REPUBLIC OF TÜRKİYE

ALTINBAŞ UNIVERSITY

Institute of Graduate Studies

Electrical and Computer Engineering

EMPOWERING ATARI GAMING WITH DEEP REINFORCEMENT LEARNING

Sadeq Mohammed Kadhim SARKHI

Master’s Thesis

Supervisor

Assoc. Prof. Dr. Hakan KOYUNCU

İstanbul, 2024

EMPOWERING ATARI GAMING WITH DEEP REINFORCEMENT LEARNING

Sadeq Mohammed Kadhim SARKHI

Electrical and Computer Engineering

Master's Thesis

ALTINBAŞ UNIVERSITY

2024

The thesis titled EMPOWERING ATARI GAMING WITH DEEP REINFORCEMENT LEARNING prepared by SADEQ MOHAMMED KADHIM SARKHI and submitted on 18/04/2024 has been **accepted unanimously** for the degree of Master of Science in Electrical and Computer Engineering

Asst. Prof. Dr. Hakan KOYUNCU
Supervisor

Thesis Defense Committee Members:

Asst. Prof. Dr. Hakan KOYUNCU	Department of Computer Engineering, Altınbaş University	
Asst. Prof. Dr. Abdullahi Abdu IBRAHIM	Department of Computer Engineering, Altınbaş University	
Asst. Prof. Dr. Warish PATEL	Department of Computer Engineering, Parul University	

I hereby declare that this thesis meets all format and submission requirements of a Master's thesis.

I hereby declare that all information presented in this graduation project has been obtained in full accordance with academic rules and ethical conduct. I also declare all unoriginal materials and conclusions have been cited in the text and all references mentioned in the Reference List have been cited in the text, and vice versa as required by the abovementioned rules and conduct.

Sadeq Mohammed Kadhim SARKHI

Signature

DEDICATION

I thank God Almighty for helping me and granting me success in completing this thesis.

This thesis is wholeheartedly devoted to the memory of my mother and father. It is impossible for me to put into words how much it means to me. There is absolutely nothing I can do to make up for what you have done to me. I shall keep doing everything in my power to live up to the standards you have set.

In conclusion, I would want to say thank you to my family. They helped me a lot and encouraged me to complete this project.

PREFACE

This study aims to develop better procedures and research ideas in our countries, as well as design tools to help future generations overcome access constraints.

I could not have reached my current level of success without the assistance of a strong support network. First and foremost, I want to thank my family for their love and understanding. Second, my supervisor, who has advised and guided me throughout the study process.

Thank you for your unending support.

ABSTRACT

EMPOWERING ATARI GAMING WITH DEEP REINFORCEMENT LEARNING

SARKHI, Sadeq Mohammed Kadhim

M.Sc., Electrical and Computer Engineering, Altınbaş University,

Supervisor: Asst. Prof. Dr. Hakan KOYUNCU

Date: 04 / 2024

Pages: 92

Deep Reinforcement Learning (DRL) has made significant strides in the domain of gaming, optimizing artificial intelligence agents to navigate intricate game environments. This study embarks on an exploration of the Snake Optimization Algorithm (SOA) and the Energy Valley Optimization (EVO), it synergizes into a unified approach, aptly named the Energy Serpent Optimizer (ESO), benchmarking their efficacy within a maze-like game setting. Within this environment, an AI agent is set to maneuver through complex pathways, while engaging with diverse challenges such as snakes, skulls, and other animated characters. The overarching objective for the agent is to adeptly navigate the maze, sidestep potential threats, and engage with specific game elements to amass points. A comparative analysis of ESO revealed notable differences in their execution time efficiencies. The SOA emerged as the more time-efficient algorithm, clocking in at a mere 0.43 seconds, This discernible time gap accentuates the superior efficiency of ESO in this particular setting. Additionally, the ESO was put to the test in the said game environment, where the optimization process entailed evolving a set of hyperparameter configurations via genetic algorithms. The goal was to iterate and adapt these configurations to maximize the AI agent's in-game reward, while ensuring the process is time-efficient. Impressively, the AI agent, under the guidance of ESO, achieved a remarkable reward score of 1100.0 in a span of 32 seconds.

Keywords: Snake Optimization Algorithm, Energy Valley Optimization, Deep Reinforcement Learning, Game Environment, AI Agent, Genetic Algorithms, Hyperparameter Configurations.

TABLE OF CONTENTS

Pages

LIST OF TABLES

Pages

LIST OF FIGURES

Pages

1. INTRODUCTION

1.1 GENERAL

Machine learning encompasses a diverse range of applications, including predictive analysis, data classification, regression assessment, and clustering tasks. In the context of supervised learning (SL) [1], the system is provided with input and output data (labeled data) to generate a model. Conversely, unsupervised learning (UL) [2] focuses on discerning patterns, resemblances, and variations within unlabeled data, unveiling concealed insights. These paradigms, along with RL [3], constitute the triad of machine learning approaches. The resultant model can prognosticate outcomes for novel, unfamiliar data instances. Noteworthy applications of supervised learning comprise spam detection [4], image recognition, object detection [5], predictive analytics [6], as well as text categorization.

Diverse algorithms, such as logistic regression, linear regression, decision trees, random forests, and support vector machines (SVM) [7], are utilized within the realm of supervised learning. Unsupervised learning, on the other hand, encompasses techniques such as the k-means clustering algorithm for grouping data and principal component analysis (PCA) for dimensionality reduction.

Overcoming the limitations of traditional machine learning, deep learning has emerged as a pivotal advancement. While conventional machine learning involves distinct feature extraction, deep learning undertakes automatic feature extraction devoid of human intervention. Moreover, deep learning exhibits prowess in handling substantial datasets, accommodating millions of data points. This makes it proficient in processing unstructured data types like images and audio. Notably, real-time scenarios like recommendation systems and autonomous driving, characterized by their dynamic nature and limited training datasets, necessitate a different approach.

Here, RL steps in to tackle this challenge. RL, the third dimension of machine learning [8], stands as a fitting solution for sequential decision-making processes. It learns through direct engagement with the environment, striving to attain long-term objectives independent of external incentives or complete environmental understanding. A visual representation of the RL framework is depicted in Figure 1-1.

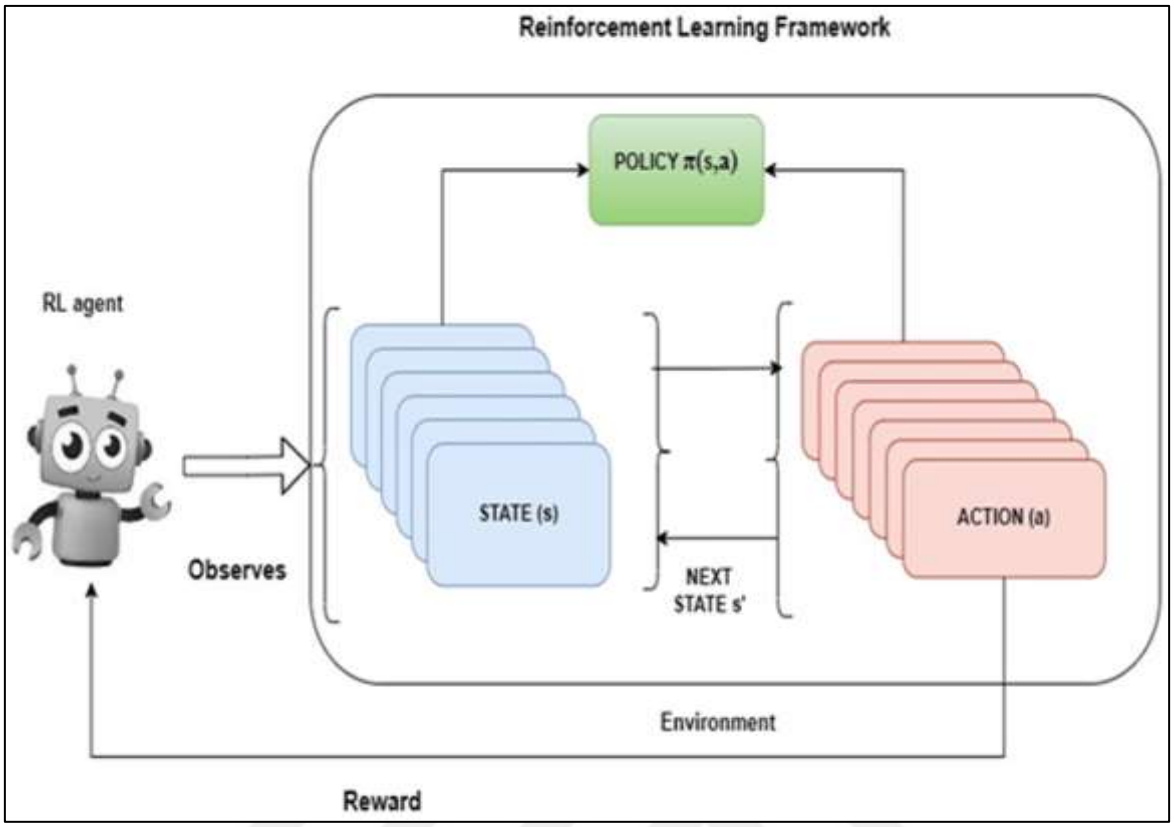

Figure 1.1: RL framework [9].

The framework of RL involves the RL agent evaluating the state of the environment and then selecting a suitable course of action. When the RL agent makes a correct decision, it garners a positive reward; conversely, an incorrect move leads to a negative outcome. Striking a balance between exploration and exploitation is a pivotal aspect of RL. Exploitation arises when the agent strives to maximize rewards based on known strategies, whereas exploration pertains to the agent's pursuit of novel pathways to its goal. This interplay between the two is crucial in the RL process.

Numerous applications harness the potential of RL. The unique characteristic of RL is its ability to learn through trial and error, obviating the need for extensive datasets. This stands in contrast to other learning paradigms. As the RL model interacts directly with the environment, it iteratively refines its actions to achieve optimal results. This dynamic self-improvement process enables RL to excel in scenarios where large datasets are lacking, and the learning process hinges on real-time interactions.

One of the most notable and pioneering applications of reinforcement learning is within the realm of playing Atari games. RL has shown remarkable prowess in mastering complex video games, achieving superhuman performance in a range of titles. The concept of using RL to play Atari games gained widespread attention with the introduction of Deep Q-Networks (DQN), a breakthrough approach that melded deep learning with RL.

In this context, the RL agent is akin to a game player who interacts with the Atari game environment. The agent perceives the game's visual and numerical information as its input and takes actions to influence the gameplay. Through this process, the agent navigates challenges, learns strategies, and optimizes its decision-making to maximize cumulative rewards over time.

The brilliance of this approach lies in its ability to learn directly from pixel inputs, transcending the need for explicit feature engineering. By leveraging convolutional neural networks (CNNs) as function approximators, the DQN model processes the raw game frames, learning to associate visual patterns with optimal actions. As the agent plays the game repeatedly, it refines its understanding, adapts to different scenarios, and gradually hones its performance.

The atypical and challenging nature of Atari games—each with distinct dynamics, objectives, and complexities—illustrates the versatility of RL. The agent not only masters the rules of the games but also discovers intricate strategies, anticipates future states, and refines its actions to achieve high scores. This adaptability showcases the essence of RL's trial-and-error learning, as the agent continually explores uncharted paths while capitalizing on established strategies.

Metaheuristic optimization techniques have emerged as compelling approaches to enhance the performance of RL agents in mastering Atari games. These techniques offer innovative ways to fine-tune the parameters and policies of RL agents, enabling them to navigate the complex game environments more effectively and efficiently.

In the context of Atari games, metaheuristic optimization serves as a valuable tool to overcome the challenges associated with high-dimensional action spaces and intricate game dynamics. Evolutionary algorithms, such as genetic algorithms and particle swarm optimization, offer the ability to explore a wide range of parameter settings and policies.

These algorithms mimic natural selection and social behaviors, respectively, to guide the RL agent towards optimal strategies.

By integrating metaheuristic optimization with RL, the agent's learning process becomes more targeted and adaptive. Rather than relying solely on the agent's trial-and-error interactions with the game, metaheuristic optimization guides the exploration of the action space, helping the agent discover promising directions for policy improvement. This synergy between RL and metaheuristics enables the agent to converge to effective policies more swiftly and often with superior performance.

Moreover, metaheuristic optimization mitigates the potential pitfalls of RL, such as getting stuck in suboptimal solutions or experiencing slow convergence. These optimization techniques introduce diversity in exploration, enabling the agent to escape local optima and uncover previously unexplored strategies. The agent thus benefits from a broader exploration of the solution landscape, leading to more robust and adaptive gameplay.

1.2 STATEMENT OF THE PROBLEM

With the rapid growth in gaming and artificial intelligence domains, traditional algorithms for game playing have often demonstrated limitations in adapting to the complexities of certain game environments, particularly those that are highly dynamic and unpredictable like Atari games [10]. While deep reinforcement learning (DRL) has shown promise in enabling agents to learn directly from raw pixel data and act intelligently within game environments, there is a prevailing need to (1) evaluate and compare the effectiveness of various DRL algorithms in playing Atari games; (2) optimize these algorithms for enhanced gameplay performance; and (3) provide a comprehensive solution that integrates optimal decision-making strategies without relying on handcrafted features. Specifically, this research seeks to address the challenge of training a deep reinforcement learning agent, using the deep Q-network approach, to achieve superior gameplay in the Atari game "Pacman" while optimizing its parameters with metaheuristic techniques like snake optimization Energy Valley Optimization. The overarching problem is: How can deep reinforcement learning be empowered and optimized for superior Atari gameplay, and what challenges and potential solutions exist in its implementation and fine-tuning?

1.3 RESEARCH OBJECTIVES

a. Literature Review and Gap Analysis:

   a. To conduct a comprehensive review of existing literature on deep reinforcement learning (DRL), its applications in Atari gaming, and the use of metaheuristic optimization techniques.

   b. To identify gaps in current research, particularly in the application of DRL algorithms for playing Atari games and their optimization using techniques like snake optimization Energy Valley Optimization.

b. Development and Implementation:

   a. To design and implement a deep reinforcement learning agent, employing the deep Q-network approach, capable of playing the Atari game "Pacman."

   b. To integrate the agent with the game environment using the gym library from OpenAI.

c. Algorithm Comparison:

   a. To compare the effectiveness and performance of traditional and current deep reinforcement learning algorithms in the context of Atari gaming.

d. Optimization and Fine-tuning:

   a. To utilize metaheuristic optimization techniques, specifically snake optimization Energy Valley Optimization, for parameter tuning of the deep Q-network.

   b. To assess the impact of optimization on gameplay performance, learning efficiency, and stability.

e. Evaluation:

   a. To evaluate the performance of the proposed deep reinforcement learning agent in terms of learning efficiency, stability, and overall gameplay in the "Pacman" environment.

b. To identify strengths, weaknesses, and potential improvements of the proposed model.

f. Recommendation and Future Directions:

a. To recommend enhancements based on the findings, particularly in areas of algorithm design, parameter tuning, and game mechanics.

b. To highlight potential areas of future research in the realm of DRL and its applications in gaming, emphasizing the integration of newer optimization techniques.

1.4 RESEARCH QUESTIONS

a. Understanding the Domain:

a. What are the primary challenges and limitations of traditional algorithms in playing Atari games like "Pacman"?

b. Efficacy of DRL in Gaming:

a. How effectively can deep reinforcement learning algorithms learn and adapt to Atari gaming environments, specifically "Pacman", compared to traditional algorithms?

c. Algorithmic Variations and Comparisons:

a. How do various deep reinforcement learning algorithms, differ in terms of performance, learning efficiency, and stability in the context of Atari games?

d. Optimization Concerns:

a. How does the metaheuristic optimization technique, specifically snake optimization Energy Valley Optimization, impact the parameter tuning of deep Q-networks in Atari gameplay?

b. What are the benefits and potential drawbacks of integrating snake optimization Energy Valley Optimization for parameter tuning in the context of deep reinforcement learning for gaming?

e. Evaluation Metrics and Findings:

a. Which evaluation metrics best capture the performance, learning efficiency, and adaptability of a DRL agent in an Atari game environment?

b. Based on the chosen metrics, how does the proposed DRL model perform in the "Pacman" environment?

f. Future Implications and Enhancements:

a. What enhancements or modifications can be recommended based on the research findings to further improve the DRL agent's performance in Atari gaming?

b. What potential avenues exist for future research in optimizing and refining deep reinforcement learning techniques for diverse game environments?

