International Research Journal of Multidisciplinary Scope (IRJMS), 2024; 5(4):730-745 IRJMS

Review Article | ISSN (O): 2582-631X DOI: 10.47857/irjms.2024.v05i04.01355

A Comprehensive Review of Reinforcement Learning Applications in Gaming

Ashish Kumar, Jasleen Kaur Bains*

Department of Computer Science and Applications, Panjab University, Chandigarh, India. *Corresponding Author's Email: jasleen@pu.ac.in

Abstract

Reinforcement learning is one of the most popular models for building agents that deal with the real world but are not distinctly told which actions to perform. In the context of gaming, the application of reinforcement learning thus spans many different categories, from classic arcade games to modern simulations. The aim of this review paper is to present a comprehensive review of reinforcement learning in gaming, its core methods or algorithms, and the results obtained. This paper first lays the groundwork by discussing the basics of reinforcement learning, among which are agents, environments, rewards, and policies. Then we discussed the mathematical framework of reinforcement learning, the Markov decision process, and the Bellman equation. Thereafter, it discusses specific reinforcement learning algorithms that have been successfully implemented in video games. A variety of algorithms has been adapted from the field of reinforcement learning and has shown huge success, like Q-learning, deep Q-networks, and policy gradients. Then we compare the different games and algorithms associated with them and their outcomes. There are also some major challenges with RL in video games, such as computational complexity, environment design, and many more. Finally, the conclusion and future aspects of applications of reinforcement learning in video games are discussed.

Keywords: Artificial Intelligence, Gaming, Machine Learning, Reinforcement Learning, Video Game.

Introduction

Artificial intelligence (AI) is a technology enabling computers and machines to mimic human intelligence and problem-solving skills. John McCarthy coined the term 'Artificial Intelligence' in 1956, but the foundational principles were initially developed in 1950 by Alan Turing (1). AI finds numerous applications, with video games being a prominent domain. It is very common for video games to have intelligent agents, which makes them more entertaining and challenging for their players. The early Atari games, such as Pong and Space Invaders, were among the first to feature AI. Video games provide a safe and cost-effective environment for AI research due to their controllable virtual settings (2). Reinforcement learning (RL) is one of the best ways to implement AI in video games. Arthur Samuel is the first person to use RL to develop a program to play checkers that learns from its experience (3). In recent years, RL has been applied to Google Deep Mind's Alpha Go and Open AI Five. Alpha Go achieved a historic victory against the world champion Go player, Lee Sedol (4), while Open AI Five secured consecutive wins against Dota 2 World Champions in 2019 (5). The objective of this study is to provide an overview of the application of RL techniques within the context of video games and to identify the significant challenges associated with their integration. This study aims to lay the groundwork for future research by offering a fundamental understanding of RL principles and their applications in gaming. The paper offers an in-depth examination of various RL techniques, including Q-learning, deep Q-networks, and policy gradients, and their implementation in video games. Understanding these techniques will reveal how RL algorithms are designed to address complex problems in dynamic and interactive environments. This paper has discussed several RL techniques, including Q-learning, deep Q-networks, policy gradients, and applications to video games. If understood, these techniques give the reader insight into the design of RL algorithms to solve complicated issues in dynamic and interactive environments. This paper discusses the

(Received 18th June 2024; Accepted 21st October 2024; Published 30th October 2024)

fundamentals of RL and popular algorithms applied in video games. It examines several games in which popular RL techniques were used, providing examples of their outcomes. Further it addresses the challenges faced by RL techniques. Finally, the paper concludes with a discussion of future research directions and potential developments. Over the past few decades, RL has advanced significantly, particularly in its application to video games. A pivotal work in this field was Sutton and Barto's 1998 publication, Reinforcement Learning: An Introduction (6), which introduced the temporal difference (TD) learning method. The TD approach integrated ideas from supervised learning and dynamic programming. It allowed an agent to learn predictions of future rewards without requiring a model of the environment. TD learning's success paved the way for further breakthroughs in RL; it promised great potential in both theoretical and practical domains, including gaming. Building upon this foundation, one of the most used model-free RL algorithms introduced was Q-learning by Watkins and Dayan in 1992 (7), The off-policy nature of Q-learning provides the ability to update policies relative to the optimal actions instead of the ones that have been taken by the agent, making Q-learning very appropriate for complex gaming environments. The versatility of Q-learning was later demonstrated in its application to classic arcade games such as Snake and Pong, where agents learned the optimal action in a trial-and-error process. 2013 saw the development of Deep Q-Networks (DQN) by Mnih *et al*. (8), where DQN represented a gigantic step forward in bringing together Q-learning and deep neural networks. Thereby, RL algorithms were given the capacity to deal with high-dimensional state spaces that usually appear in Atari games. DQN agents demonstrated superhuman performance in several games; the shift marked the importance of dominance in AI-based game strategies by RL. Notably, DQN addressed the stability issues faced by traditional Q-learning through techniques like experience replay and target networks, significantly enhancing the training efficiency. Another turning point for RL in gaming application was reached in 2016 by Alpha Go (9), developed by Google's Deep Mind. Alpha Go conquered human Go professionals by using Monte Carlo tree search paired with deep RL techniques, demonstrating that RL can be powerful enough to solve highly strategic and complex tasks in real-time, multiplayer environments. This achievement became the milestone for future breakthroughs in other games, such as Dota 2 and StarCraft II, where RL agents like OpenAI Five and AlphaStar, beat human champions (10, 11). These developments highlight RL's evolution from early theoretical frameworks to practical, high-impact applications in gaming. Since policy gradient methods and Proximal Policy Optimization (PPO, 12), have been proposed, RL has found its application in the most sophisticated strategies involving real-time strategy games, offering a flexible and scalable manner of handling dynamic multi-agent environments. Sutton and Barto (6), in their foundational work in 1998, Watkins and Dayan in 1992 (7), Mnih *et al*., in 2013 (8), as well as Silver *et al*. in 2016 (13), introduced core algorithms currently inherent to RL and proved the former's applicability to playing games. Each successive development has made RL capable of dealing with more sophisticated game environments-from the simple, dumb arcade games early on to the rather sophisticated real-time strategy games that demand a high sense of decision-making ability and flexibility. These works are the backbone of success in RL gaming, and their on-going influence continues to mold the application of RL in modern video game AI.

Comparison of RL with other Approaches to AI in Gaming

As artificial intelligence keeps developing and getting better for games, many approaches must be used, including supervised learning, unsupervised learning, and reinforcement learning, each with different advantages and disadvantages. Supervised and unsupervised learning are based on predefined data, but RL stands out because it learns by continuous trial and error. This section addresses how RL compares to these AI approaches, showing how interaction-based learning particularly suits the complexity of game environments and their many dynamic changes (Figure 1).

