotentialMoves(NumRows, NumColumns) to get the number of potential moves for a board of a certain size.</td>
</tr>
<tr>
<td></td>
<td>- These two functions can be implemented to create a new move.</td>
</tr>
<tr>
<td>FromMoveIndex()</td>
<td>- This function iterates over all potential moves on the board by iterating from 0 to the move. NumPotentialMoves() function</td>
</tr>
<tr>
<td>FromPositionAndDirection()</td>
<td>This function creates a move through rows, columns, directions, and board size.</td>
</tr>
</tbody>
</table>

As shown in Figure 8, the Match3Sensor class uses the AbstractBoard interface to create an observation board for the game’s state. Developers can choose between using the Vector Observation method or the Visual Observation method. In addition, the Match3SensorComponent class is responsible for creating the Match3Sensor class variable during runtime. By adding this class script to the same GameObject as the implemented agent script, its functionality becomes available.

In the game learning environment, the Match3Actuator class plays a crucial role in converting the learned or inferred behavior of the agent into a move, which is then used as an argument value passed to the AbstractBoard.MakeMove() function. For each potential move, the Match3Actuator class checks the validity of the move using the AbstractBoard.IsMoveValid() function and sets the agent’s behavior mask accordingly.

In this study, the AbstractBoard class was incorporated into the game learning environment through the creation of a Match3Board class that inherits the implementation from the AbstractBoard class. The Match3Board class enables the agent to receive reward values for specific actions such as matching multiple blocks in a row or matching blocks with special block types. This was achieved by adding a class script that inherits from the Agent and AbstractBoard classes alongside the Match3SensorComponent and Match3ActuatorComponent class scripts to the same GameObject. The agent was then programmed to call the Agent.RequestDecision() function when it is ready to move on to the next Academy Step, and the next Academy Step calls the board’s MakeMove() function.
Through this well-structured implementation, the game learning environment can efficiently integrate the AbstractBoard class and enable the agent to learn and respond effectively and acquire reward values based on its actions in the match-3 puzzle game.

3.3. Setting Up Machine Learning Style

In this study, the PPO algorithm was applied, and four different learning methods were employed to verify the feasibility of AI machine learning in the proposed match-3 puzzle game system. The following Table 2 describes the four learning methods.

Table 2. Four ways of machine learning algorithm.

<table>
<thead>
<tr>
<th>Machine Learning</th>
<th>Method</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Observation</td>
<td>Vector Observation</td>
<td>Determines the next action based on vectors</td>
</tr>
<tr>
<td></td>
<td>Visual Observation</td>
<td>Determines the next action based on visual elements such as color</td>
</tr>
<tr>
<td>Heuristic</td>
<td>Simple Heuristic</td>
<td>Imitation learning where the agent moves with code that reflects human intuition</td>
</tr>
<tr>
<td></td>
<td>Greedy Heuristic</td>
<td>Learning by doing where human decisions are applied with greater precision and granularity to reduce meaningless actions as much as possible</td>
</tr>
</tbody>
</table>

Given the complexity of match-3 puzzle games, randomly acting agents find it challenging to obtain rewards and makes it difficult to approximate appropriate agent behavior through the observation method alone. To overcome this, this study employed a heuristic method for learning of the agent. In order to gain comprehensive comparison results, a total of four learning methods were employed.

4. Results

To verify the effectiveness of applying the proposed match-3 game system in designing stage levels, this paper examined the training outcomes recorded on TensorBoard through the utilization of the PPO algorithm. The implementation of AI-driven level balancing and stage construction may demand heightened computational resources. Thus, this study recommended investigating its effect on game performance and determining its feasibility for typical end-user devices. The use of TensorBoard and visualization of results serve the purpose of simplifying the complexity of the training process. In order to monitor the results generated by the ML-Agent during training within the Unity Editor environment, the Python environment was appropriately configured.

Figure 9 shows the experiment results obtained using PPO algorithm with four different methods as displayed on the TensorBoard. By specifying appropriate parameter values and the number of training runs, it is confirmed that the system can be executed in various ways to derive training outcomes. These results are subsequently utilized for the purpose of comparing and analyzing the performance of each algorithm. One example is shown in Figures 10 and 11.
In Figure 10, the color palette is employed to represent each of the four learning methods as they appear in the cumulative reward graph when using the PPO algorithm. The first color corresponds to the Vector Observation learning method (blue), the second to the Simple Heuristic learning method (yellow), the third to the Visual Observation learning method (white), and the fourth to the Greedy Heuristic learning method (purple). The above mentioned clear, four-colored lines (blue, yellow, white, and purple) in the graph represent smoothed values, while the faint lines surrounding them depict the actual measured data values. Smoothing refers to the process or technique of removing noise from data to enhance the understanding and analysis of trends, patterns, and other data characteristics. It aids in more effective data analysis to facilitate the identification of key patterns and trends and improving predictive models.

On the other hand, Figure 11 displays the graphs of the four learning methods utilizing the PPO algorithm with the entropy being presented as the training progresses. Similar to the Figure 10, each line in a different color represent four distinct learning methods.
In the cumulative reward graph within the learning results, the agent’s learning efficiency across training iterations is measured. Higher values on this graph indicated superior learning performance. The entropy graph depicted entropy values for each training run where lower values signify greater stability in the learning outcomes. Therefore, it is confirmed the system can be effectively used to compare learning outcomes for each algorithm. Moreover, it enables the determination of the optimal algorithm for match-3 puzzle games by comparing the cumulative rewards and entropy values associated with each algorithm.

5. Conclusions

With the continuous advancement of machine learning technology, AI-based games have emerged in various game genres. However, the match-3 puzzle game genre, despite its large user base and ongoing development, has not fully embraced machine learning for efficient game development. Many developers still invest significant time and effort in manual level balancing tests while the potential for using machine learning to streamline game development remains largely untapped.

To address this issue and reduce the labor cost and time required for level balancing and verification for stage updates, this study developed a match-3 puzzle game system. This system allows for the testing and identification of the most effective reinforcement learning algorithm for training AI agents by comparing their machine learning results. To validate the proposed system’s usefulness, this paper conducted experiments with the PPL algorithm and received cumulative reward and entropy result graphs. Thus, it is confirmed the system can be utilized to compare learning outcomes for each algorithm and identify the optimal algorithm that is suitable for match-3 puzzle games.

Further studies will continue to compare different algorithms including PPO and SAC algorithms as well as different machine learning methods within each algorithm. In addition, while this study did not include variables such as items, stage missions, and special blocks that introduce randomness of match-3 puzzle games, further study can extend to incorporate these elements of randomness and complexity in the training environment. This expansion will enable more precise level balancing and difficulty tests.

Overall, this study expects to take a meaningful step towards unlocking the full potential of machine learning in the match-3 puzzle game genre and streamlining game development practices.

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-202009460010).

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


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.