1.5 CONTRIBUTION

This research pioneers an intricate exploration into the realms of Atari gaming through the lens of DRL, distinguishing itself with a unique intersection of advanced DRL algorithms and optimization techniques. By venturing beyond traditional game-playing algorithms, the study introduces a model employing the deep Q-network approach tailored for the Atari game "Pacman", a significant stride towards enhancing gaming AI's adaptability and performance. Furthermore, the research uniquely integrates the snake optimization And Energy Valley Optimization metaheuristic, a seldom-explored avenue in the context of DRL for Atari gaming, to fine-tune the model parameters. Through rigorous evaluations and comparisons, the study not only delineates the strengths and weaknesses of prevalent DRL algorithms but also offers a comprehensive blueprint for future endeavors in the domain. In essence, this research fills a critical gap in gaming AI literature, amalgamating advanced learning techniques with cutting-edge optimization, poised to set new benchmarks and open doors to innovative approaches in AI-driven game development.

1.6 METHODOLOGY

To holistically examine the application of deep reinforcement learning (DRL) in Atari gaming, a systematic and multi-faceted methodology is adopted. The journey commences with a thorough literature review, scrutinizing existing research to capture the current state-of-the-art and potential research gaps in DRL and Atari gaming. Armed with this

foundational understanding, data is collected from the Atari game environment, particularly "Pacman", and subsequently preprocessed to ensure compatibility with DRL agents. Multiple DRL models are then implemented and experimented with, aiming to glean insights into their relative performance metrics.

The research also integrates an innovative layer of algorithmic refinement using the metaheuristic snake optimization And Energy Valley Optimization technique for parameter tuning. Rigorous evaluations, encompassing both qualitative and quantitative aspects, subsequently follow. Performance metrics assess learning efficiency, gameplay effectiveness, and adaptability of the proposed models. Finally, findings are distilled into a comprehensive discussion and analysis segment, leading to informed recommendations and suggesting pathways for further exploration in the domain.

1.7 SIGNIFICANCE

The profundity of this research lies in its fusion of cutting-edge deep reinforcement learning techniques with the dynamic realm of Atari gaming, setting a precedent in artificial intelligence-driven game development. By navigating beyond the confinements of traditional algorithms and introducing innovative optimization strategies, the study illuminates new avenues for creating highly adaptive and efficient game-playing agents. The specific focus on "Pacman" offers tangible benchmarks for future research, ensuring replicability and scalability. Furthermore, the integration of snake optimization And Energy Valley Optimization for parameter tuning presents a novel intersection of DRL and metaheuristic techniques, with the potential to revolutionize algorithmic fine-tuning practices across multiple disciplines. This research doesn't just push the boundaries of what's achievable in gaming AI, but it also serves as a beacon for interdisciplinary studies, demonstrating the transformative impacts of merging diverse technological domains. In essence, the significance of this study reverberates through its potential to inspire new paradigms, drive technological advancements, and shape the future narrative of AI in gaming.

1.8 RESEARCH LIMITATIONS

While this study promises a groundbreaking examination of deep reinforcement learning in Atari gaming, certain inherent limitations should be acknowledged. Firstly, the research's concentration on a singular game, "Pacman", might not capture the complete spectrum of challenges posed by other Atari games, thereby limiting the generalizability of findings. The choice of snake optimization And Energy Valley Optimization as the primary metaheuristic technique, though innovative, leaves out other potential optimization strategies that could offer varied perspectives. The reliance on specific DRL models might overlook emerging or less-documented algorithms, which could perform differently under similar conditions. Additionally, the real-time adaptability of the proposed models in varying game scenarios, influenced by diverse external factors, hasn't been exhaustively tested. Lastly, computational constraints and the evolving nature of AI tools and libraries might introduce variations in replicability and scalability in future research endeavours.

1.9 REPORT ORGANIZATION

Chapter I: Introduction

This chapter sets the stage for the research by highlighting the motivation behind the study. It introduces the primary objectives, the problem statement, and the overarching significance of the research in the broader context of AI-driven game development. Additionally, it provides a brief overview of the research methodology and the scope of the study, guiding readers on what to expect in subsequent chapters.

Chapter II: Background and Literature Review

Serving as the theoretical backbone of the research, this chapter delves deep into the existing literature. It scrutinizes prior works on Atari gaming, deep reinforcement learning algorithms, and metaheuristic optimization techniques like snake optimization Energy Valley Optimization. And Energy Valley Optimization By establishing a solid theoretical foundation, this chapter aims to highlight the current state-of-the-art, identify research gaps, and underline the need for the present study.

Chapter III: Methodology

Detailing the heart of the research process, this chapter elucidates the systematic approach adopted to achieve the research objectives. It outlines data collection and preprocessing from the "Pacman" Atari game, the design and implementation of various DRL models, the unique integration of snake optimization Energy Valley Optimization for parameter tuning, and the evaluation criteria and techniques. Each segment is expounded with meticulous detail, ensuring replicability and clarity.

Chapter IV: Experimental Results

Here, the outcomes of the implemented models and experiments are presented. The chapter provides a comprehensive account of performance metrics, comparisons between various DRL algorithms, and the impact of snake optimization Energy Valley Optimization on gameplay efficiency. Visual aids like graphs, tables, and charts may be employed to offer a clearer picture of the results. Analytical commentary accompanies the raw data to offer insights and preliminary interpretations.

Chapter V: Conclusion and Future Works

Culminating the research journey, this chapter revisits the primary objectives, encapsulating the key findings and their implications. It discusses the research's contributions to the field, acknowledges the limitations identified earlier, and provides recommendations for improvements. The chapter concludes with a glimpse into potential future research avenues, inspiring further exploration and study in the domain.

2. LITERATURE REVIEW

2.1 INTRODUCTION

The domain of artificial intelligence has persistently sought inspiration from human endeavors, and one such pursuit that stands out is gaming. The intricate dance of strategy, adaptability, and decision-making in games has long been a tantalizing frontier for AI researchers. Atari games, with their pixelated charm and myriad challenges, hold a special place in this exploration. As we venture into the depths of this study, it becomes imperative to first understand the foundational theories, groundbreaking achievements, and existing gaps in the intricate nexus of AI, deep reinforcement learning, and Atari gaming. This chapter endeavors to weave together the tapestry of past research, drawing insights from pioneering works and critically analyzing the evolution of AI-driven game development. Through a comprehensive review of literature, we aim to position our research within the broader academic discourse, appreciating the milestones already achieved and highlighting the territories yet to be explored. As we traverse through the annals of past studies, readers will gain a profound understanding of the journey thus far, setting the stage for the innovative endeavors this research promises.

2.2 BACKGROUND

2.2.1 Traditional Reinforcement Learning

Within the realm of RL, an agent endeavors to conquer a task through an iterative process of trial and error. This process involves the agent's dynamic engagement with an environment of unknown dynamics. The agent, in its pursuit, wields the power to manipulate the environment's state through its actions, promptly receiving consequential feedback. At its core, the agent strives to unearth an optimal sequence of actions that unravels the task's solution.

Noteworthy is the intrinsic distinction between reinforcement learning and conventional machine learning methods, be they supervised or unsupervised. Unlike the latter, RL isn't reliant on data accumulation. Rather, it thrives on the agent's self-derived experiences shaped in tandem with the environment. The crux here is the agent's autonomy in learning, entirely

devoid of an external overseer. This separation from data dependence exemplifies the autonomous learning essence of RL.

Furthermore, the focal point of RL diverges from data analysis. Rather than dissecting data, RL is ardently concerned with shaping an impeccable policy. The policy, essentially a strategic roadmap, guides the agent's actions towards optimal outcomes. Visualizing this, Figure 2.1 illustrates the framework of reinforcement learning, where the agent and environment engage in dynamic interplay, resonating with the essence of autonomous learning and policy optimization.

Envisioned as a Markov Decision Process, RL typically finds its formal representation. This model encapsulates the sequential decision-making inherent in RL, highlighting the dynamic interrelation between the agent's actions and the environment's responses. Consequently, the intricate dance of RL emerges as a paradigm distinct from conventional machine learning, epitomizing autonomy, policy optimization, and the collaborative journey between the agent and its environment.

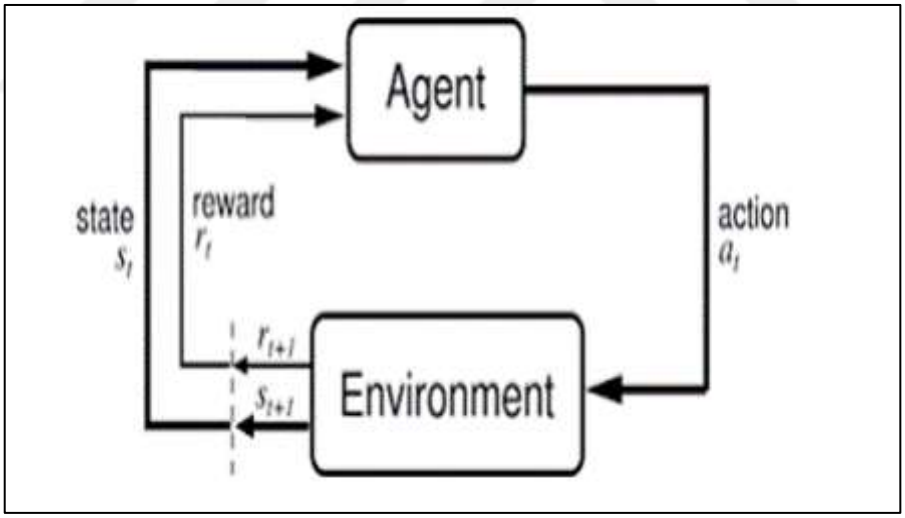

Figure 2.1: The Illustration Of Reinforcement Learning.

The agent receives the environment's current state, decides on a course of action, and then receives back the updated environment along with a reward that rates the effectiveness of that action. Markov Decision Processes (MDPs) represent a stochastic mathematical framework tailored to decision-making scenarios. In each step of this process, a decision-maker or agent opts for an action, thereby triggering outcomes influenced by both randomness and the chosen action [11]. MDPs find application in modeling an array of

optimization challenges, amenable to resolution through dynamic programming (DP) and RL. A Markov decision process encompasses five fundamental elements: states (S), actions (A), transition probabilities (P), rewards (R), and the discount factor $(\gamma)[i]$. Here:

Set S denotes the states in the system.

Set A captures the available actions.

$$P(s, a, s') = \Pr(s_{t+1} = s' | s_t = s, a_t = a) \tag{2.1}$$

Transition matrix quantifies the likelihood of transitioning from state s to state s' under action a at time t. Reward function $R(s, a, s')$ or alternatively $R(s)$ embodies the immediate or anticipated rewards upon taking action a from state s and reaching state's'.

The discount factor γ interplays short-term and long-term rewards' significance.

Central to MDPs is the quest for a "policy," represented by the function π, which links states to actions—$\pi(s) = a$. This policy can manifest either deterministically or stochastically. Upon amalgamating a Markov decision process with a policy in this manner, a definitive course of action is established for each state. Consequently, the amalgamation embodies the attributes of a Markov Reward Process, collectively shaping the agent's trajectory and influencing the interplay between actions, rewards, and state transitions.

The primary objective revolves around identifying a policy, denoted as π, that maximizes the anticipated discounted summation of rewards beyond the current state st. This cumulative return, often referred to as G_t, can be expressed as a summation of rewards from time $i = t$ onwards, weighted by the discount factor γ to account for the time horizon. In mathematical terms:

$$G_t = \sum_{i=t}^{\infty} \gamma^i . R(s_i, a_i, s_{i+1}) \tag{2.2}$$

Here, $a_i = \pi(s_i)$, and γ is the discount factor constrained within the range $0 \leq \gamma \leq 1$.

With this overarching goal established, let's delve into the methodologies for attaining such optimal behavior. For scenarios where the dynamics of the MDP and reward functions are known, a model-based approach relying on dynamic programming can offer a solution.

However, when this knowledge is absent, necessitating a model-free approach, RL methods step in.

Reinforcement learning strategies can be categorized into two principal streams: Value-Function based algorithms and Policy-based algorithms. Value-Function based algorithms center around the value-function Vπ(s), which assigns values to states. This value signifies the expected cumulative return G_t from state s onward, in line with policy π. It is governed by Bellman's equation:

$$V_\pi(s) = \sum_{a \in A} \pi(a|s) \sum_{s_0 \in S} P(s_0|s,a)[R(s,a,s_0) + \gamma V_\pi(s_0)] \quad (2.3)$$

Similarly, the value of executing action at in state st (referred to as Q(st, at)) is defined as the expected cumulative return Gt following the action. This is also governed by a Bellman equation:

$$Q_\pi(s,a) = \sum_{s_0 \in S} P(s_0|s,a)[R(s,a,s_0) + \gamma \sum_{a_0 \in A} \pi(a_0|s_0) Q_\pi(s_0,a_0)] \quad (2.4)$$

An optimal policy π^* is one where the value of any state s under $\pi *$ is greater than or equal to the value under any other policy π_0 for all s in the state space S.

Value-function based algorithms focus on learning the values of states and actions, allowing the derivation of an optimal policy π^* from the Q function. This approach involves comprehensively learning the value associated with each state s in S and action a in A, ultimately enabling the derivation of an optimal policy π^* from the Q function.

At the heart of value-function based algorithms lies the iterative enhancement of Q function parameters θ. This iterative process aims to diminish the squared error δ between the estimated Q value and its actual counterpart. Given that the real Q value remains elusive, its estimation is attained through experiential data.

The optimal methodology to assess the Q value materializes through temporal difference methods (TD). This technique incorporates immediate rewards and the evaluation of the subsequent state. Specifically, the TD evaluation is represented by $\tilde{Q}(s,a) = r + \gamma(s_0, a^*)$ and $\delta = \hat{Q}(s,a) - \tilde{Q}(s,a) = \hat{Q}(s,a) - r - \gamma\hat{Q}(s_0,a^*)$, with Q signifying the value in the table and $\tilde{Q}$ representing the approximated target value.

Formally, the optimization process is depicted as:

$$\theta_{i+1} = \theta_i + \Delta\theta_i \tag{2.5}$$

where $\Delta\theta_i$captures the parameter update, learned from δ.

Conversely, policy-based algorithms adopt a more direct approach. Instead of first ascertaining the value of all feasible states to then deduce an optimal policy, policy-based strategies aim to directly uncover a policy π that maximizes the anticipated return G. This entails iteratively adjusting the policy parameters θ to augment the expected return.

The exploration-exploitation dichotomy shapes the agent's behavior during the learning process. In navigating the environment, two strategies are available:

a. Exploration: Opting for random actions introduces novelty, allowing the agent to uncover new states and potentially superior policies.

b. Exploitation: Prioritizing known best actions based on existing knowledge optimizes cumulative rewards, aligning with the agent's understanding.

Effectively, exploration is favored when uncertainty surrounds the agent's knowledge, while exploitation is preferred when confidence prevails in the estimate of $Q(st, at)$ aligning with the true value. This strategic balance ensures effective learning and decision-making in dynamic environments.

Striking the right balance between exploration and exploitation is imperative for the agent's successful learning. Solely favoring exploration may hinder the agent from achieving high task scores and refining its actions. Conversely, an exclusive focus on exploitation could lead to the agent becoming entrenched in its current policy, overlooking potentially superior trajectories. This risk of missing the optimal policy underscores the necessity for a nuanced equilibrium between these two strategies.