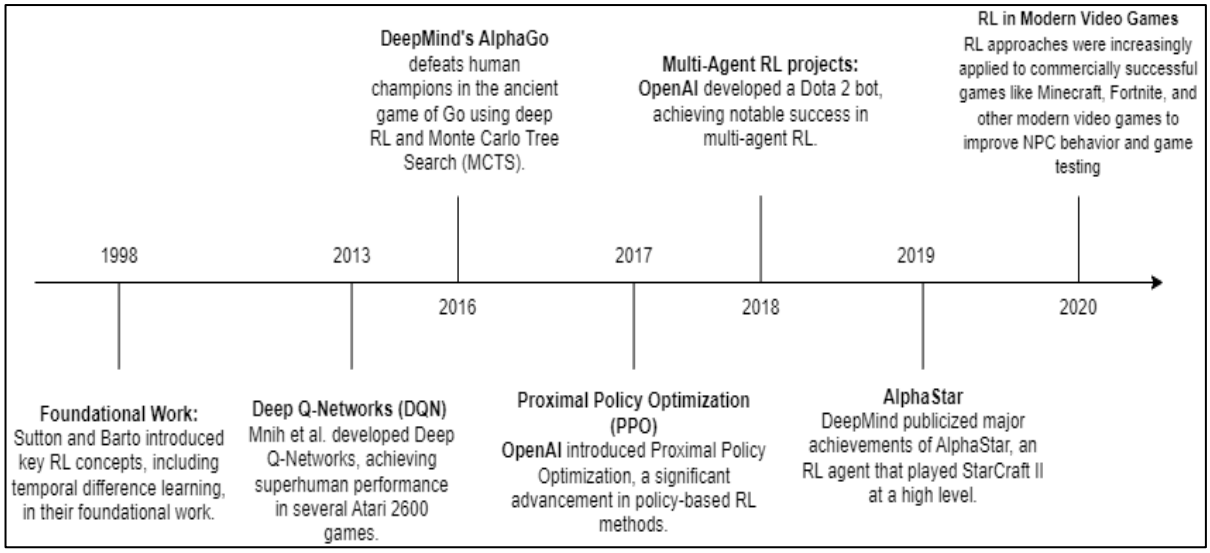

Figure 1: Brief History of Reinforcement Learning

Supervised Learning

Supervised learning builds models on labelled data. In this, the output to be obtained given an input is known in advance (14). Error is minimized, and that's how it learns. This really proves to be efficient with tasks such as classification, object detection, and pattern recognition -that is frequently used in the game to recognize avatars of the player or NPCs. For example, such facial recognition in games can rely on supervised learning in order to determine the difference between the player and the NPC (Figure 2). However, supervised learning is not sufficient in changing game environments that demand real-time decisions as it cannot adapt without retraining to new cases or situations. In games like StarCraft II (10), where strategies evolve dynamically, supervised learning becomes impractical because it depends on fixed datasets. If the game's strategy changes, the model must be retrained, making supervised learning less adaptable compared to RL, which continuously updates its strategy through interaction with the game environment.

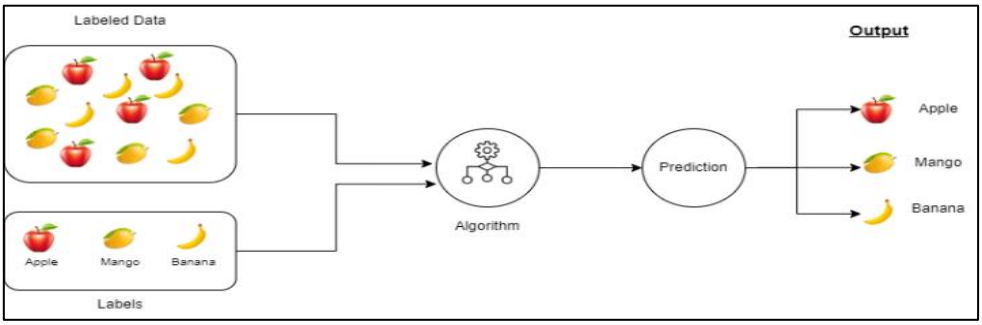

Figure 2: Supervised Learning

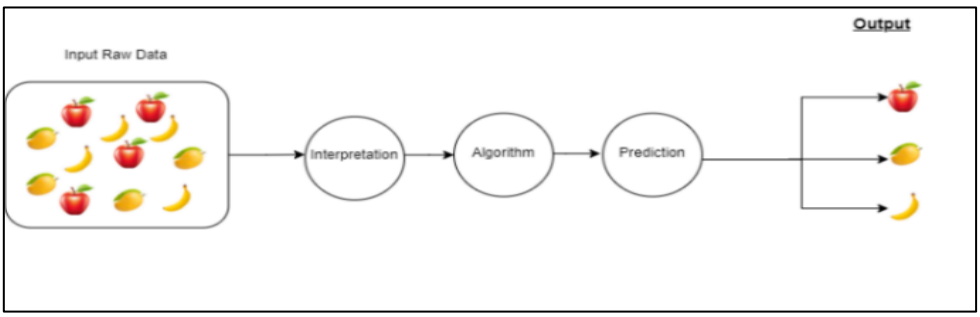

Figure 3: Unsupervised Learning

Unsupervised Learning

Unsupervised learning deals with unlabelled data, clustering similar data points to discover hidden patterns or structures (14). It is very handy for tasks like procedural content generation, in which game levels or worlds are created without any human intervention being as direct as possible, so that there is variability and uniqueness in gameplay. For example, the procedurally generated worlds in No Man's Sky (15) are produced through techniques of unsupervised learning. The algorithm can generate infinite worlds based on the patterns in the underlying data, giving a broad and varied player experience. But, in real-time, interactive games that entail decision-making, unsupervised learning is limited because it does not contain a reward-based feedback loop. In contrast, RL continuously increases the performance of the agents as it learns from rewards and penalties. In scenarios involving gameplay where the environmental and strategic conditions change, this makes it relatively more potent (Figure 3).

Reinforcement Learning

Reinforcement learning differs fundamentally from other AI approaches in that agents can learn, interacting with their environment to adapt their strategies over time (16). All agents receive rewards or penalties depending on actions taken, which helps agents update their strategies to maximize cumulative rewards. Because of this property, reinforcement learning is suited to complex, dynamic environments where real-time adaptations are necessary. For example, AlphaGo (13) was an RL agent that learned to play Go within millions of simulations and kept improving through trial and error. Furthermore, since supervised learning depends on labelled data, an RL-trained agent such as AlphaGo managed to come up with innovative strategies not explicitly programmed or trained against particular data. This learning process through trials and errors together with the real-time environment in RL provides the latter a competitive advantage in AI methods used for gaming applications (Figure 4).

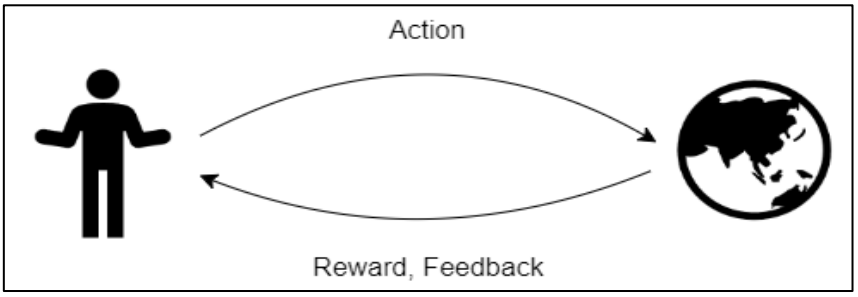

Figure 4: Reinforcement Learning

Reinforcement learning thus has the power to adapt to changing environments, making it very effective in games nowadays. Supervised and unsupervised learning play their roles especially in classification and content generation, but it lacks real-time adaptability in complex games. The dynamic process of RL, based on rewards and penalties, makes it efficient in complex interactive games where strategy evolves continuously.

Fundamentals of RL

Reinforcement Learning is an ML technique that allows an agent to learn in an uncharted environment through trial and error, using feedback from its actions. The main components of an RL system are the agent and the environment, the state, the action, the policy, a reward signal, the value function and the model (6, 17). Figure 5 shows the working of RL. Agent: The person being trained. In GTA, the agent would be either a player or an AI-controlled character. Environment: this refers to the surroundings where the agent is located. It is the area in which the agent performs some actions and gets rewards or punishment as feedback for positive and negative behaviour. In GTA, the environment is the virtual city and its inhabitants. The state represents the current situation in the environment. In GTA, it could be the agent's position, nearby cars, available missions and the current wanted level. Action refers to the decision or move that the agent can make. In GTA, actions could include walking, running, driving, shooting, or starting a mission. The policy is the way an agent acts within the environment and learns to behave in the environment under given circumstances. In simple words, the policy maps the states of action. The policy can be a function

or even some lookup table. The policy is considered as a core of RL agents because it alone is enough to determine agents' behaviour (6, 17, 18). In GTA, if the agent's state indicates that a mission is nearby, then the policy might prioritize moving towards the mission starting point. A reward signal indicates how favourable or unfavourable an event is. It defines the goal of a RL problem. At every event, the environment sends a signal to the agent based on the actions performed in that particular event. That signal is known as a reward signal. The agent's only objective is to maximize the total reward in the long run. In a biological system, the reward signal is similar to the experience of pleasure or pain. If some action generates low or negative reward then the policy will shift to some other action which will generate positive or non-negative reward. In GTA, rewards can be money earned from missions, reputation points, or even penalties like a wanted level for breaking the law. The value function is the future reward that an agent would be able to get by executing any particular action in a certain state. Thus, it tells the evaluation of the states and, according to this evaluation, the optimal action. In simple terms, the value function estimates in what state the agent will be given and how good a certain state is in terms of future rewards. In GTA, for example, proximity to a mission start would be of high value due to the possible reward from the successful mission completion. In everyday terms, rewards are like pleasure and pain, while value corresponds to how pleased or dissatisfied you are in some particular state.

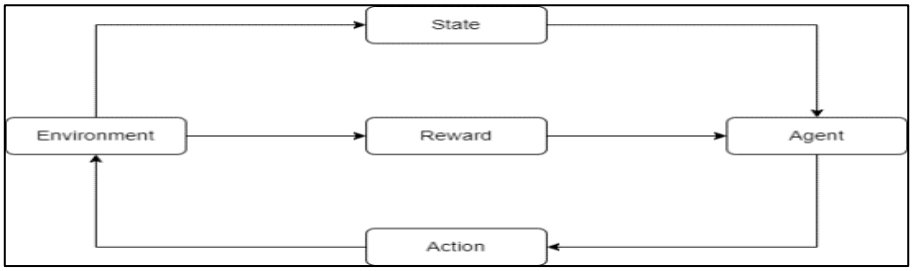

Figure 5: Working of RL

The final component of RL systems is a **model** of the environment. This model allows us to mimic the behaviour of the environment or predict how the environment will respond to certain actions. However, the model is an optional component of the RL system. In GTA, a model could simulate a driving scenario to predict the outcome of actions such as turning at an intersection or braking, which might help the agent evade the police or complete a mission quickly. In RL, an agent needs to learn to take appropriate actions in unknown environments in order to maximize the overall return. For this, a highly formal mathematical framework is needed to model the interaction between the agent and the environment. In doing so, an appropriate formulation and solution of RL problems can be realized in the context of a Markov Decision Process.

Markov Decision Process

Markov decision Process (MDP) is a mathematical framework that is used to study decision-making processes between an agent and its environment, where outcomes depend on the actions taken and some element of randomness. MDP aims at providing the mapping of best actions for each condition of an environment. MDP is based on Markovian property, which only takes into account the present state and ignores information from the past state. Future state prediction is entirely unaffected by the prior state. The physical objects in the surroundings remain unchanged, and the laws of physics remain constant (19). Chess is one of the games where the rules don't change and you don't have to recall your previous moves to play the next one. The five-tuple (S, A, P, R, ϒ) can be used to characterize the Markov decision process (20).

States(S)

The S represents all possible states of the system. A state is a detailed description of the environment at a given time. In GTA, a state can be described by the current location of the player, their health level, the current amount of money they have, and the presence of non-player characters (NPCs) around them.

Actions

The set A represents all the possible actions that the decision-maker or agent can take. An action influences the transition from one state to another. In GTA, actions could be driving to a new location, engaging in combat, interacting with NPCs, and performing missions.

Transition Probability (P)

This function describes the probability of transitioning from one state to another based on the action performed by the agent. It is denoted as P (s, a, s'), where s is the current state, a is the action performed, and s' is the next state. In GTA, if the player chooses to drive from point A to point B, then the transition function represents the probability of arriving at point B without dying or encountering police. Figure 6 depicts the state transition diagram.

Reward Function (R)

This specifies the reward received by the agent after transitioning from the state (s) to (s') due to action a. In GTA, this reward could be money earned from a mission, losing health or earning a bad reputation for not completing objectives.

Discount factor (ϒ)

The discount factor signifies the difference between the present and future returns, implying that the present reward holds greater significance than the future reward. In GTA, the discount factor might prioritize immediate rewards such as quickly gaining money and health over long-term incentives.