An e-greedy policy stands as one of the most prevalent approaches, deftly combining exploration and exploitation. This policy hinges on a single parameter, denoted as "e," bounded within the range 0 to 1 ($0 \leq e \leq 1$). This parameter dictates the degree to which the agent departs from pure greediness when selecting actions. For every action choice, the agent probabilistically toggles between exploration and exploitation. With a probability of

e, it explores by randomly selecting an action from the available set, while with a probability of $1 - e$, it exploits by selecting the action deemed optimal.

The action selection process adheres to:

For $rand(0,1) \leq e: a = rand(a_n)$

Otherwise: $a = argmax_a Q$

Higher values of e steer the agent toward more frequent exploration, diminishing the probability of optimal action selection. This trade-off enables the agent to swiftly respond to environmental changes. Conversely, lower e values push the agent to exploit actions deemed optimal. Often, the e value within an episode follows a decreasing trend over time, where e approaches 0 as time progresses indefinitely. Such a choice imbues the agent with a growing inclination toward greedy behavior.

Another strategy for action selection is the Boltzmann Distribution (BD) policy, also known as the Gibbs Distribution or $softmax$ policy. BD seeks to temper the exploration tendency over time and rests on the assumption of model enhancement during learning. BD ascribes a probability to each possible action based on its anticipated utility and a parameter called "temperature" (T). This probability assignment occurs through the equation:

$$P(a|s) = \frac{e^{\frac{Q(s,a)}{T}}}{\sum_{a_0 \in A} e^{\frac{Q(s,a_0)}{T}}} \tag{2.6}$$

Employing the Boltzmann distribution for action selection offers the advantage of generating a stochastic policy. This policy injects a controlled degree of variability into behavior, intrinsically linked to the action-Q function and, consequently, the actions themselves. Moreover, the Boltzmann distribution serves as a model that mirrors human decision-making processes. Its stochastic nature not only enriches the exploration of potential strategies but also finds application as a cognitive model.

In the context of multi-goal environments, the conventional MDP model can be expanded to encompass scenarios where goals shift episodically. In such setups, each episode commences with the sampling of a goal from a set G, effectively introducing a dynamic element to the agent's objectives. In this context, the MDP framework becomes augmented by this set of

goals, prompting the agent to factor in the current goal while evaluating states or selecting actions. This expansion necessitates an extension of the notations used for value-function, Q function, and policy to accommodate the goal-oriented dimension: $V(s,g)$, $Q(s,a,g)$, and $\pi(s,g)$, respectively. This alteration reflects the agent's need to harmonize its actions with the current goal in multi-goal environments, underscoring the versatility of the MDP framework in accommodating diverse scenarios.

2.2.2 RL Algorithms

Q-learning, also recognized as a temporal difference method (TD), constitutes a value-function based algorithm with considerable contemporary significance [12]. In the realm of modern deep reinforcement learning, numerous algorithms draw inspiration from Q-learning. For instance, the Deep Q Network (DQN) algorithm leverages a variant of Q-learning wherein a neural network is employed as a function approximator. Temporal difference methods build on the TD evaluation of Q value, to iterate on the estimated Q function. This iterative process is reflected in the update rule, detailed as follows:

$$Q(s,a) = Q(s,a) - \alpha.\left(Q(s,a) - r - \gamma Q(s_0, a_0{}^*)\right) \tag{2.7}$$

Here, α represents the learning rate—a parameter governing the speed at which the Q function evolves. This Q-learning approach is instrumental in fine-tuning the agent's Q function, facilitating the convergence to optimal strategies in dynamic environments. The TD update rule captures the agent's incremental learning process, as it systematically enhances its action choices based on immediate rewards and anticipated future outcomes.

Algorithm 1 Q-learning

Precondition: α - learning-rate, γ - discount factor, λ - decay rate

Initialize $Q(s,a)$ to 0 $\forall s,a$

for each episode **do**
  $s \leftarrow s_0$
  while s in not terminal **do**
    $a \leftarrow \pi(s)$ ▷ according to the policy (e.g. epsilon-greedy)
    play action a and get reward r and next state s'
    $a'^* \leftarrow max_{\tilde{a}}(Q(s',\tilde{a}))$
    $\delta \leftarrow Q(s,a) - r - \gamma Q(s',a'^*)$ ▷ error
    $Q \leftarrow Q - \alpha * \delta$ ▷ update Q
    $s \leftarrow s'$ ▷ update state
  end while
end for

Figure 2.2: Pseudocode Q-Learning.

Universal Value Function Approximators (UVFA) represents an advancement of the conventional Q-learning algorithm, introducing a novel dimension by tethering the Q function and policy to the prevailing goal.

In UVFA, the Q function and policy become inherently linked with the current goal, extending their influence beyond the traditional scope. This expansion necessitates an adjustment to the update rule, now represented as:

$$Q(s,a,g) = Q(s,a,g) - \alpha.\big(Q(s,a), g - r - \gamma Q(s_0, a_0{}^*, g)\big) \tag{2.8}$$

Here, α mirrors the learning rate, dictating the rate at which the Q function adapts. The incorporation of the goal in the Q function and policy imbues UVFA with a heightened adaptability to dynamic multi-goal environments. This framework enables the agent to seamlessly align its actions with the current goal, optimizing its decision-making in scenarios characterized by shifting objectives.

2.3 DEEP REINFORCEMENT LEARNING

Traditional RL algorithms adopt a tabular structure, wherein all values are stored in tables. This approach, while straightforward, bears notable drawbacks. Primarily, its application is confined to problems characterized by a modest count of states and actions. In real-world scenarios, the sheer scale of the state space often renders storage infeasible for regular computers. Moreover, even with access to extensively capacious computing resources, the time required to traverse all states becomes impracticable. This renders the tabular approach unfit for handling complex environments.

Another limitation lies in the inability of tabular algorithms to harness the similarities between states and share acquired knowledge. This undermines the potential efficiency gains that could arise from recognizing patterns and associations in the data.

To surmount these limitations, a widely adopted strategy involves replacing the tabular structure with function approximators. These approximators learn to map the features that define the environment's state to the values of an approximated function. A prevailing choice for function approximation in reinforcement learning is the utilization of deep neural networks. The appeal of deep neural networks stems from their capability to approximate non-linear functions and extract pertinent features from raw inputs. This proficiency empowers them to generalize effectively across unseen states, transcending the limitations of tabular methods.

By employing deep neural networks as function approximators, RL algorithms attain the agility to operate in high-dimensional state spaces, paving the way for the practical application of reinforcement learning in a multitude of complex, real-world scenarios.

2.3.1 Deep Q-Network

The fusion of Q-learning with function approximators has been a subject of exploration over the past decades, but its success has been hampered by the challenge of unstable learning. However, in 2015, a pivotal breakthrough occurred when a team of researchers at DeepMind introduced a pioneering algorithm known as Deep Q-Network (DQN) [13]. This algorithm ingeniously merged Q-learning with neural networks, yielding remarkable achievements in mastering Atari games.

DQN's innovation lay in its capacity to process raw pixels of the game as inputs, eliminating the need for hand-crafted features. This end-to-end architecture was transformative, enabling a single algorithm to learn and excel across multiple games. The neural network's outputs provided the estimated values for each potential action, while its intrinsic capabilities empowered it to autonomously extract pertinent features.

The network architecture, as depicted in Figure 2.3, underscores the design's sophistication. Subsequent to the estimation of all available actions, the algorithm leverages an epsilon-greedy policy to select an action. Addressing the aforementioned instability concerns, the DeepMind team introduced two novel mechanisms: experience replay and frozen target network. Experience replay alleviated instability by introducing a memory buffer that stored past experiences. This buffer, comprised of a diverse range of experiences, facilitated a more stable learning process by reducing the correlation between consecutive updates. The frozen target network approach addressed another challenge by introducing a separate target network that was periodically updated. This separation between the online and target networks prevented the algorithm from chasing a moving target and stabilized the learning process.

The emergence of DQN heralded a new era in reinforcement learning, showcasing the remarkable potential of combining Q-learning with neural networks and paving the way for further advancements in training agents to excel in complex environments.

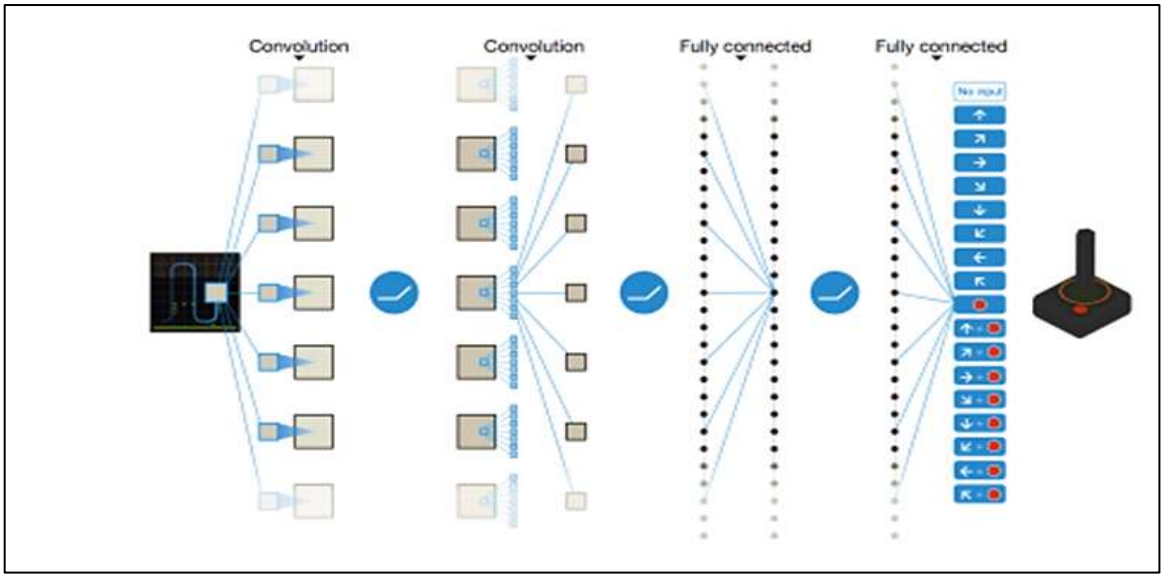

Figure 2.3: The Final Eight Frames Of The Game Are Used As The Network's Input.

These eight frames are then passed via two convolutional layers to analyse the visual data and two fully connected layers for global reasoning. The network's output is an assessment of the value of each potential action.

Experience replay is a pivotal concept aimed at enhancing the stability and efficiency of training in reinforcement learning algorithms. The central premise involves storing the agent's experiences—comprising the state St, action At, reward Rt, and subsequent state St+1—in a dedicated buffer. During each training iteration, a mini-batch of experiences is uniformly sampled from this buffer and utilized for stochastic gradient descent (SGD) updates in the neural network.

Neural networks necessitate independent data to prevent overfitting to sequential patterns, especially in scenarios where frames exhibit high correlations, as seen in game environments. Experience replay mitigates this concern by mitigating the correlation between samples, thereby enabling the network to learn without being excessively influenced by recent experiences. This technique not only fosters robust learning but also leverages the reuse of older experiences, contributing to smoother learning trajectories and greater sample efficiency. In the DQN algorithm, the Mean Squared Error (MSE) loss function is harnessed. This loss function hinges on the squared Temporal Difference (TD) error—a cornerstone of the original Q-learning algorithm. The loss function is computed using the equation:

$$L_i(\theta_i) = \mathbb{E}_t[(R_t + \gamma max_a\big(Q(S_{t+1}, a, \theta_i)\big) - Q(S_t, A_t, \theta_i))^2] \quad (2.9)$$

This equation also introduces the concept of a "frozen target network." The DQN algorithm employs two networks, each with an identical architecture but distinct weight values. These networks are denoted as the Q-network with weights θ and the target network with weights θ-. The Q-network is updated using the loss term from equation 2.9, while the target network remains static and is updated every C timesteps. During these updates, the parameters of the Q-network are copied to the target network, effectively synchronizing their values. This mechanism contributes to the stabilization of learning, attenuating policy oscillations and promoting more consistent learning outcomes.

Algorithm 2 Deep Q-Network

Input: the pixels and the game score, a preprocessing map ϕ

Output: Q action value function

Initialize replay buffer $\mathcal{D}$ to capacity N

Initialize action-value function Q with random weights θ

Initialize target action-value function $Q(\cdot, \theta^-) = Q(\cdot, \theta)$

for each episode **do**

  $s \leftarrow s_0$

  while s in not terminal **do**

  Play

    $a \leftarrow \begin{cases} \text{random action} & \text{with prob } \epsilon \\ argmax_a(Q(\phi(s), a|\theta)) & \text{otherwise} \end{cases}$

    Execute action a and observe reward r and next state s'

    Store transition $(\phi(s), a, r, \phi(s'))$ in $\mathcal{D}$

    $s \leftarrow s'$

  Train

    Sample random minibatch B of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$

    Set $y_j = \begin{cases} r_j & \text{episode terminated at j+1} \\ r_j + \gamma\, max_{a'}(Q^-(\phi_{j+1}, a'|\theta^-)) & \text{otherwise} \end{cases}$

    Perform a gradient descent step on $(y_i - Q(\phi_i, a_j|\theta))^2$ on θ

    Every C steps set $\theta^- \leftarrow \theta$

  end while

end for

Figure 2.4: Pseudocode Deep Q-learning.

2.3.2 Deep Deterministic Policy Gradient

In scenarios where the action space is continuous, the conventional architecture of DQN becomes impractical. This shift occurs because identifying the optimal action for a particular state transform into a separate optimization problem, intensifying computational complexity. To address this challenge, the Deep Deterministic Policy Gradient (DDPG) algorithm, developed by Google DeepMind, emerges as a solution [14]. DDPG circumvents the issue by implementing a distinctive architecture, often referred to as "actor-critic."

The actor-critic architecture employs two distinct neural networks working in tandem: the actor network and the critic network. The actor network is responsible for predicting the best possible action for each state, effectively approximating the policy. This network's primary objective is to guide the agent in making decisions that optimize long-term rewards. On the

other hand, the critic network functions as an evaluator, estimating the value of different states or state-action pairs. This estimation aids the agent in gauging the desirability of specific actions within a given state.

The synergy between the actor and critic networks enables DDPG to navigate the intricacies of continuous action spaces efficiently. The actor network refines the agent's policy, directing it toward rewarding trajectories, while the critic network supplies valuable insights into the quality of chosen actions. By capitalizing on this actor-critic architecture, DDPG opens the door to effective reinforcement learning in domains characterized by continuous action spaces. The actor-critic architecture, as illustrated in Figure 2.10, constitutes a fundamental framework within deep reinforcement learning:

Actor: The actor network, denoted as μ, is responsible for selecting actions. Given the environment's state St as input, the actor network produces the chosen action μ(St|θμi). The actor's weights θμ are optimized through gradient ascent over the Q value—a strategy aimed at maximizing the Q value. Equations 2.22 and 2.23 outline the actor's loss function and update rule, with the negative loss designed for maximization.

Actor's Loss Function: $$L_i(\theta_{\mu_i}) = E_t[Q(S_t, \mu(S_t|\theta\mu_i)|\theta\mu_i)]) \quad (2.10)$$

Actor's Update Rule: $$\theta_{\mu_{i+1}} = \theta_{\mu_i} + \alpha_a \nabla_{\theta_{\mu_i}} L_i(\theta_{\mu_i}) \quad (2.11)$$

Critic: The critic network evaluates the Q value of the state and action chosen by the actor. Taking the state St and action μ(St|θμi) as input, the critic network estimates their Q value. Similar to the Q network, the critic network is trained. Equations 2.12 and 2.13 define the critic's loss function and update rule.

Critic's Loss Function: $$L_i(\theta_{Q_i}) = E_t\left[\left(\left(R_t + \gamma Q\_(St+1, \mu\_(St+1|\theta_{\mu\_i})|\theta_{Q\_i})\right) - Q(St, At|\theta_{Q_i})\right)^2\right] \quad (2.12)$$

Critic's Update Rule: $$\theta_{Q_{i+1}} = \theta_{Q_i} + \alpha_a \nabla_{\theta_{Q_i}} L_i(\theta_{Q_i}) \quad (2.13)$$

To address the challenge of unknown Q values and derivatives, a second network—the critic network—is introduced. The critic network evaluates the chosen state and action, estimating their Q value. Its training closely resembles that of the Q network.

The actor's loss function leverages the critic's estimations to compute derivatives. By applying the chain rule, the gradient of the actor's loss can be expressed. Equation 2.14 outlines this gradient.

The algorithm DDPG employs experience replay similarly to the approach presented in DQN. Moreover, DDPG introduces an alternative version of the frozen target network concept known as "soft target update." Unlike copying weights every C timesteps, the target network is updated continually, gradually converging towards the original parameters. This update process is represented by the equation:

$$\theta^{-} = \tau\theta + (1 - \tau)\theta^{-} \tag{2.14}$$

Here, the parameter τ lies within the range of [0, 1] and determines the tracking speed. A smaller value of τ results in a slower convergence of the target weights towards the original weights. To introduce exploration within a continuous action space, noise N is added to the output of the actor network $\mu(s|\theta\mu)$. This noise is generated using an Ornstein-Uhlenbeck process [15].

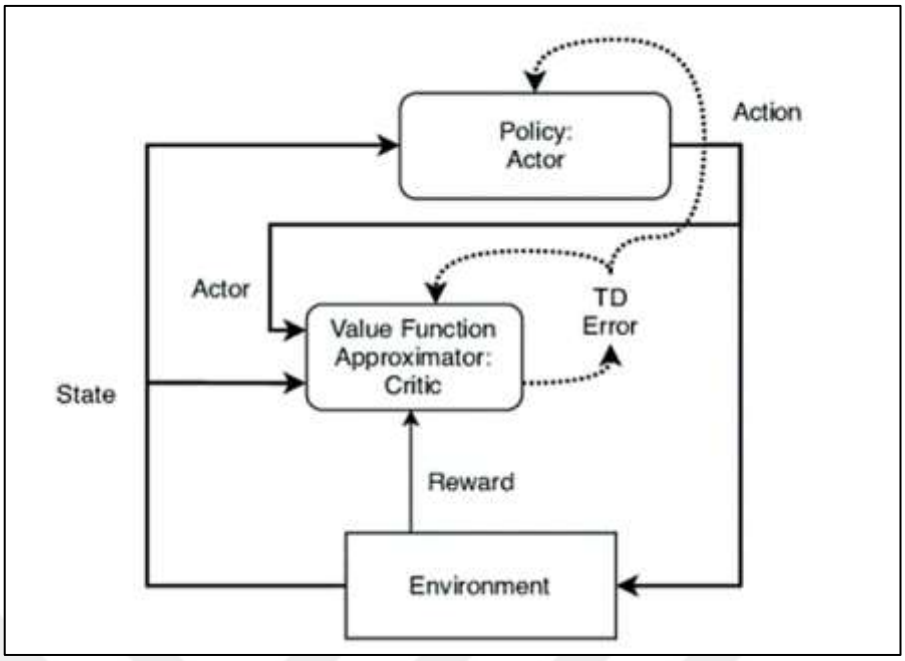

Figure 2.5: Given The Situation, The Actor Chooses A Course Of Action.

The critic assesses the situation and what was done. The critic's assessment is used to train the actor, and the critic is trained utilizing the TD mistake.

Algorithm 3 Deep Deterministic Policy Gradient

Input: the state of the environment **Output:** action to perform

Initialize replay buffer $\mathcal{D}$ to capacity N

Randomly initialize critic $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$

Initialize target network Q^- and μ^- with weights $\theta^{Q^-} \leftarrow \theta^Q$, $\theta^{\mu^-} \leftarrow \theta^\mu$

1: **for** each episode **do**

2: Initialize a random process $\mathcal{N}$ for action exploration

3: $s \leftarrow s_0$

4: **while** s in not terminal **do**

Play

5: $a \leftarrow \mu(s|\theta^\mu) + \mathcal{N}$ according to current policy

6: Execute action a and observe reward r and next state s'

7: Store transition (s, a, r, s') in $\mathcal{D}$

8: $s \leftarrow s'$

Train

9: Sample random minibatch B of N samples (s_j, a_j, r_j, s_{j+1}) from $\mathcal{D}$

10: Set $y_j = \begin{cases} r_j & s_{j+1} \text{ is a termination sta} \\ r_j + \gamma Q^-(s_{j+1}, \mu^-(s_{j+1}|\theta^{\mu^-})|\theta^{Q^-}) & \text{otherwise} \end{cases}$

11: Update critic by minimizing the loss $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$

12: Update the actor policy using the sampled policy gradient:

$\nabla_{\theta^\mu} L = -\frac{1}{N}\sum_i \nabla_{\mu(s_i)} Q(s_i, \mu(s_i)|\theta_Q) \nabla_{\theta^\mu} \mu(s_i|\theta^\mu)$

13: update the target networks:

$\theta^{Q^-} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q^-}$

$\theta^{\mu^-} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu^-}$

14: **end while**

15: **end for**

Figure 2.6: Pseudocode DDPG.

2.3.3 Prioritized Experience Replay

Prioritized Experience Replay (PER) [16] introduces a refinement to the experience replay mechanism inherent in DQN and DDPG. Central to PER's advancement is the strategic prioritization of experiences, spotlighting those that contribute richer insights to the agent's learning process. Each experience is endowed with an additional parameter ω_i, denoting its priority. Consequently, experiences with higher ω values enjoy elevated probabilities P_i during experience replay, as expressed in equation 2.15.

$$P(i) = \frac{\omega_i^{\alpha}}{\sum_k \omega_k^{\alpha}} \tag{2.15}$$

The exponent α within the equation governs the degree of prioritization, with α = 0 representing uniform priority distribution.

An effective gauge of significance is the TD-error, serving as an indicator of the learning potential an experience holds. A smaller TD error implies heightened familiarity with the relevant state-action pairs, rendering experiences with higher TD errors more conducive to the agent's learning process.

2.3.4 Reinforcement Learning for Sparse Reward Function

Reward functions can be categorized into two distinct types: Dense rewards and Sparse rewards, the latter of which is also referred to as binary rewards. Dense reward functions offer a straightforward link between improved policies and higher returns, ensuring the agent's awareness of its progress. Conversely, sparse reward functions exhibit a different dynamic, with most policies yielding identical returns. Only once a certain performance threshold is exceeded does the agent discern its advancement. In instances of sparse rewards, the reward function typically assumes a value of 0 upon target achievement and -1 otherwise. A quintessential example of a binary reward context is evident in grid-world games, where each step incurs a penalty until the target state is reached. Notably, learning from sparse rewards presents a renowned challenge in reinforcement learning, demanding extensive exploration for goal attainment and signal acquisition. Subsequent sections will introduce two algorithms adept at grappling with sparse reward environments.

2.3.5 Demonstration-Initialized Rollout Worker

Effectively tackling sparse reward learning necessitates substantial exploration. A viable strategy to alleviate this exploration requirement is the utilization of expert demonstrations coupled with supervised learning techniques. However, this approach harbors two critical drawbacks. Primarily, an abundance of demonstrations is essential for proficient agent training. Secondly, by mimicking the expert's actions, the agent's skill ceiling aligns with that of the expert, limiting further improvement. To address these limitations, the Demonstration-Initialized Rollout Worker algorithm was introduced [17]. Employing a form

of dynamic programming, this algorithm streamlines the learning process. Leveraging a single expert demonstration, the algorithm aids the agent's training. During each episode, the game initializes from a state along the given trajectory. Initial episodes begin proximate to the termination state, denoted at step t within a span of T steps. The agent's reinforcement learning is employed to teach it how to attain the target from this initialized state. Should the agent's performance match or surpass that of the expert, incremental updates shift the initial state closer to the actual inception point. For a detailed outline, refer to algorithms 4 and 5. This "reverse curriculum" expedites the agent's proficiency in reaching the target state, potentially surpassing expert levels. Notably, Demonstration-Initialized Rollout Worker has certain drawbacks. It necessitates expert demonstrations for effective learning and is applicable solely to single-target environments.

Algorithm 4 Demonstration-Initialized Rollout Worker

Precondition:

- an expert's demonstration $\tau = \{(\bar{s}_i, \bar{a}_i, \bar{r}_i, \bar{s}_{i+1}, \bar{d}_i)\}_{i=0}^{T}$
- D - number of possible starting points
- M - number of rollouts in batch

Initialize max starting point $i_{max} \leftarrow T$
while True **do**
  Get latest policy $\pi(\theta)$ from optimizer
  Get latest reset point i_{max} from optimizer
  Initialize success counter $\mathcal{W} = 0$
  Initialize batch $\mathcal{D} = \{\}$
  for *rollout* $\leftarrow 1, M$ **do**
    Sample starting point i by sampling uniformly from $\{i_{max} - D, ..., i_{max}\}$
    $t \leftarrow 0$
    Initialize state s_t to state s_i in the trajectory τ
    done $\leftarrow$ *False*
    while not done **do**
      Sample action $a_t \sim \pi(s_t)$
      Take action a_t in the environment
      Receive reward r_t, next state s_{t+1} and done signal d_t
      done $\leftarrow d_t$
      Add date $\{s_t, a_t, r_t, s_{t+1}, d_t\}$
      $t \leftarrow t + 1$
      if if *done* **then**
        if $\Sigma_{i=0}^{t} \gamma^i \cdot r_i \geq \Sigma_{i=0}^{t} \gamma^i \cdot \bar{r}_i$ **then** ▷ As good as demo
          $\mathcal{W} \leftarrow \mathcal{W} + 1$
        end if
      end if
      $t \leftarrow t + 1$
    end while
  end for
  Send batch $\mathcal{D}$ and counter $\mathcal{W}$ to optimizer ▷ see algorithm 5
end while

Figure 2.7: Pseudocode Demonstration-Initialized Rollout Worker.

Algorithm 5 Optimizer

Precondition:

- i_{max} - max starting point
- $\mathcal{D}$ - batch of agent's experience
- $\mathcal{W}$ - number of seccessful rollouts
- Δ - starting point shift size, ρ - success threshold, θ - agent's parameters, $\mathcal{A}$ - learning algorithm, D - number of possible starting points

1: **if** $\frac{\mathcal{W}}{\mathcal{D}} > \rho$ **then** ▷ The agent is successful sufficiently often
2: $\quad i_{max} \leftarrow i_{max} - \Delta$
3: **end if**
4: $\theta \leftarrow \mathcal{A}(\theta, \mathcal{D})$

Figure 2.8: Pseudocode Optimizer.

2.3.6 Hindsight Experience Replay

Numerous past achievements in RL have been oriented towards specific objectives, such as "checkmating the opponent in chess." Yet, real-world scenarios often demand more diverse accomplishments. These situations require the agent to achieve a multitude of different goals. The agent must discern the current goal within the game and select actions accordingly. Such scenarios are referred to as multi-goal tasks. The extension of Deep Deterministic Policy Gradient (DDPG) to multi-goal tasks utilizes Universal Value Function Approximators (UVFA) [18]. The core concept behind UVFA involves enriching action-value functions and policies with goal states. This enhancement ensures that each transition encapsulates the intended goal alongside the standard elements. This augmentation enables the generalization of learning not only across states but also across goals, facilitated by neural networks as function approximators.

In multi-goal tasks characterized by sparse rewards (i.e., 0 for achieving the goal and -1 for all other states), mastering the task and making any headway poses a significant challenge. Hindsight Experience Replay (HER) [19], an algorithm devised by OpenAI, serves as a solution to this conundrum. Traditional algorithms perceive failures as mere setbacks, learning that a given trajectory did not yield success. Consequently, discerning what constitutes a genuinely useful strategy often requires serendipitous goal attainment.

HER tackles this challenge by transforming failures into successes through the lens of alternative or virtual goals (refer to figure 2.9). HER incorporates the UVFA framework and supplements the experience replay buffer with additional transitions involving virtual goals. This mechanism enables the agent to learn from failures by extrapolating to actual objectives. Empirical evidence underscores that HER substantially enhances performance across various intricate simulated robotic environments. Each transition <ST, AT, Rt, St+1, G> within the trajectory is augmented in the buffer with an alternative goal, projected for achievement in the future.

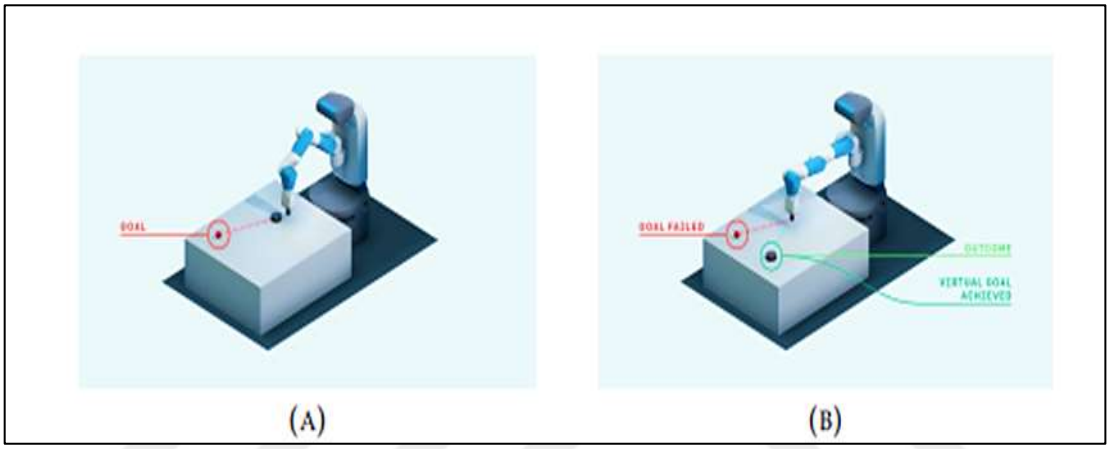

Figure 2.9: HER Makes Use Of The Openai Push Environment.

The puck's first goal target is seen in (A). (B) demonstrates the creation of a virtual objective following the failure of the initial job.

2.4 RRL AND GAMES

2.4.1 Statistics

In order to substantiate the primary assertion, we conducted a comprehensive exploration of various scholarly databases, namely Scopus (a meticulously curated abstract and citation repository, accessible at https://www.scopus.com/ as of 20th December 2022), Google Scholar (a prominent search engine catering to scholarly literature, available at https://scholar.google.com/ as of 20th December 2022), and Dimensions (a repository interlinking research data, accessible at https://www.dimensions.ai/ as of 20th December 2022). We meticulously examined publication and literature records for each year, employing targeted terms and keywords such as "deep reinforcement learning," "games,"

"reinforcement learning," and "deep learning." These specific inquiries were carried out to gain a comprehensive perspective on the subject matter's scholarly activity and evolution. A visual representation of our findings is showcased in the corresponding graph, depicted as Figure 2-10.

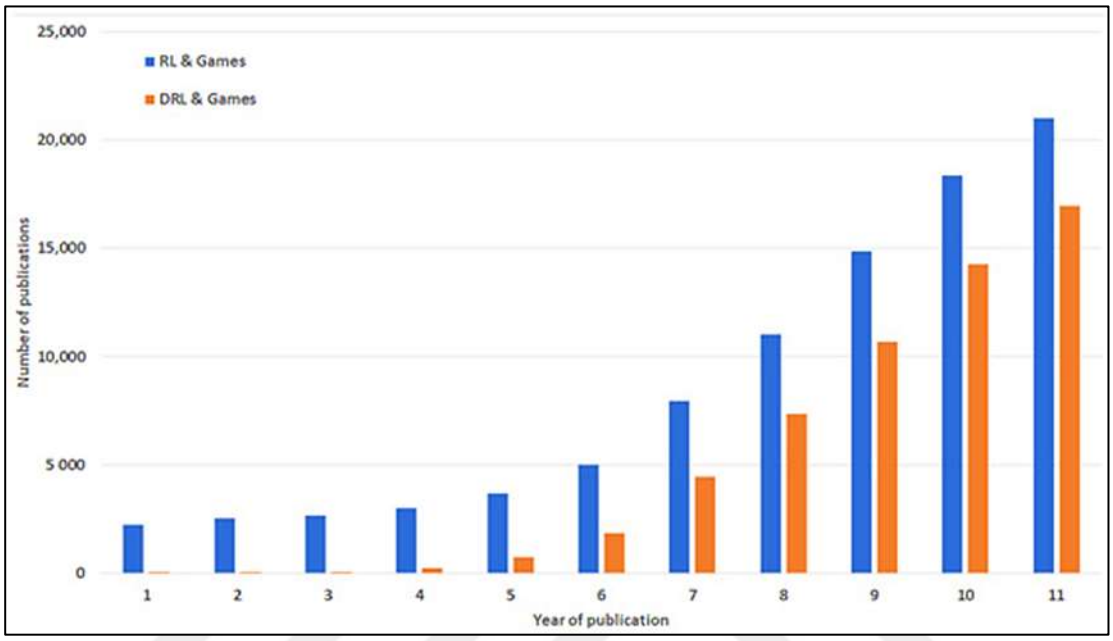

Figure 2.10: DRL And Games Publications Per Year.