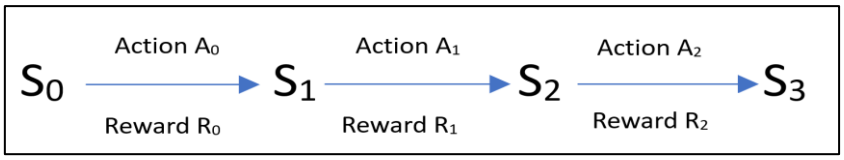

Figure 6: State Transition Diagram

Bellman Equation

When it comes to solving MDP and RL problems, Richard Bellman's renowned Bellman equation, which he created in 1953, is of utmost significance. The value function can be computed using the Bellman equation. In RL, the Bellman equation connects the value of a state to the expected value of future states (21). It decomposes the value of a state into immediate rewards and the discounted value of successor states under a certain policy. The Bellman equations involve two types of value functions:

State Value Function (V_π (s)): This represents the expected return (total discounted rewards) starting from the state (s) and following a particular policy (π).

Action Value Function (Q_π (s, a)): This represents the expected return starting from state (s), taking action (a), and thereafter following a policy (π).

Bellman equation for State Value function (7):

$V_\pi(s) = \sum_a \pi(s|a) \sum_{s'} P^a_{ss'} [r(s,a) + \gamma V_\pi(s')]$ [1]

$V_\pi(s)$ in equation 1 represents the State value function, where (s) is the state and the policy (π). π (s|a) is the probability of taking action (a) in the state (s) under policy (π). $P^a_{ss'}$ is the probability of transitioning from state (s) to state (s') after taking action (a). r (s,a) is the immediate reward. ϒ is the discount factor which balances the importance of immediate and future rewards.

Bellman equation for Action Value function (19):

$Q_\pi(s,a) = \sum_{s'} P^a_{ss'} [r(s,a) + \gamma \sum_{a'} \pi(s') Q_\pi(s',a')]$ [2]

$Q_\pi(s,a)$ in equation 2 represents the action value function, where (s) is state and (a). $Q_\pi(s',a')$ represents the action value function, where (s') is state and (a'). π (s'|a') is the probability of taking action (a') in state (s') under policy (π). $P^a_{ss'}$ is the probability of transitioning to state (s') from state (s) after taking action (a). r (s,a) is the immediate reward. ϒ is the discount factor which balances the importance of immediate and future rewards.

RL Algorithms Used in Video Games

RL algorithms have recently opened up a new era of artificial intelligence in video games. While current approaches to classical AI in games involve the creation of predefined behaviors and rules, reinforcement learning allows game characters and agents to learn and adapt by interacting with their surroundings. This dynamic technique ensures not only realism but also unpredictability among non-player characters (NPCs), leading to even more complex and demanding game aspects. This section looks at the use of RL algorithms in video games. But before that, let us understand some key terms:

- **Model-Free RL**
  Model-Free RL algorithm relies not on a model of the environment; instead, it learns a policy or value function directly from interactions with the environment. Two well-known kinds of model-free algorithms are Q-learning and policy gradient methods.
- **Model-Based RL**
  Model-based RL algorithms, on the other hand, typically obtain the dynamics. They require learning a model of the dynamic environment, predicting the next state and the reward given the present state and action. The agent uses this model to plan its actions, which are the predictions of future states, and selects behaviours in order to maximize expected rewards. One example is Dyna-Q, a class of algorithms that combines direct learning with model-based planning.
- **On-policy**
  Algorithms evaluate an existing policy through and through. They demand that the policy needs to be updated according to the actions that have been taken by the agent. For example, the SARSA algorithm updates the action-value function according to the actions of the current policy, thus making sure learning is on par with the experience of the agent.
- **Off-policy**
  Algorithms involve analysing or improving policies other than those used to generate data. This allows higher flexibility: they can even learn from activities falling outside the present policy. A classic example of an off-policy strategy is Q-learning, where the agent learns about the optimal policy even while pursuing a different policy for exploration.

Q-learning

Q-learning is a model-free, off-policy approach for learning long-term optimal behaviour (7). It focuses on developing a policy that maximizes long-term benefits by learning to react optimally in an environment. Despite its simplicity, Q-learning is effective in complicated environments with discrete state and action spaces, making it useful in a variety of disciplines, including gaming, robotics, and finance (22). This algorithm learns the action value function rather than the state value function (23), which is defined as

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \ _{a' \in A} Q(s_{t+1}, a') - \ Q(s_t, a_t)] \quad [3]$$

In equation 3, the learned action-value function Q is directly approximated to the optimal action-value function, regardless of the policy used (23). Q-learning, a reinforcement learning algorithm has been used in several video games. One noteworthy example is the classic game Snake.

In the Snake game, the player controls a snake that moves around the screen to eat food (represented by dots, 24). The snake's length increases as it consumes food. The snake's goal is to avoid hitting the walls or its own tail. Q-learning teaches the snake agent how to play the game by rewarding good behaviour and penalizing bad behaviour. The agent discovers which activities, like approaching food, result in favourable rewards and which actions, such as leaving the screen, result in undesirable outcomes. At each step of the game, the agent chooses the action with the highest expected reward based on its current state.

State-Action-Reward-State-Action

The goal of the state-action-reward-state-action (SARSA) on-policy method is to tackle RL problems by teaching a machine learning model a new Markov decision process policy (25). According to the algorithm, the agent takes an action (A) in the current state (S), receives a reward (R), moves on to the next state (S1), and then takes another action (A1) in S1. Consequently, the abbreviation SARSA is represented by the tuple (S, A, R, S1, A1). The algorithm is known as an "on-policy" algorithm since it modifies the policy in response to activities (6). In comparison to Q-learning, the SARSA algorithm is a model-free algorithm, and in on-policy learning, it evaluates and follows a single policy. SARSA may converge slower than Q-learning since it updates based on policy actions rather than optimal actions. SARSA is an on-policy reinforcement learning algorithm that balances exploration and exploitation. One of the best games where SARSA is applied is the Cartpole Game (26). In the Cartpole game from OpenAI Gym, the goal is to balance a pole on a moving cart. The agent must learn to control the cart's movements to prevent the pole from falling. Using Deep SARSA (a variant of SARSA with deep neural networks), the agent learns an effective policy for balancing the pole. The trained agent becomes

proficient at maintaining balance and preventing the pole from falling off. This demonstrates how SARSA can be used to optimize decision-making in a simple game like Cartpole.

Monte Carlo Method

Monte Carlo methods in RL do not assume complete knowledge of the model or the environment. Rather, it learns from its own experiences as it interacts with the environment either real or simulated. With the simulation of an environment, some model information is needed but quite less detailed than in methods like dynamic programming because, under dynamic programming, one needs the probabilities of every transition. Monte Carlo methods depend on averaging complete returns and are usually employed in episodic tasks. This implies that the value of a state is estimated, and the policy is updated only after an episode is completed. Just like all the other RL methods, we shall first look at the prediction problem for the Monte Carlo methods and then see how it can be used in solving the control problem. Prediction is nothing but estimating the values of states, which indicates how much useful it is for the agent to be in a particular state. The higher the value of a state, the better it is for the agent. And the control problem is that of finding the best policies. A specific Tic Tac Toe game instance is being played by an AI based on the Monte Carlo method (27), which is playing random variations of the game in the background. For each game outcome, the AI scores the outcome. Then, it picks the move that most influences the selection for achieving a win. In this way, the Monte Carlo method enables the AI to learn the most effective strategy without an exhaustive search. It is efficient and effective.