To provide more granular insights, we delved into the specific statistics regarding publications within the realm of "reinforcement learning" and "games." Our analysis encompassed the time span from 2012 to 2022, and our findings revealed that the amalgamation of the terms "reinforcement learning" and "games" yielded a total publication count of 92,367. Further scrutinizing the intersection of "deep reinforcement learning" and "games" for the same period, we identified a publication count of 56,613.

For the subsequent aspect of our investigation, spanning 2012 to 2022, we meticulously combed through publication data from Scopus, Google Scholar, and Dimensions, with our focus directed towards the broader domains of "deep reinforcement learning" and "reinforcement learning." Within the context of the keywords "reinforcement," "learning," and "games," our exploration yielded a total of 92,367 findings within the 2012–2022 timeframe. Notably, a significant proportion of these findings—73,245 to be precise—materialized between 2018 and 2022, representing nearly 73% of the cumulative publications.

In summary, the recent trajectory in research related to reinforcement learning in the context of games has prominently embraced approaches rooted in deep learning (DL). Specifically, a range of methodologies grounded in genetic programming (GP) have demonstrated competitive outcomes across diverse platforms such as ALE, ViZDoom, StarCraft, and Dota 2. Intriguingly, this paradigm shift also indicates that the computational overhead associated with deploying DL solutions for these gaming domains is substantially higher when compared to GP alternatives. This discrepancy arises due to conventional methods bearing notable drawbacks in terms of memory utilization and computational complexity. DL, on the other hand, has been able to surmount these limitations owing to its prowess in handling multifaceted data dimensions and its inherent scalability. The same visual representation in Figure 2.10 aptly captures the research activity encompassing the "deep," "reinforcement," "learning," and "games" keywords within the 2012–2022 timeframe. Additionally, the depiction includes the research engagement centered around the keywords "reinforcement," "learning," and "games" within the aforementioned period, contextualized relative to the count of publications [20].

2.4.2 Trends

The extensive body of literature dedicated to the domain of deep reinforcement learning has unveiled distinct patterns that exhibit correlations with specific game titles [21]. In the case of certain earlier game titles, the research outcomes were generated through the application of "deep reinforcement learning" or "reinforcement learning" methodologies. Notably, the realm of ATARI games has particularly captivated the attention of the scientific community, drawing significant interest. Additionally, the landscape encompasses board games, with a notable portion centering around the game of "Chess," alongside a selection of more contemporary multiplayer online battle arena (MOBA) and strategic games.

Subsequent sections of this study delve into an in-depth analysis of the integral role that each component of deep reinforcement learning plays within the context of various game titles. Consequently, this investigation keenly examines six distinct game "trends," with particular emphasis placed on the prominence of ATARI games [22].

2.4.3 Transition to DRL

Human beings possess a remarkable ability to discern patterns and execute intricate tasks within remarkably brief timeframes. Within this framework, they adeptly decipher optical data, translate it into meaningful constructs, process the information, and ultimately arrive at informed judgments. Over the preceding decades, the realm of scientific inquiry has been fervently dedicated to emulating these critical human faculties. Even prior to the turn of the century, scholarly investigations indicated the potential of intelligent algorithms that could learn and adapt from data to address these challenges [23]. The prevailing consensus points toward a mechanism akin to the workings of the human brain and its central nervous system. These algorithms, often referred to as "brain modeling," demonstrate an adaptable nature across diverse problem domains. However, their progress was restrained by the limited computational power available during that era.

During its zenith, the collection of theories, methodologies, and algorithms rooted in this concept was collectively known as "neural networks," and it stood as the forefront choice for a multitude of problem-solving contexts [24]. Over the past decade, the advent of vast and parallel computing capabilities has empowered us to confront immensely complex foundational issues within compressed timeframes, often surpassing human performance. Within this evolving landscape, reinforcement learning has embraced this newfound potential, transcending the confines of unsupervised learning and evolving into an innovative realm known as "deep reinforcement learning."

2.4.4 Deep Q–Network in ATARI

As documented in the publication by DeepMind in February 2015 [25], the fusion of deep neural networks (DNNs) with conventional RL techniques was expounded upon. Notably, this amalgamation yielded impressive outcomes within games featured on the ATARI console. The pinnacle of this achievement was exemplified by achieving high scores in renowned games such as breakout, as depicted in Figure 2.11.

Figure 2.11: ATARI Pinball Game High Scores, With Double DQN [26].

Numerous games have witnessed achievements reaching superhuman-level performance. The Deep Q-Network (DQN) employs deep convolutional neural networks (CNNs) to represent states, a technique that mitigates the complexity inherent in real-world problems, as explored in the earlier sections. The network architecture is visually depicted in Figure 2-12. Specifically, the DQN architecture encompasses a 4-dimensional input image measuring 84x84, encompassing the RGB channels of the game's present frame along with the luminance channel. Sequentially, this input layer is succeeded by three convolutional layers: the first employs 32 filters of size 8x8 with a stride of 4, the second uses 64 filters of size 4x4 with a stride of 2, and the third employs 64 filters of size 3x3 with a stride of 1. Notably, the initial two convolutional layers employ a rectifier nonlinearity, while the third layer is followed by a simple rectifier. The terminal layers consist of a densely connected layer with 512 units, followed by the output layer containing a solitary output for each of the valid actions, ranging from 4 to 18, including the option of "no action."

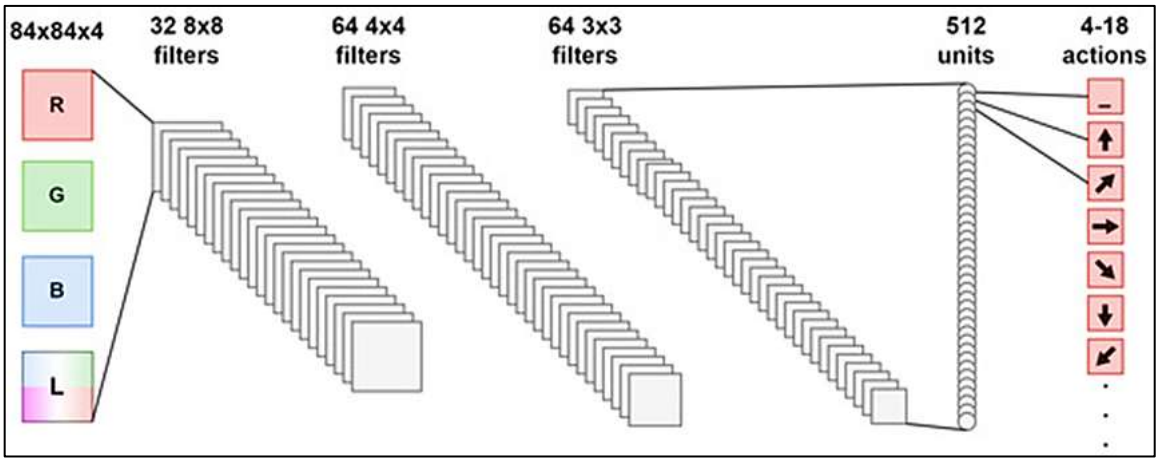

Figure 2.12: Adapted From Deepmind's 2015 Nature Article [27], The ATARI Deep Q-Network Architecture.

The ultimate aim of the agent in the interactive environment is to select actions that will yield the highest cumulative rewards, operationalized through the DNN approximating a specific energy evaluation function [28]. Contrary to traditional understanding, RL can be inherently volatile, particularly when nonlinear function approximations like neural networks (NNs) are employed to forecast the energy evaluation function. Factors contributing to this instability include sequential observation correlations, along with a tight link between temporal-difference (TD) targets and energy values [29]. To mitigate these issues, DeepMind introduced the Deep Q-Network (DQN), a specialized variation of the Q-learning approach [30].

DQN employs two main strategies for stabilization. Firstly, it adopts an experience replay mechanism, which randomly selects observations for learning, thereby negating the correlation effects within a sequence of observations [30]. Secondly, DQN employs a periodic update strategy for the Q-value function directed towards the TD target [31,32]. This is achieved by intermittently cloning the active Q-network parameters into a secondary Q-network used for TD target calculations [33].

For implementation, the neural network represented in Figure 2-12 is used to approximate the energy evaluation function $(Q(s,a|\theta_i))$, where (θ_i)denotes the iteration-specific parameters (weights) of the Q-network [44]. The agent's experiences are categorized into tuples $(e_t = (S_t, A_t, r_t, S_{t+1}))$to facilitate the experience replay process. The Q-value

function is updated using a stochastic gradient descent algorithm, based on the following loss function:

$$L(\theta_i) = E_{(s,a,r,s')\sim U(D)}\left[\left(r + \gamma \max_{a'} Q\,(s', a'|\bar{\theta}_i) - Q(s, a|\theta_i)\right)^2\right] \quad (2.16)$$

Here, (γ) represents the discount factor, (θ_i) are the parameters of the Q-network in the (i^{th}) iteration, and ($\bar{\theta}_i$) are the parameters used for calculating the TD target in the (i^{th}) iteration [34]. The ($\bar{\theta}_i$) parameters are updated every (C) iterations by copying the parameters from the main Q-network and remain stable between updates [34].

DeepMind leveraged the Deep Q-Network (DQN) methodology to train an AI agent on ATARI games using purely raw video data, rewards, termination flags, and possible actions [34]. In stark contrast to conventional methods that might incorporate game-specific insights, this approach mimicked the typical user's learning experience, supplying the neural network agent with the same basic elements: visual data, incentives, game-ending signals, and potential moves. The overarching objective was to engineer a universal neural network agent capable of learning to play multiple ATARI games effectively. Remarkably, this agent outperformed all other competing algorithms in six out of the seven games it was trained on. Moreover, it set new human-level performance records in three of these games (as visualized in Figure 2-13) [30].

Figure 2.13: The First Video Games That Deepmind Technologies Learned.

Beam Rider, Breakout, Enduro, Pong, Qbert, Seaquest, and Space Invaders are listed in order from left to right and top to bottom. All photos were shot with the Python module for OpenAI Gym [36].

2.4.5 Double Q-Learning

While the advent of Double Q-learning and DQN were undeniably groundbreaking developments in the realm of RL, they were not without their limitations. Numerous scholars have proposed refined versions of these algorithms to tackle inherent flaws and limitations [37]. Not resting on its laurels, DeepMind itself has also iteratively improved both the deep Q-learning algorithm and the corresponding neural network architecture. These enhancements led to even more remarkable performance benchmarks, including notably elevated scores in ATARI games like pinball, as illustrated in Figure 2-11.

Hado van Hasselt highlighted that the Q-learning algorithm encounters difficulties in certain stochastic environments [38]. This is primarily due to the overestimation of energy values when using $\max Q\,(s', a)$. A proposed remedy for this issue involves maintaining two separate Q-value functions, Q_A and Q_B, which are mutually updated for the subsequent state. This update process starts by identifying the energy $a*$ that maximizes $Q_A$in the subsequent state:

$$[Q(s, a) = \max Q\,(s', a)] \tag{2.17}$$

Subsequently, the value of $Q_B(s, a)$ is computed based on $a*$, enabling the update of Q_A as $Q_A(s, a)$. This innovative method can be integrated with DQN to enhance its performance by utilizing the following loss function:

$$\left[R_{t+1} + \gamma_{t+1} q_\theta\left(S_{t+1}, \arg\max q_\theta\,(S_{t+1}, a')\right) - \widetilde{q_\theta}(S_t, A_t)\right]^2 \tag{2.18}$$

This adjustment has been demonstrated to curtail the overestimation of energy values in the DQN algorithm, thereby bolstering its overall efficiency.

2.4.6 Prioritized Experience Replay

Tom Schaul introduced the notion of "priority experience replay" to make the sampling of experiences in DQN more meaningful [39]. In this method, experiences with significant

differences between the estimated Q-value and the Temporal Difference (TD) target are selected more frequently for training. The selection probability (p) is determined based on the last observed absolute error:

$$p_t = \left| R_{t+1} + \gamma_{t+1} \max_{a'} q_\theta\left(S_{t+1}, a'\right) - q_\theta(S_t, A_t) \right| \tag{2.19}$$

where (ω) is a hyperparameter that shapes the distribution.

2.4.7 Dueling Networks

Dueling Networks extend the DQN architecture to decouple the state-value function from the advantage function for each action [20]. Unlike traditional DQN architectures that used a single set of fully connected layers, Dueling Networks employ two separate streams to independently estimate the value and advantage functions. The estimations are then aggregated to output a single Q-value. This architectural choice is designed to better handle situations where the action doesn't significantly affect the outcome. The Q-value is then expressed as:

$$Q(s, a) = A(s, a) + V(s) \tag{2.20}$$

In Dueling Networks, two streams are used:

a. One stream calculates the state value $(V(s))$

b. The other estimates the advantage of taking each action (a) in state (s), denoted $(A(s, a))$

These estimates are then combined in an aggregation layer to produce $(Q(s, a))$.

The Dueling Network model can be described using the following equation:

$$q_\theta(s, a) = v_n\left(f_\xi(s), a\right) + \frac{\sum_{a'} a_\psi\left(f_\xi(s), a'\right)}{N_{\text{actions}}} \tag{2.21}$$

where (ξ, n, ψ) are parameters of the convolutional encoder (f_ξ), the value stream (v_n), and the advantage stream (a_ψ) respectively, and ($\theta = \{\xi, n, \psi\}$).

By decomposing the Q-value into its components, Dueling Networks can be more efficient in situations where the choice of action has minimal impact, thereby focusing on the state value ($V(s)$).

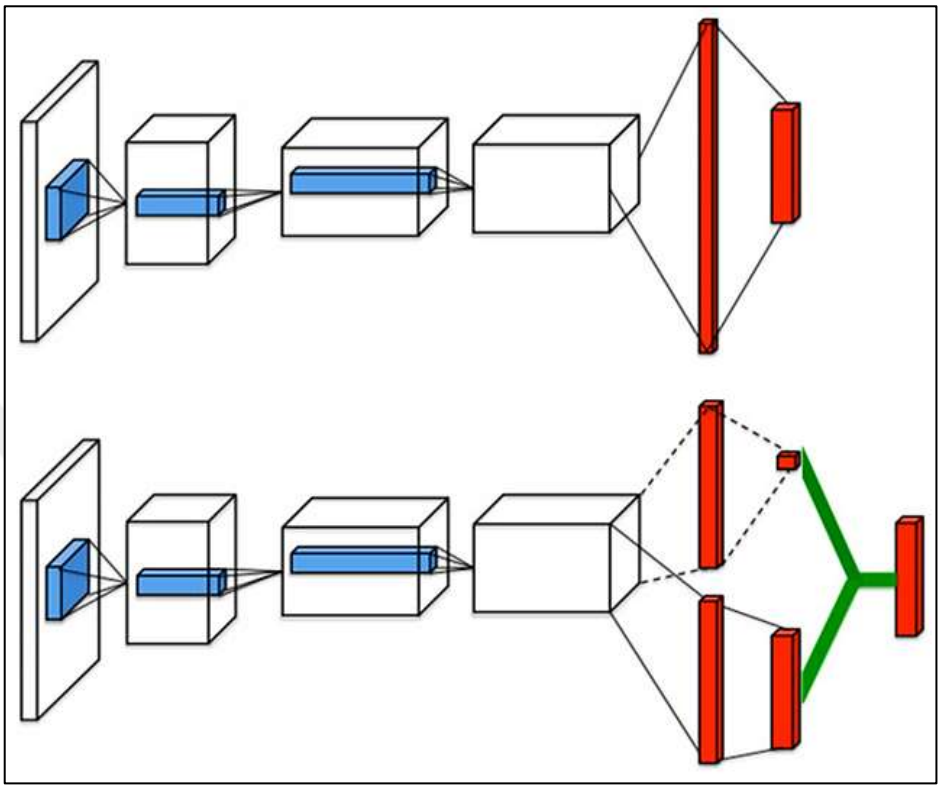

Figure 2.14: Dueling Q–Network Architecture, Deepmind [20].

In summary, DeepMind's implementation of the Dueling Network architecture significantly improved the performance of the DQN algorithm in a wide array of ATARI games. The new architecture excelled particularly in games like Asterix and Atlantis, outpacing previous models like Double DQN. However, it's essential to note that the gains weren't universal. For instance, Dueling DQN performed slightly worse in Bowling, with a top score reduction of approximately 1.89%. Furthermore, there were games, such as Pinball and Freeway, where the new architecture didn't contribute to performance enhancement—in fact, leading to reductions by 72.10% and 100% respectively. Therefore, while the Dueling Network architecture has been generally successful in improving the efficiency of DQN, its efficacy can vary depending on the specific challenges presented by each game.

Figure 2.15: Dueling Q-Networks Outperformed The Double DQN By 1097% In The Asterix Game For The Atari. Using The Openai Gym Python Module, This Image Was Captured [36].

Figure 2.16: Dueling Q-Networks In The Atlantis Atari Game Surpassed The Traditional DQN By 296%. Using The Openai Gym Python Module, This Image Was Captured [36].

In a striking display of its capabilities, the Dueling DQN architecture markedly surpassed the performance of the standard DQN model in specific ATARI games, most notably achieving gains of over 100% in Atlantis, Tennis, and Space Invaders (as depicted in Figure 2-16). However, it's important to consider the unique characteristics and challenges of each game when evaluating these results. Notably, the earlier, less advanced DQ learning algorithm outperformed even the Dueling DQN in certain scenarios. Ms. Pacman serves as a prime example where the older algorithm had the edge. This highlights the fact that while advancements in DQN architectures like the Dueling Network offer significant improvements, their effectiveness can be context-dependent, varying from game to game.

In a notable accomplishment, Microsoft reported in June 2017 that they had achieved the highest possible score in Ms. Pacman, a feat that had eluded both Google's neural networks and various other efforts. Remarkably, this record score of 999,990 had only been previously achieved through cheats. The highest human-scored game, as recorded by Twin Galaxies, is 933,580 by Abdner Ashman.

This breakthrough came at a time when significant strides had already been made in more complex games by DeepMind’s AlphaGo in Go and by Libratus and DeepStack in heads-up no-limit Texas Hold'em poker. The team from Microsoft's Maluuba used a sophisticated approach involving a hybrid reward architecture that combined reinforcement learning with a "divide and conquer" strategy. Over 150 agents were employed, each with a specific task—such as locating a specific pill in the game—which they would then share with a "super-agent." This super-agent would synthesize all of the sub-agents' information and make the final decision for Ms. Pacman's next move.

Interestingly, the optimal performance was achieved when each sub-agent acted purely based on their own specialized function, while the super-agent focused on making the most beneficial move for the group as a whole. This approach was backed by weighted decision-making, emphasizing the importance of each task, as pointed out by Harm Van Seijen [40]. If the super-agent's decisions were solely based on majority vote, it would risk catastrophic outcomes, like running into an enemy ghost.

The system was trained using reinforcement learning, where each action led to either a positive or negative reward, facilitating learning through trial and error. Over 800 million

game frames were used for training. Remarkably, the scoring algorithm was designed in such a way that upon reaching one million points, the score resets to zero, as illustrated in Figure 2-17. This makes Microsoft's achievement all the more impressive, and it stands as a historical milestone in the realm of AI performance in gaming.

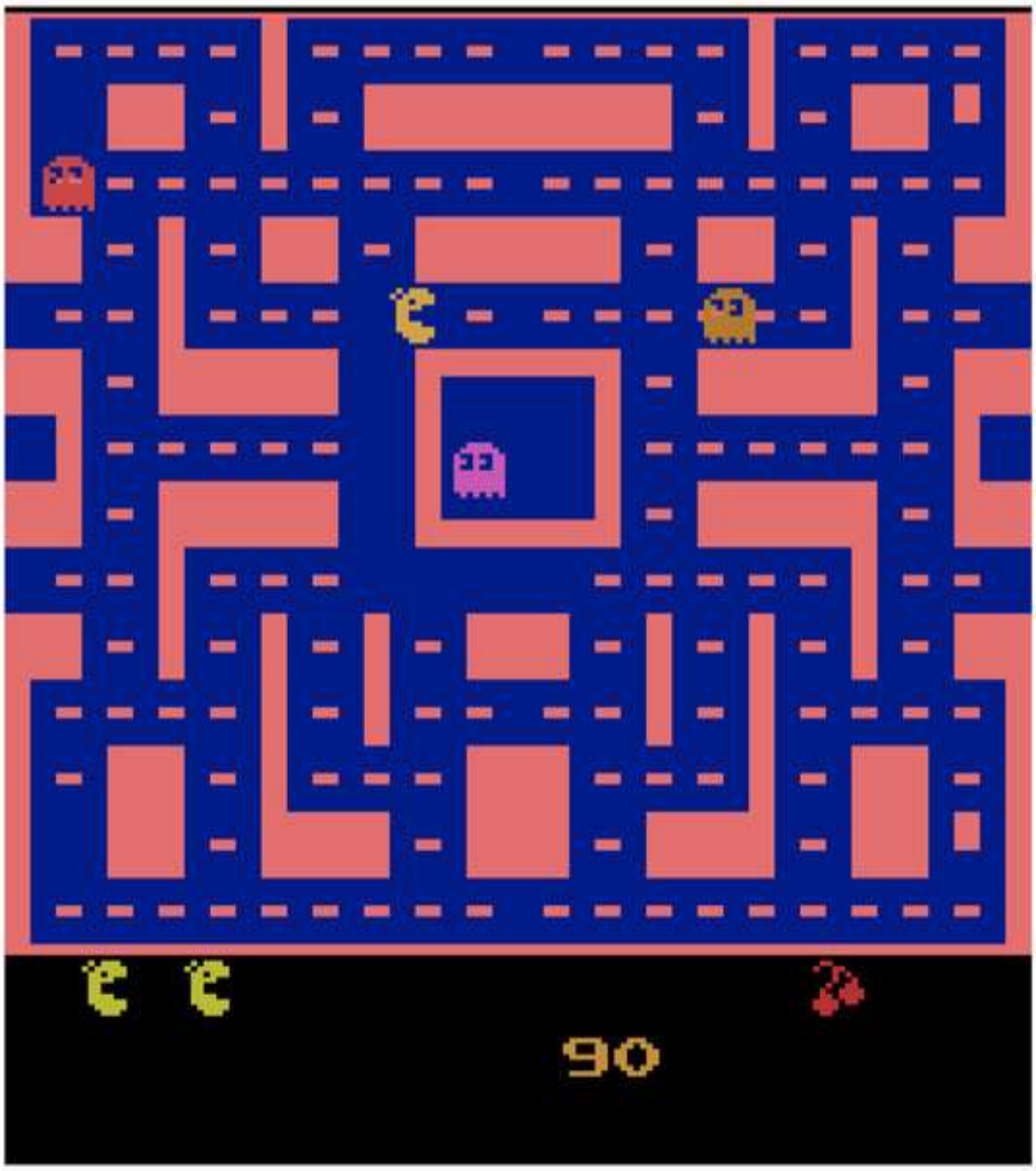

Figure 2.17: Microsoft Outperformed All Previous Human/Computer High Scores In 2017 With A Score Of 999,990 In The Game Ms. Pacman. Using The Openai Gym Python Module, This Image Was Captured [36].

2.5 METAHEURISTIC

2.5.1 Overview

To grasp the essence and functionality of Metaheuristic Algorithms (MH), it's important to first understand their origins. According to Talbi's research [41], optimization techniques can be broadly categorized into two groups. The first group is exact methods, which aim to find the best possible solutions within a specific search space. The second group is approximate methods, which are designed to deliver high-quality solutions within a reasonable computational time frame, making them more practical for real-world applications. Within this category of approximate methods, we find heuristic algorithms, and among them are Metaheuristic Algorithms. These MH algorithms can further be divided into two subcategories: those that focus on optimizing a single solution and those that work on a population of solutions.

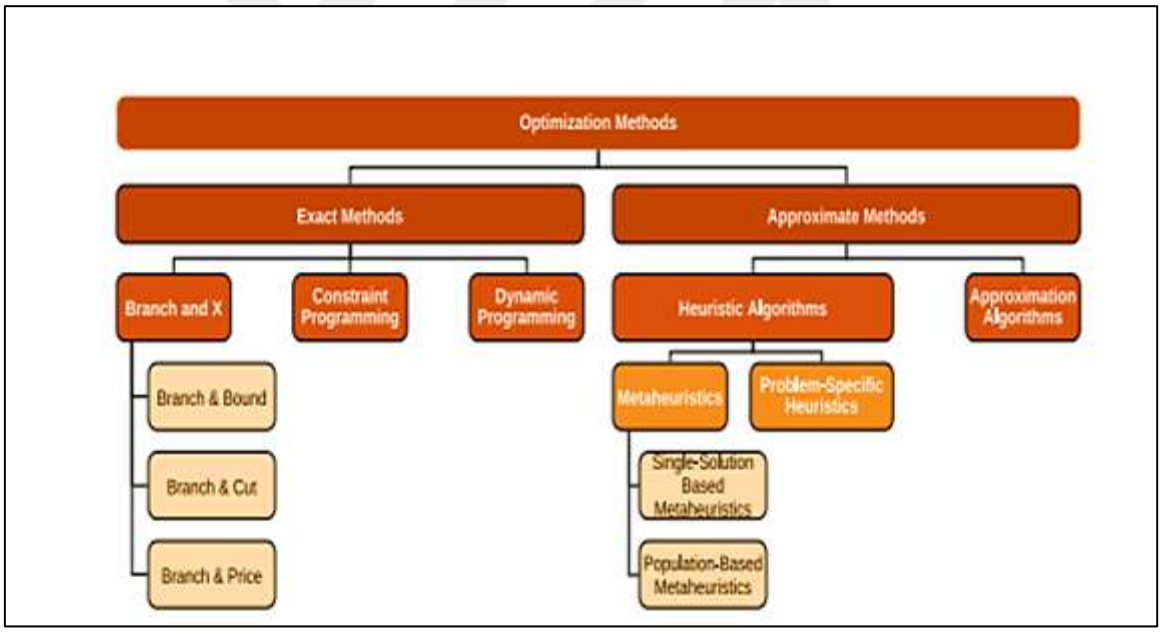

Figure 2.18: Categories and Subcategories of Optimization Methods.

Metaheuristic Algorithms (MH) occupy a specialized niche in the realms of computer science and applied mathematics, intersecting with algorithmic design and computational complexity theory. In essence, an MH is an iterative method geared toward problem-solving. It operates based on specific parameters to guide a "subordinate heuristic" toward solutions

that are closely aligned with the optimal outcome. The method balances both exploration and exploitation effectively within the search space.

Over the last two decades, MH algorithms have proliferated across various interdisciplinary fields, including artificial intelligence, computational intelligence, soft computing, mathematical programming, and operations research. A distinctive feature of many MH algorithms is their tendency to draw inspiration from natural phenomena to tackle intricate optimization challenges [41]. This has led to a diverse array of MH types, each tailored to specific problem domains and exhibiting unique search behaviors.

The categorization of these algorithms often hinges on the source of their inspiration. For instance, some are modeled on evolutionary principles [42], others emulate swarm intelligence [43], some draw from the laws of physics [44], and still, others are inspired by human behavior and social constructs [45]. This wide range of inspirations has given rise to various solution-building methods and search strategies, making MH a highly versatile tool in optimization problems.

2.5.2 Snake Optimization

Snake Optimization [46] is an iterative metaheuristic algorithm designed for solving optimization problems. The algorithm mimics the behavior of snakes and is structured around a series of phases including Initialization, Evaluation, Determination of Phase, Exploration, Exploitation, Laying Eggs (Replacement), and Termination Check. Initially, a population of candidate solutions, represented as male and female snakes, is randomly positioned in the search space. The algorithm also initializes parameters like quality factor (Q) and temperature (Temp). The fitness of each snake is then evaluated based on an objective function.

The algorithm operates in either an exploration or exploitation phase, determined by the quality factor (Q). In the exploration phase, snakes search the space randomly, guided by a leader snake's attractiveness. In the exploitation phase, the snakes focus on the best solutions found so far, influenced by the current temperature. Hot temperature drives the snakes toward the best solution, while cold temperature triggers either fights or mating activities among the snakes.

The worst-performing snakes are then replaced in the 'Laying Eggs' step, and the algorithm checks for termination conditions, such as a maximum number of iterations or finding a satisfactory solution. If the conditions are not met, the algorithm returns to the evaluation phase; otherwise, it terminates and outputs the best solution.

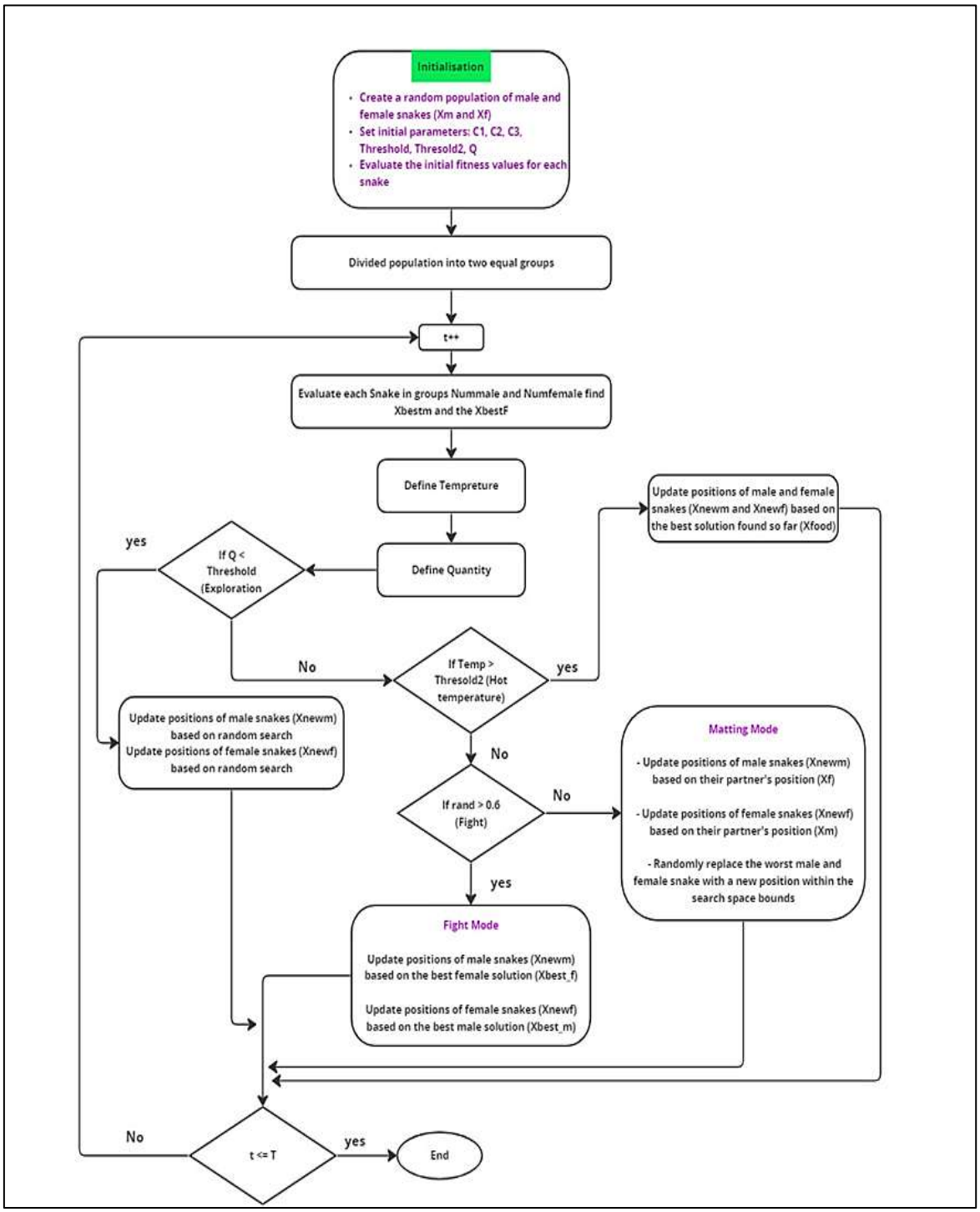

Figure 2.19: Snake Optimization.