Policy Gradient Algorithm

It aims to search for the best policy that will maximize the expected return by directly modifying policies through policy iteration. Such algorithms are referred to as model-free because they do not assume prior knowledge about the model of the environment. In simpler terms, we do not know the environmental dynamics or the transition probabilities (28). This can be seen as the likelihood of moving to the next state (S') by performing an action from the current state (S). Transition probability and policy are sometimes mixed up. Distribution of actions provided to states is known as policy (π). Put simply, the policy determines how the agent acts, while the transition probability explains how the environment changes, which is often unknown in real-world scenarios. The Policy Gradient algorithm has been applied to different video games to improve decision-making by training the agents to learn optimal policies. Pong, of course, is a classic video game where the player controls a paddle in order to hit a ball back and forth. Policy Gradient was used in the Pong game optimization by researchers (29). The way the agent played Pong was learned from the agent, which was being controlled by the algorithm by modifying the movements of its paddle. This kind of research is also done by researchers in various other ATARI games, including Breakout, Pong, and Space Invaders. The trained agents have produced very good results and have outperformed human players in some games. For example, in Breakout, the agent learns how to break the bricks efficiently in order to maximize its score. The result underlines how powerful PG methods are to master complex video games.

Deep Q-Network Algorithm

The Deep Q-Network algorithm integrates training deep neural networks with RL. DQN operates directly on raw visual input, as seen in the Atari 2600 Games (8, 25). DQN, developed by DeepMind researchers, integrate deep neural networks with RL algorithms to provide a ground-breaking answer to the complex challenges posed by high-dimensional state and action spaces. This invention has significantly advanced the profession by enabling efficient learning and decision-making under complex contexts (18). DQN overcomes the basic instability problem using two different practices, which include experience replay and target network. The fundamental idea used by DQN is to implement a deep neural network to take over the Q-function. The DQN algorithm implemented and optimized the action-value functions using a deep neural network-based approach. Below is an outline of the working procedure:

Step 1: State Representation

An input function that assigns a suitable numerical representation to the current state of the environment, either raw pixel values or previously pre-processed characteristics. The deep neural network should be of a particular design, most often a convolutional neural network

(CNN) that takes up the state of nature as input and outputs the action values for all possible actions that can be taken. This is termed the neural network architecture.

Step 2: Experience Replay

A replay memory buffer is a collection of the experiences of the agent, which consists of states, actions, rewards, and the next states from which the tuple is formed.

Step 3: The Q-Learning Update

Use small-sized mini-batches of experienced samples drawn from the replay memory buffer to update the weights of the neural network. A loss function for the Bellman equation between the goal and projected action values minimize the loss to achieve the update.

Step 4: Exploration and Exploitation

Actions to be taken either stochastically to explore the surroundings or greedily based on the current policy to achieve the balance between exploration and exploitation

Step 5: Target Network

Use an alternative target network of the same architecture for the design implemented to stabilize the learning. Update the target network periodically by replacing the weight of the main network with an equivalent copy of the weights. Now repeat all the steps: engage with the environment, collect data, update the network, and iteratively improve the policy until convergence. Researchers used DQN to play the classic arcade game, Pac-Man (18). The agent learned to negotiate the maze, evade ghosts, and gather pellets. After training, the DQN agent was able to reach human performance levels, thus proving that it indeed could learn the best policies for playing Pac-Man. It even devised gameplay strategies that topped human baselines. A number of other Atari 2600 games, from Space Invaders to Breakout, have also had DQN applied to them.

RL in Video Games

In game development, on the other hand, RL has been greatly hyped for its promising game AI revolution and potential betterment in the player experience. This paper discusses the core RL methods, focusing on their application to video games. A detailed overview of RL fundamentals, including agents, environments, policies, and rewards, is provided in the 'Fundamentals of RL' section. This section tabulates and compares these implementations in a selection of video games as shown in Table 1.

Table 1: Shows an Overview of Selected Video Games, Their Associated RL Algorithms, and the Results from Running the RL Algorithms

Category	Game	RL Algorithm	Speed	Accuracy	Complexity	Outcomes
Strategy and Board Games	AlphaGo (9, 13)	MCTS with deep neural networks	High computational cost, slow training	Superhuman-level accuracy	High; requires sophisticated strategy and planning	Defeated top human Go players, showcasing superior strategic thinking and learning from self-play.
	Chess (9)	AlphaZero (Combines MCTS and Deep Learning)	Moderate to high speed with self-play	Superhuman-level accuracy	Moderate; strategic planning and foresight	AlphaZero defeated top human and computer players, mastering the game through self-play.

Real-Time Strategy Games	Dota 2 (11)	Proximal Policy Optimization (PPO)	Moderate training speed	High accuracy in real-time strategy	Very high; multi-agent coordination	OpenAI Five defeated professional human teams, demonstrating advanced real-time strategy and coordination.
	StarCraft II (10)	Deep Q-Learning, Supervised Learning	Slow training due to complex environment	High accuracy with strategic depth	Very high; long-term planning required	AlphaStar outperformed top human players, managing complex long-term strategies in a real-time setting.
Classic Arcade Games	Ms. Pac-Man (30)	Monte Carlo Tree Search (MCTS)	Moderate speed due to tree search	High accuracy in-game navigation	Moderate; non-deterministic elements	Achieved high scores by effectively managing non-deterministic elements.
	Atari Games (8, 14)	Deep Q-Network (DQN)	Moderate training speed	High accuracy across multiple games	Moderate; varying game mechanics	Achieved human-level performance across various Atari games by learning game mechanics autonomously.
Platformer Games	Super Mario Bros. (31)	NEAT (Neuro Evolution of Augmenting Topologies)	Slow due to the evolutionary process	High accuracy in level completion	High; requires evolving neural network topology	Developed specialized models capable of autonomously navigating complex game levels.

First-Person Shooter (FPS) Games	Quake III Arena (Capture the Flag) (32)	Deep Q-Learning	Moderate training speed	High accuracy in multi-agent scenarios	High; 3D environment and team strategy	Agents developed by DeepMind exhibited human-level performance in team-based first-person shooter games.
Multi-Agent and Cooperative/Competitive Games	FIFA Soccer (33)	Multi-Agent RL, Imitation Learning	Moderate training speed	High accuracy in teamwork and tactics	High; team-based tactics and strategies	Agents developed by Google Research demonstrated advanced playmaking abilities and teamwork.
	League of Legends (34)	Hierarchical RL	Slow due to multi-level decision-making	High potential accuracy in team strategies	Very high; complex team strategies and roles	Research in progress, with the potential to manage complex team strategies and role-specific actions.
Simulation and Robotics	Minecraft (35)	Asynchronous Advantage Actor-Critic (A3C)	Moderate to high speed with parallel training	High accuracy in task execution	High; open-ended environment and tasks	Trained agents to perform complex tasks and navigate open-ended environments.
	Gran Turismo (36)	Soft Actor-Critic (SAC)	Moderate to high-speed	High accuracy in driving simulation	High; continuous action spaces	AI developed by Sony AI achieved expert-level performance in racing, handling complex car dynamics.

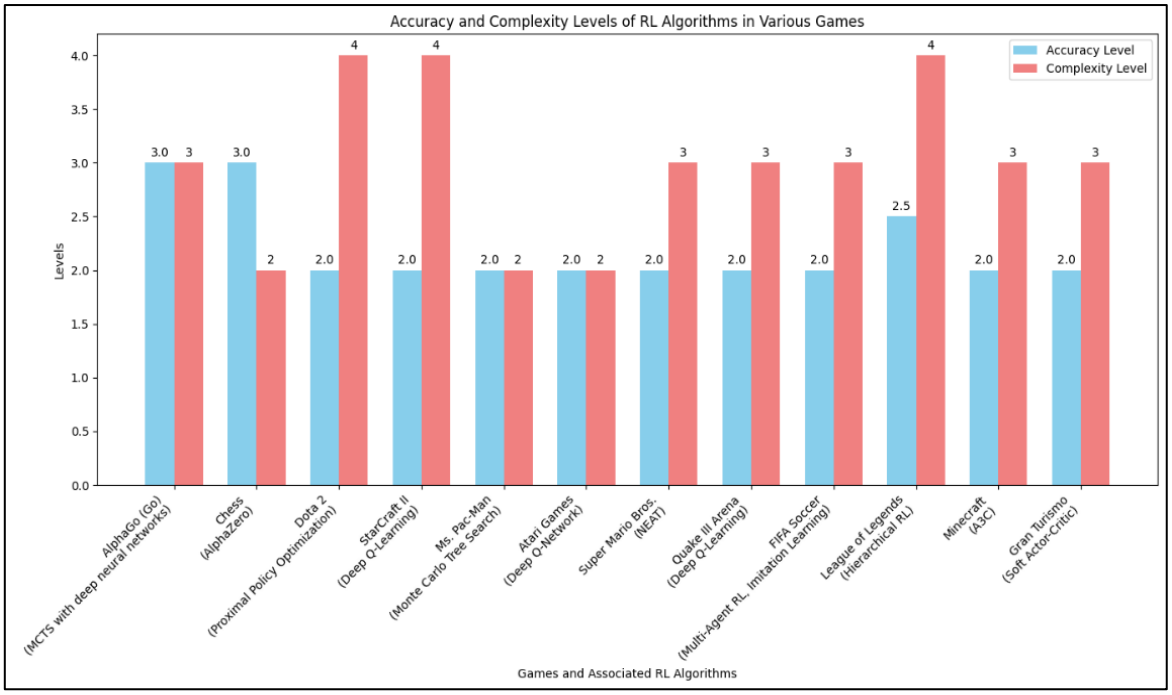

Figure 7: Accuracy and Complexity Levels of RL Algorithms in Various Games

Figure 7 shows how different RL algorithms perform in terms of accuracy and complexity when used in a range of video games. The X-axis displays every game with its RL algorithm, with the Y-axis indicating the levels of accuracy and complexity. The accuracy of RL algorithms is shown by the blue bars, and their complexity levels are represented by the red bars. Games such as AlphaGo and Chess achieved accuracy at a superhuman level by utilizing reinforcement learning algorithms, which demand high levels of strategic planning and precision. In contrast, Ms. Pac-Man and Atari Games were successful due to their high precision despite being moderately complex, requiring navigation through predictable environments. In games such as Dota 2 and StarCraft II, the level of complexity ranges from high to very high due to the importance of real-time strategy and coordination, yet the RL algorithms still achieved high accuracy despite the intricate gameplay. Super Mario Bros. and FIFA Soccer achieved high accuracy with moderate complexity, while Minecraft and Gran Turismo showed high accuracy with very high complexity because of their open-ended environments and continuous action spaces. This comparison aids in grasping how well RL algorithms perform in various gaming environments.

Games Benefiting from Reinforcement Learning

Considering the discussion made above on the key reinforcement learning algorithms applied in game playing, it is important to notice that different game types pose unique challenges and have had success differently with the application of RL. RL has been applied with great success across a wide spectrum of game genres going as far as turn-based strategy games requiring long-term planning to real-time environments demanding fast responses to decisions. Categorizing these games according to the nature of their interaction with RL would thus let us better understand how reinforcement learning has revolutionized each genre and driven new developments in game AI.

Strategy Games

Turn-based strategy games, in particularly Chess and Go (9), rely greatly on experience learning since planning needs to be rather long-term. The RL techniques, applied here, strive greatly for optimal actions by analysing the great number of possible future scenarios. AlphaGo (4) of DeepMind used MCTS with the help of RL to beat human go champions. The algorithm continuously learnt the best strategy from millions of simulated games by using reward signals to learn the best moves.

Real-Time Strategy Games

Real-Time Strategy (RTS) games like StarCraft II and Dota 2 require agents to make real-time decisions involving resource management, multi-agent coordination, and long-term strategies. Example: AlphaStar is an RL-based agent that was trained by Deep Q-Learning and policy gradients for superhuman performance in StarCraft II (10). Here, the agent will first learn from the human replays, then perfect its strategy through self-play. Open AI Five is another example where the RL agent defeated professional human players at real-time decision-making in the multi-player online game, Dota 2 (11).

Arcade Games

First experiments with RL algorithms were done on less complex games like Pong and Breakout. These games made good bench marks for early work in RL because their state spaces are small and the rewards are well defined. Example: DQN of Mnih *et al*. (8) achieved superhuman performance in Atari 2600 games, including Pong and Breakout, by learning to maximize rewards through trial and error, outperforming human players.

First-Person Shooter (FPS) games

The challenge, then, is much greater for RL agents with FPS games such as Quake III Arena because it involves the need for navigation through 3D environments, teamwork, and instantaneous actions (37).

Example: In Quake III's Capture the Flag mode, DeepMind trained RL agents to collaborate with teammates and defeat opponents in a multiplayer environment. This involved mastering strategies such as defence, offense, and resource management, all learned autonomously through reinforcement learning. Platformer Games Classic platformer examples, such as Super Mario Bros, require running through levels full of obstacles and enemies. RL agents optimized in such environments are trained to optimize movement and actions to complete levels efficiently. Example: The NEAT algorithm, Neuroevolutionary of Augmenting Topologies, evolved RL agents to play Super Mario Bros: it modified the neural network of the agent based on performance; thus, it learned policies on how it should be navigating through complex levels.