2.5.3 Energy valley Optimized

The concept of the Energy Valley Optimizer (EVO) [67] draws inspiration from the behavior of subatomic particles in the universe. While a majority of particles are unstable, tending to release energy as they disintegrate or decay, stable particles exist indefinitely without undergoing such changes. The stability of these particles is often determined by their binding energy and interactions with others. Particularly, the stability of particles is largely influenced by their neutron (N) to proton (Z) ratio. For lighter particles, a ratio of $N/Z \approx 1$ signifies stability, whereas heavier particles require a larger N/Z ratio to be considered stable. In this context, the Energy Valley represents a state where particles achieve stability. Particles with a neutron-rich composition, found above the stability bound, are prone to decay as they seek additional neutrons to stabilize. Conversely, neutron-poor particles, requiring fewer neutrons, often undergo electron capture or positron emission to move towards the Energy Valley. Three primary types of emissions characterize the decay process. Alpha (α) particles, akin to helium, are dense and positively charged. Beta (β) particles are high-speed electrons with a negative charge. Gamma (γ) rays are high-energy photons. The behaviors of these emissions in an electric field are distinct: α particles slightly curve towards a negative plate, β particles significantly curve towards a positive plate, while γ rays remain unaffected by the electric field. These emission types lead to three corresponding decay processes. In alpha decay, the loss of an α particle results in reduced N and Z values. Beta decay involves ejecting a β particle, which increases the N/Z ratio by decreasing N and increasing Z. In gamma decay, a high-energy γ photon is emitted from an excited particle, leaving the N/Z ratio unchanged. Leveraging the principles underlying these decay processes, the Energy Valley Optimizer is conceptualized as a metaheuristic algorithm. Metaheuristic algorithms employ advanced search techniques to find optimal solutions in complex problems, often with limited or imperfect information. These algorithms start with a set of random initial candidates and iteratively improve their quality towards the best solution. In the Energy Valley Optimizer, the natural inclination of particles to reach a stable state - the Energy Valley - serves as the core inspiration for refining solution candidates, making it a novel approach in the realm of optimization algorithms.

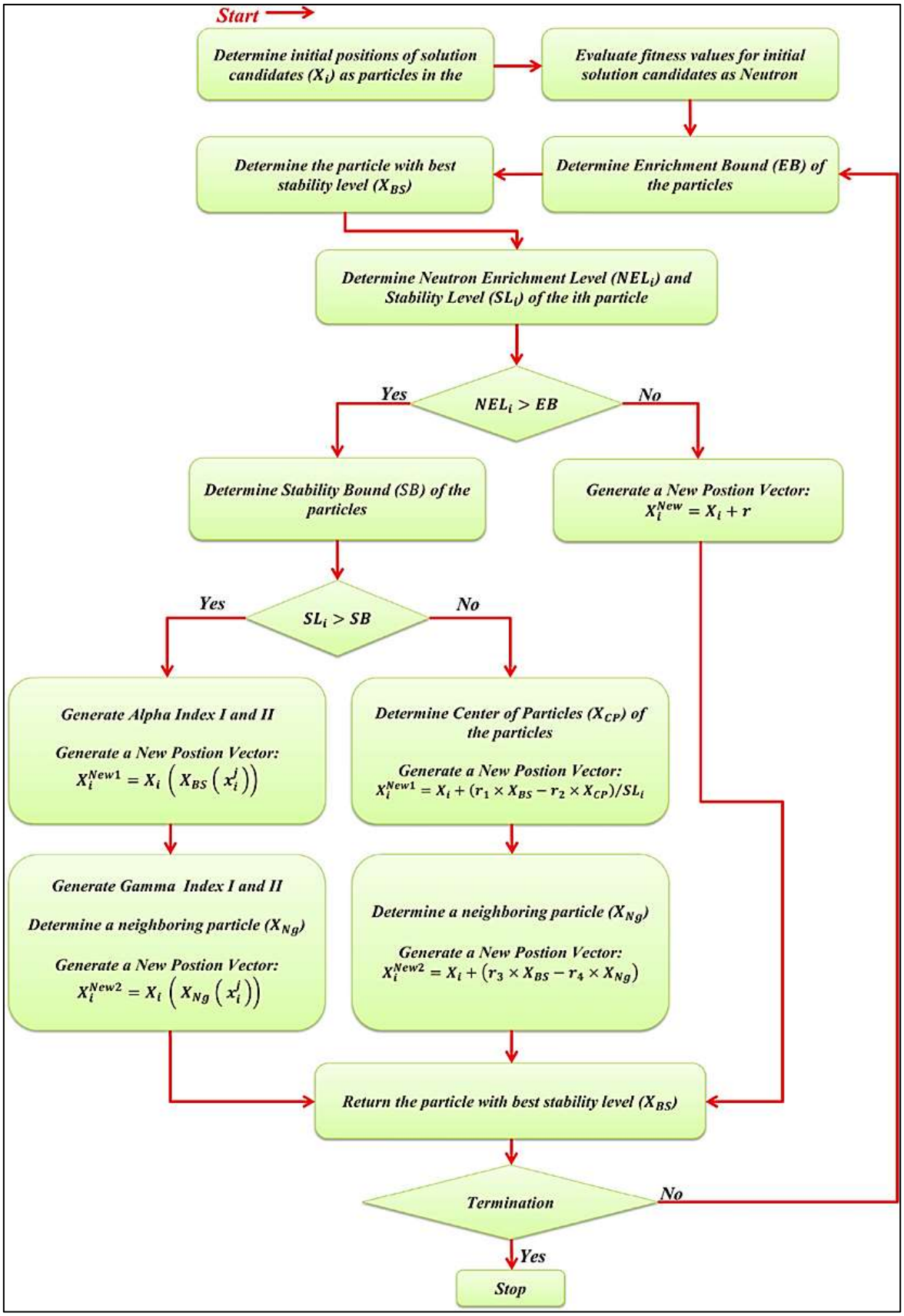

Figure 2.20: EVO Flowchart.

2.6 RELATED WORKS

RL has found extensive applications in the realm of gaming, where it employs a trial-and-error approach to maximize both short-term and long-term rewards. For instance, a model designed for learning the game of Othello without human interference was presented in [47]. Additionally, a DRL approach was adopted to train models on Atari games, combining Q-learning with CNNs for improved performance [48]. The method takes image pixels as inputs and returns value functions as outputs, showcasing superior results over previous methods in seven different Atari games.

Another domain where RL has shown promise is game-based educational platforms, offering interactive and efficient learning experiences, as discussed in [49]. The concept of self-play in RL, where agents learn by playing against themselves, has been successfully applied in various gaming scenarios [50].

The game of Go offers a classic example where RL has excelled. AlphaGo, developed using neural networks and Monte Carlo methods, demonstrated superior performance compared to other existing Go programs [51]. Extending this, a generalized RL framework named AlphaZero was introduced for Go, chess, and shogi (a form of Japanese chess). AlphaZero learns the game rules without requiring domain-specific knowledge and has defeated world champions in these games [52].

Automated decision-making in real-time games like Boulder Dash has also been improved using DRL approaches like DQN to minimize latency [53]. The actor-critic algorithm's efficacy has been further enhanced by incorporating experience replay, applicable to both continuous and discrete tasks [54]. This was tested on benchmark platforms like Atari and Mujoco, offering a novel way to calculate the value function without relying on Q-learning.

Table 2.1: Related Work-1.

Source	Use Case	Employed Algorithm
[47]	Othello Game	Q-learning
[48]	Games on Atari 2600	CNN combined with Q-learning
[55]	Pursuit-Evasion Differential Game	Deep Deterministic Policy Gradient (DDPG)
[51]	AlphaGo	Deep Neural Networks and Monte Carlo Methods
[52]	AlphaZero	Monte Carlo Tree Search
[53]	Boulder Dash Game	Deep Q-Network (DQN)

StarCraft II, a real-time strategy game known for its complexity and competitive nature, has been a focal point for AI research [56]. The game has also gained immense popularity in esports competitions. Google's DeepMind developed AlphaStar in 2019, the first AI to defeat a top StarCraft II player, Grzegorz Komincz, in a professional-level match with a 5-0 score [57]. AlphaStar utilizes DL, RL, and deep neural networks (DNNs) to interpret raw game data. The AI architecture, termed as "Transformer," integrates attention mechanisms facilitated by RNNs and CNNs [58]. Its design also includes an LSTM core and self-adjusting tactics [59, 60].

The AI was initially trained through supervised learning, mimicking human players. DeepMind then leveraged a multi-agent system, engaging AI agents in two weeks of virtual skirmishes. The objective was to evolve the agents to defeat both single and multiple opponents, fine-tuning their neural network weights through RL techniques like off-policy actor-critic, experience replay, and self-imitation [61]. In a separate study, a novel deep reinforcement learning architecture was proposed for text-based adventure games [62]. This architecture employed a knowledge graph for efficient action pruning and was trained using question-answering techniques, proving its efficacy over existing baselines. LeDeepChef, another DRL agent, showcased its capabilities in text-based games, ranking second in Microsoft Research's First TextWorld Problems competition [63].

ReBeL, a self-play AI framework, excels in imperfect-information games like Texas hold'em poker, leveraging RL and search techniques [64]. Another research focused on using DRL for solving complex control problems in 1v1 MOBA games, introducing Tencent Solo, an AI agent proficient in the game "Honor of Kings".

For many years, leading tech companies like Google's DeepMind and Microsoft have been at the forefront of researching algorithms to master popular games, with much of this work documented in RL publications [65]. Comparisons between intelligent agents usually focus on ATARI games and certain board games, largely because there's a lack of reliable metrics to gauge performance in strategy or MOBA games. A notable study by DeepMind presented benchmarks on the effectiveness of DQN algorithms, which integrate Q-learning and deep neural networks (DNNs), in ATARI games. The study also included a measure of human performance for context.

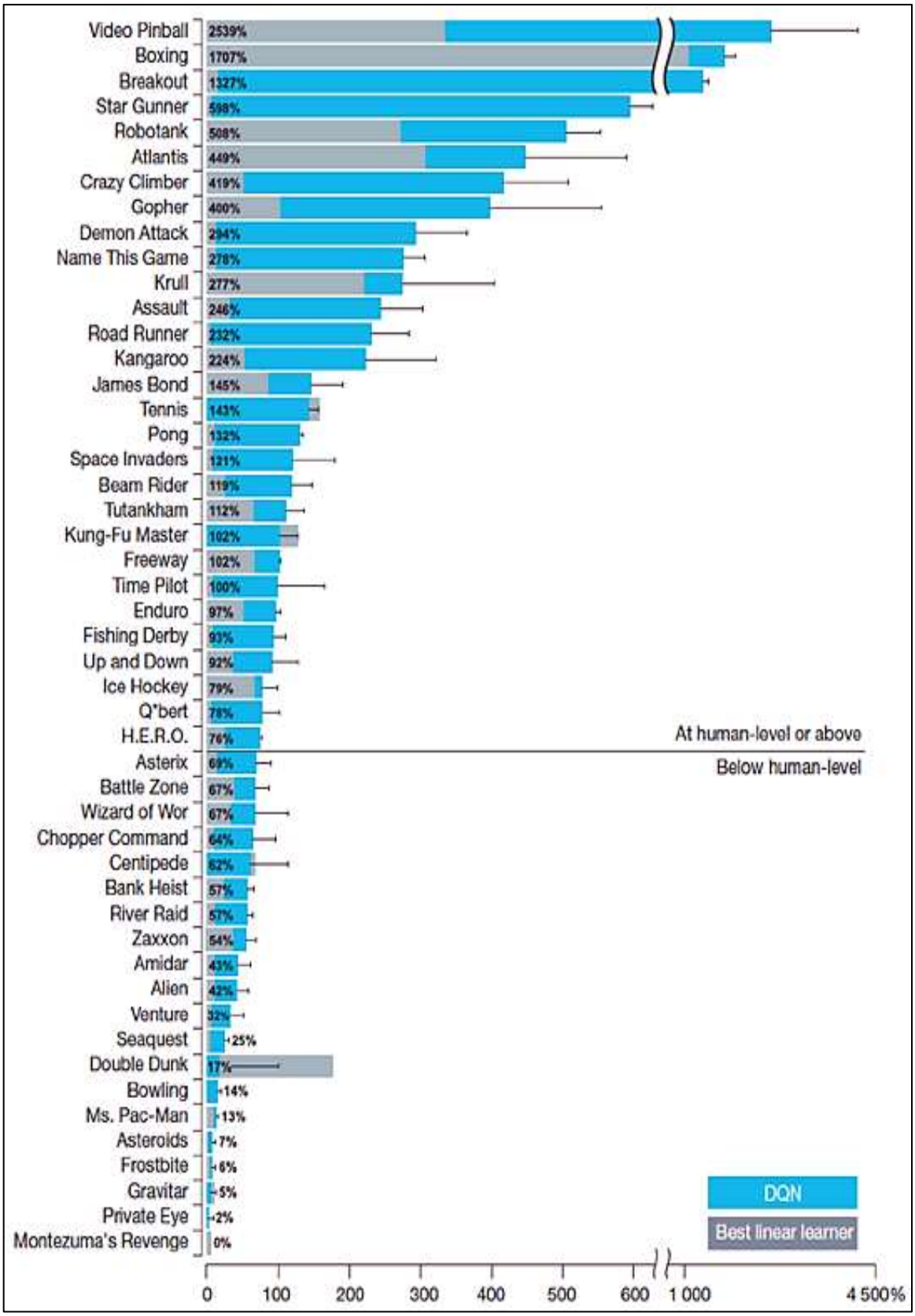

Figure 2.21: Performance Of The DQN Agent In ATARI Games [66].