Additional Metrics and Challenges in RL for Games

Even though game scores and win rates have been fairly popular criteria to measure the performance of reinforcement learning agents, such metrics alone do not allow one to hold a comprehensive assessment of how an agent performs, especially in more intricate gaming environments. Other important aspects, including involvement by players, computational efficiency, and generalization across various game contexts, are critical determinants of overall effectiveness for RL algorithms in video games. These additional metrics offer deeper insights into how well RL agents perform beyond achieving high scores, focusing instead on their adaptability, efficiency, and interaction with human players. Furthermore, the challenges encountered by RL agents, such as computational complexity, environment design, generalization, and real-time constraints, demonstrate the obstacles that need to be overcome for RL to reach its full potential in gaming.

Player Involvement

Beyond game scores, RL agents can be evaluated based on how engaging they make the game for human players, particularly in co-op or team-based settings. For instance, in Quake III Arena, DeepMind's RL agents learned to collaborate with human teammates, making strategic decisions that enhanced player engagement, such as defending teammates or coordinating attacks. Player involvement is usually measured through feedback, time spent in the game, or the level of collaboration between the RL agent and human players.

Computational Complexity

RL algorithms tend to be computationally expensive, and this is highly exacerbated in complex games with massive numbers of states and actions. The more subtle an environment is in a video game, the greater it becomes to handle computationally. For instance, in games like StarCraft II, an agent must deal with multiple resources, multiple units, and strategy framing against opponents in real-time (9). Thus, the number of states explodes exponentially as the number of elements, and it is impossible to evaluate all cases in detail. Likewise, in games with large action spaces, such as Dota 2, the agent has to choose the best equipment, skills, and

strategies, making it even tougher for RL agents to learn efficient policies. Thus, RL algorithms must be computationally efficient to process these extensive arrays of states and actions in real-time settings. Example: In Alpha Go, RL algorithms required tremendous computational infrastructure, consuming as many as 1,920 CPUs and 280 GPUs for training. Even though improvements like PPO have reduced the overhead, the resource burden is still heavy, especially in RTS games. What actually measures efficiency would be train time, resource usage-e.g., GPU/CPU-and whether or not the algorithm can scale to larger environments.

Generalization across Game Contexts

One key limitation of RL agents is over fitting, where an agent trained in specific scenarios performs well in those conditions but struggles when faced with unfamiliar situations (33). This highlights the difficulty of generalization in games with diverse content. For example, an RL agent trained on a specific map in Fortnite may fail to adapt to a new map with different terrain and environmental conditions. Transfer learning techniques are also being explored to improve the generalization, which enables the RL agents to apply learned knowledge from one game or environment with minimal retraining to another. For example, an RL agent mastered in navigation of levels in Super Mario Bros. can transfer its knowledge to other plat formers such as Celeste, despite the differences in game mechanics. Generalisation is probed by testing the performance of an agent in an unseen, entirely new environment. Dropped performance reveals limitations in adaptability in the agent.

Real-Time Constraints

Latency and response time are key aspects of video games that require real-time decision-making, and delays can have a substantial impact on performance. To compete with human players, RL agents must make quick judgments, necessitating algorithms that operate on a tight time schedule (38). For instance, in fast-paced games such as "Over watch," split-second decisions are very important. An RL agent would need to react practically immediately to the actions of the opponents in order to be effective, meaning that algorithms need to be highly fine-tuned. Besides, in real-time strategy games, agents must strike a balance between long-term planning and instant action in managing resources, creating plans, and responding to opponents in real time. For instance, in "Age of Empires," an agent would have to collect resources, create structures, train soldiers, and generate a strategy against other players. This does require sophisticated planning and swift decision-making, posing serious obstacles to RL algorithms.

Conclusion

The relationship between video games and RL is complementary and keeps redefining possibilities in the gaming industry. Through this review paper, we have examined the fundamental ideas of RL and its use in a variety of gaming contexts—from vintage arcade games to contemporary multiplayer experiences. Several RL algorithms have been analyzed in order to shed light on their advantages and disadvantages when it comes to solving challenging gameplay tasks, such as mastering subtle control tasks in immersive virtual environments or learning optimal strategies in strategic simulations (39). It seems certain that the incorporation of AI-driven agents into video games will go further in the future, providing players with previously unheard-of degrees of immersion, difficulty, and engagement.

However, incorporating RL into video games involves a number of serious challenges that must be overcome in order to fully realize its potential. The key hurdles for RL in video games include computational complexity, environment design, generalization, and real-time limitations. These difficulties stem from the complex nature of video game settings, the variety of player interactions, and the requirement for agents to make quick, strategic judgments in real-time. Despite these obstacles, tremendous progress has been made in developing RL techniques and methodologies. Researchers have created novel algorithms, frameworks, and platforms for training RL agents in various gaming scenarios, paving the way for more intelligent and engaging gaming experiences.

Future of RL in Video Games

The future of RL in video games holds vast potential for ongoing research and innovation. Advances in algorithmic techniques, such as deep reinforcement learning, multi-agent systems, and hierarchical reinforcement learning, are poised to improve AI agents' capabilities in gaming

contexts. Furthermore, creating complex settings that effectively imitate real-world dynamics, such as dynamic weather patterns, complex physics interactions, and evolving player strategies, will require RL agents to adapt and learn robust methods. As researchers investigate new approaches to reward engineering, generalization, and transfer learning, RL agents are projected to become more versatile and adaptable across a variety of games, scenarios, and settings.

Abbreviations

AI: Artificial Intelligence, A3C: Asynchronous Advantage Actor-Critic, CNN: Convolutional Neural Network, DQN: Deep Q Network, FPS: First Person Shooter, MDP: Markov Decision Process, ML: Machine learning, NEAT: Neuro Evolution of Augmenting Topologies, NPCs: Non-Player Characters, PPO: Proximal Policy optimization, RL: Reinforcement Learning, RTS: Real-Time Strategy, SAC: Soft Actor Critic, SARSA: State Action Reward State Action, TD: Temporal Difference.

Acknowledgement

The authors want to acknowledge the enabling research environment provided by the Department of Computer Science and Applications, Panjab University, Chandigarh.

Author Contributions

Ashish Kumar conducted the literature review, synthesized the findings, and drafted the manuscript. Jasleen Kaur Bains guided the research process, conceptualized the study, and revised the manuscript. Both authors read and approved the final manuscript.

Conflict of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Ethics Approval

This study did not involve any human or animal subjects, and therefore did not require ethics approval.

Funding

No funding was received.

References

1. Hodges A. Alan Turing and the Turing test. Springer. 2009.
2. Arulkumaran K, Deisenroth MP, Brundage M, Bharath AA. Deep reinforcement learning: A brief survey. IEEE Signal Process Mag. 2017;34(6):26–38.
3. Togelius J. Playing smart: On games, intelligence, and artificial intelligence. MIT Press; 2019.
4. Wang FY, Zhang JJ, Zheng X, Wang X, Yuan Y, Dai X, et al. Where does AlphaGo go: From church-turing thesis to AlphaGo thesis and beyond. IEEE/CAA Journal of Automatica Sinica. 2016;3(2):113–20.
5. Ramlan AA Bin, Ali AM, Hamid NHA, Osman R. The implementation of reinforcement learning algorithm for ai bot in fighting video game. In: 2021 4th International Symposium on Agents, Multi-Agent Systems and Robotics (ISAMSR). IEEE. 2021:96–100.
6. Sutton RS. Reinforcement learning: An introduction. A Bradford Book. 2018.
7. Watkins C, Dayan P. Q-learning. mach. learn. In Learn. 1992.
8. Mnih V. Playing atari with deep reinforcement learning. arXiv preprint arXiv:13125602. 2013.
9. Silver D, Hubert T, Schrittwieser J, Antonoglou I, Lai M, Guez A, et al. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. Science. 2018;362(6419):1140–4.
10. Vinyals O, Babuschkin I, Czarnecki WM, Mathieu M, Dudzik A, Chung J, et al. Grandmaster level in StarCraft II using multi-agent reinforcement learning. Nature. 2019;575(7782):350–4.
11. Berner C, Brockman G, Chan B, Cheung V, Dębiak P, Dennison C, et al. Dota 2 with large scale deep reinforcement learning. arXiv preprint arXiv:191206680. 2019.
12. Schulman J, Wolski F, Dhariwal P, Radford A, Klimov O. Proximal policy optimization algorithms. arXiv preprint arXiv:170706347. 2017.
13. Silver D, Huang A, Maddison CJ, Guez A, Sifre L, Van Den Driessche G, et al. Mastering the game of Go with deep neural networks and tree search. Nature. 2016;529(7587):484–9.
14. Kumar S, Punitha S, Perakam G, Palukuru VP, Raghavaraju JV, Praveena R. Artificial Intelligence (AI) Prediction of Atari Game Strategy by using Reinforcement Learning Algorithms. In: 2021 International Conference on Computational Performance Evaluation (ComPE). IEEE. 2021:536–9.
15. Shaker N, Togelius J, Nelson MJ. Procedural content generation in games. 2016.
16. Naeem M, Rizvi STH, Coronato A. A gentle introduction to reinforcement learning and its application in different fields. IEEE access. 2020;8:209320–44.
17. Kavukcuoglu K. Asynchronous methods for deep reinforcement learning. In: Int Conf Mach Learn(ICML). 2016:2850–69.
18. Hammoudeh A. A concise introduction to reinforcement learning. Princess Suamaya University for Technology: Amman, Jordan. 2018.
19. Sigaud O, Buffet O. Markov decision processes in artificial intelligence. John Wiley & Sons; 2013.
20. Jia J, Wang W. Review of reinforcement learning research. In: 2020 35th Youth Academic Annual Conference of Chinese Association of Automation (YAC). IEEE. 2020:186–91.
21. O'Donoghue B, Osband I, Munos R, Mnih V. The uncertainty bellman equation and exploration. In: International conference on machine learning. 2018:3836–45.

22. Sebastianelli A, Tipaldi M, Ullo SL, Glielmo L. A Deep Q-Learning based approach applied to the Snake game. In: 2021 29th Mediterranean Conference on Control and Automation (MED). IEEE. 2021:348–53.
23. Jang B, Kim M, Harerimana G, Kim JW. Q-learning algorithms: A comprehensive classification and applications. IEEE access. 2019;7:133653–67.
24. Almalki AJ, Wocjan P. Exploration of reinforcement learning to play snake game. In: 2019 International Conference on Computational Science and Computational Intelligence (CSCI). IEEE. 2019:377–81.
25. Mnih V, Kavukcuoglu K, Silver D, Rusu AA, Veness J, Bellemare MG, et al. Human-level control through deep reinforcement learning. Nature. 2015;518(7540):529–33.
26. Kumar S. Balancing a CartPole System with Reinforcement Learning--A Tutorial. arXiv preprint arXiv:200604938. 2020.
27. Fu MC. A tutorial introduction to Monte Carlo tree search. In: 2020 Winter Simulation Conference (WSC). IEEE. 2020:1178–93.
28. Farag W. Multi-agent reinforcement learning using the deep distributed distributional deterministic policy gradients algorithm. In: 2020 International Conference on Innovation and Intelligence for Informatics, Computing and Technologies (3ICT). IEEE. 2020:1–6.
29. Phon-Amnuaisuk S. Learning to Play Pong using Policy Gradient Learning. arXiv preprint arXiv:180708452. 2018.
30. Pepels T, Winands MHM, Lanctot M. Real-time monte carlo tree search in ms pac-man. IEEE Trans Comput Intell AI Games. 2014;6(3):245–57.
31. Stadie BC, Abbeel P, Sutskever I. Third-person imitation learning. arXiv preprint arXiv:170301703. 2017.
32. Haarnoja T, Zhou A, Abbeel P, Levine S. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In: International conference on machine learning. PMLR. 2018:1861–70.
33. Cobbe K, Klimov O, Hesse C, Kim T, Schulman J. Quantifying generalization in reinforcement learning. In: International conference on machine learning. PMLR. 2019:1282–9.
34. Lohokare A, Shah A, Zyda M. Deep learning bot for league of legends. In: Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment. 2020:322–4.
35. Angulo E, Lahuerta X, Roca O. Reinforcement Learning in Minecraft. 2020. https://scholar.google.com/scholar?hl=en&as_sdt=0%2C5&q=%5D+Reinforcement+Learning+in+Minecraft+by+E+Angulo&btnG=
36. Fuchs F, Song Y, Kaufmann E, Scaramuzza D, Dürr P. Super-human performance in gran turismo sport using deep reinforcement learning. IEEE Robot Autom Lett. 2021;6(3):4257–64.
37. Jaderberg M, Czarnecki WM, Dunning I, Marris L, Lever G, Castaneda AG, et al. Human-level performance in 3D multiplayer games with population-based reinforcement learning. Science (1979). 2019;364(6443):859–65.
38. Bard N, Foerster JN, Chandar S, Burch N, Lanctot M, Song HF, et al. The hanabi challenge: A new frontier for ai research. Artif Intell. 2020;280:103216.
39. Setiaji B, Pujastuti E, Filza MF, Masruro A, Pradana YA. Implementation of reinforcement learning in 2d based games using open AI gym. In: 2022 International Conference on Informatics, Multimedia, Cyber and Information System (ICIMCIS). IEEE. 2022:293–7.