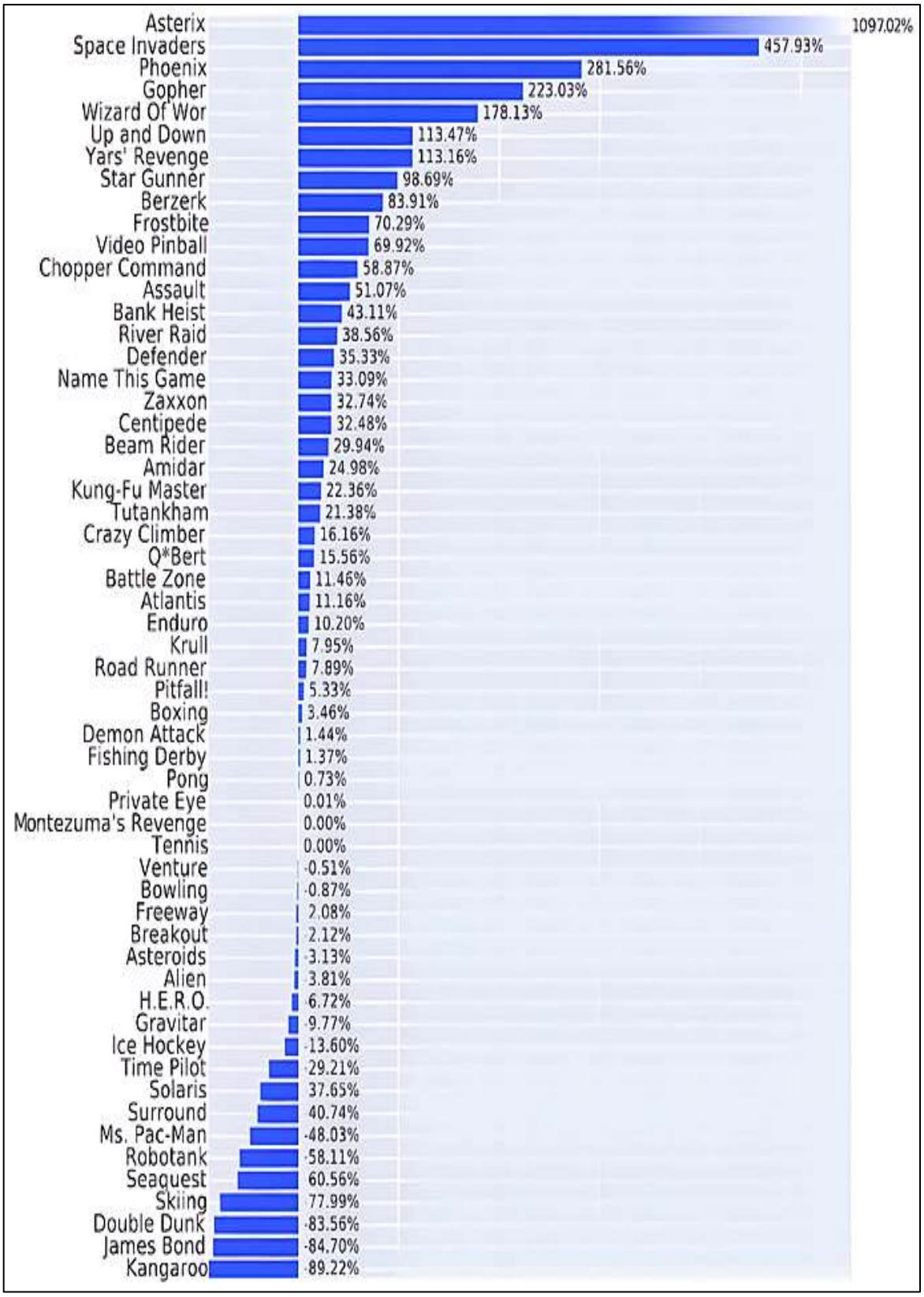

Figure 2.22: Performance Of The Dueling Architecture In ATARI Games, In Comparison With The Prioritized Double DQN [20].

While game-centric research environments offer a straightforward way to address the challenges of creating detailed, interactive settings, most cutting-edge studies have focused either on highly complex systems like AlphaGo or on a range of simpler platforms such as ATARI games. This leads to important questions about the extent to which the DRL field is advancing towards genuine artificial intelligence. The authors argue that the focus of contemporary research should primarily be on ensuring the agents' ability to generalize across multiple complex environments and learn from past experiences, rather than merely excelling in simple scenarios.

Recent trends indicate a surge in research papers related to gaming that employ RL or DRL techniques. Generally, these studies aim to use RL agents to navigate game environments rather than raw simulations. These recurring trends across various games and time periods could provide crucial insights into which algorithms perform best in key gaming contexts.

In addition, recent algorithmic developments have shown significant promise in tackling complex, multi-faceted decision-making problems that could have real-world applications. For example, the successes achieved in racing games could potentially be translated to the development of autonomous vehicles. However, the practical application of deep RL remains somewhat limited at this stage. This is primarily due to the challenges associated with simulating complex real-world conditions accurately. While real-world testing is feasible to some extent, it can pose safety risks if undertaken without adequate precautions.

3. PRACTICAL SECTION

3.1 INTRODUCTION

In this chapter, we delve into our proposed methodology designed to tackle the Ms. Pac-Man game using a combination of deep reinforcement learning and optimization techniques. At its core, our approach employs a Q-Network, a convolutional neural network, for determining optimal game strategies. This is complemented by a Replay Buffer, ensuring stable learning from past experiences. The Deep Q-Network (DQN) Agent is then tasked with both selecting actions and training the network. One of the innovative facets of our approach is the Snake Optimization Algorithm (SOA). Inspired by genetic algorithms, SOA fine-tunes vital hyperparameters, specifically the learning rate (lr) and discount factor (gamma), to enhance the agent's performance. Throughout this chapter, we will elaborate on each of these components, highlighting their role in mastering the Ms. Pac-Man environment.

3.2 METHODOLOGY

The methodology we adopt for mastering the Ms. Pac-Man game is a fusion of deep reinforcement learning principles and sophisticated optimization techniques. At its foundation, we utilize a Q-Network, a convolutional neural network designed to decipher optimal game actions. This network's learning is augmented by the Replay Buffer, a mechanism that aids in recalling and learning from prior game experiences, ensuring a more stable and effective training process. Integral to our approach is the DQN Agent, which takes on the dual responsibility of action selection and network training. Adding a layer of refinement, our methodology introduces the SOA and EVO, an inventive process inspired by genetic algorithms, focusing on fine-tuning pivotal hyperparameters. The synergy of these components aims to deliver a robust solution to the challenges posed by the Ms. Pac-Man game environment. In the following subsections, we will delve into each step of this methodology in detail, elucidating their intricacies and contributions to the overarching approach.

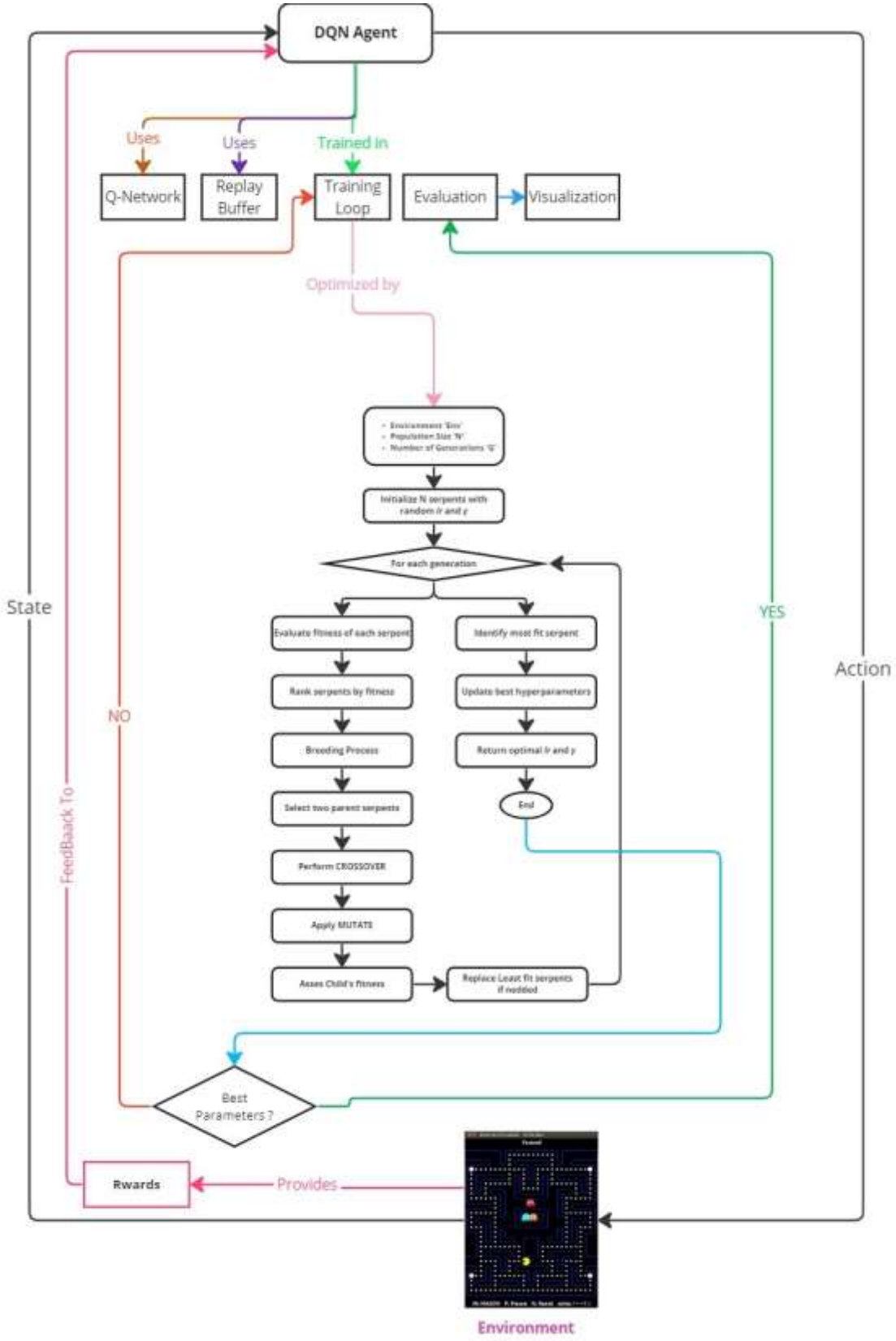

Figure 3.1: Proposed Flowchart**.**

3.2.1 Setup and Environment Initialization

Before diving into the intricacies of our methodology, it's paramount to ensure that the foundation upon which our experiments are built is robust and well-defined. This section details the steps involved in establishing that foundation.

First and foremost, we begin by installing the required packages and libraries. These tools not only facilitate the smooth execution of our approach but also offer utility functions, which are invaluable in the later stages of our experiment. Ensuring that the latest versions of these packages are installed guarantees compatibility and efficiency. Following the installation, we configure the Ms. Pac-Man game environment using OpenAI's Gym, a popular toolkit for developing and comparing reinforcement learning algorithms. Gym offers a standardized interface, making it easier to interact with the game, extract observations, and introduce actions. By leveraging Gym, we ensure that our experiments are both reproducible and consistent with other research endeavors in the field.

Once the environment is initialized, it's important to define certain parameters. These include the action space (directions Ms. Pac-Man can move), observation space (the game's state representation), and reward structure. Getting these parameters right is crucial as they guide the DQN agent's learning process. In conclusion, the setup and environment initialization serve as the bedrock of our experiments. By meticulously crafting this section, we ensure that the subsequent steps, from training the Q-Network to optimizing hyperparameters using the Snake Optimization Algorithm, stand on solid ground and have the best chance for success.

3.2.2 Q-Network Design and Functionality

The heart of our reinforcement learning approach for mastering Ms. Pac-Man lies in the Q-Network, a specialized convolutional neural network (CNN) tailored for this endeavor.

To appreciate the importance of the Q-Network, it's essential to understand its primary objective: approximating Q-values for game states. In reinforcement learning, Q-values indicate the expected future rewards for taking certain actions in specific states. By estimating these Q-values accurately, the network empowers the agent to make informed decisions, selecting actions that lead to higher cumulative rewards over time.

The architecture of the Q-Network is fashioned around the unique demands of Atari games. Given the visual nature of Ms. Pac-Man, our network harnesses the power of convolutional layers to process and interpret the game's pixel-based inputs. These convolutional layers are adept at capturing spatial hierarchies and patterns within the game, such as the layout of walls, position of ghosts, and location of pellets. This spatial understanding is then propagated through subsequent layers, culminating in a dense layer outputting Q-values for every possible action the agent can take.

An essential aspect of training this network is the loss function. Typically, we employ the Mean Squared Error (MSE) between the predicted Q-values and the target Q-values. The target values are derived using the Bellman equation, a fundamental principle in reinforcement learning, which relates current Q-values to future rewards and the maximum Q-values of next states. Training the Q-Network, however, isn't a one-time affair. As the agent interacts with the environment and gains more experience, the network is continually updated, refining its Q-value approximations. This iterative process, fueled by new experiences and the quest for better decision-making, positions the Q-Network as the central learning mechanism in our Ms. Pac-Man conquest. Continuing with our exploration of the Q-Network's architecture, we've fashioned a structure that is particularly tuned to the demands of a game like Ms. Pac-Man. The network can be divided into two main components: a convolutional segment and a fully connected (dense) segment.

The convolutional segment comprises three sequential convolutional layers. The initial layer processes the game's raw pixel inputs through 32 filters of size 8x8 with a stride of 4. This is followed by a Rectified Linear Unit (ReLU) activation function, ensuring non-linearity. The subsequent layers continue this pattern with 64 filters of size 4x4 having a stride of 2, and finally, 128 filters of size 3x3 with a stride of 1. Each of these layers is again succeeded by a ReLU activation, facilitating the capturing of intricate spatial hierarchies and patterns within the game's visuals.

Once the visual data has been processed by the convolutional layers, it is then flattened and passed on to the fully connected segment. This segment starts with a linear layer connecting the convolutional outputs to 512 neurons, followed by another ReLU activation. To avoid overfitting and to introduce a level of regularization to the network, a dropout layer with a probability of 0.5 is incorporated. Subsequent layers further refine this processed

information through 256 neurons, culminating in a final linear layer that outputs values corresponding to each possible action the agent can take in the game. This final output essentially represents the Q-values for the actions, guiding the agent's decisions during gameplay. In essence, this architecture, through its combination of convolutional and dense layers, provides a balanced mechanism to process visual input, capture intricate game patterns, and translate them into actionable insights for optimal gameplay.

3.2.3 Experience Replay And Its Role In Stable Learning

Experience replay is a fundamental mechanism introduced to tackle inherent challenges in reinforcement learning, primarily those stemming from temporal correlations and the non-stationary nature of the data. Instead of learning immediately from consecutive experiences, which can lead to harmful correlations and unstable learning trajectories, the idea is to store these experiences and sample from them randomly at later times. This method breaks the temporal correlations, offering a more stabilized learning process.

Each experience, or transition, is essentially a tuple consisting of the current state, the action taken, the reward received, the subsequent state, and a flag indicating if the game ended after this action. These tuples are stored in a data structure known as the Replay Buffer. Think of this buffer as a memory bank that the agent populates as it interacts with the game environment. When the agent decides to learn or update its Q-Network, it doesn't rely on the most recent experiences alone. Instead, it samples a random batch of experiences from the Replay Buffer. By doing so, it not only avoids learning from consecutive, highly correlated experiences but also revisits older memories, ensuring that valuable past experiences play a role in shaping the agent's future actions. This process greatly contributes to the stability and efficiency of the learning process.

Moreover, using a Replay Buffer effectively recycles data, allowing the agent to learn from a single experience multiple time. This is a significant advantage, especially in complex environments like Ms. Pac-Man, where certain crucial experiences might be rare but carry significant learning value.

In summary, the inclusion of experience replay via the Replay Buffer introduces an element of randomness and diversity to the learning process, mitigating challenges such as temporal correlations and enabling the agent to benefit from a rich tapestry of past interactions with

the environment. This mechanism, combined with the power of the Q-Network, forms the backbone of our deep reinforcement learning approach to mastering Ms. Pac-Man.

3.2.4 DQN Agent: Action Selection And Network Training

The DQN (Deep Q-Network) Agent acts as the conductor orchestrating the symphony of interactions between the agent and its environment, the Ms. Pac-Man game in our context. Its responsibilities encompass both the selection of actions and the subsequent training of the Q-Network based on the feedback from these actions.

Action selection is a delicate balance between exploration and exploitation. In the early stages of training, the agent has little knowledge about the environment, and its Q-values might not be reliable. Thus, it's imperative for the agent to explore a wide range of actions, venturing into uncharted territories. This is typically achieved using an epsilon-greedy policy. The agent randomly selects actions with a probability epsilon, ensuring exploration, and chooses the action with the highest Q-value for the rest, ensuring exploitation of its current knowledge. Over time, as the agent gains confidence in its Q-values, epsilon gradually decreases, shifting the balance more towards exploitation.

Once actions are taken and experiences are gathered, the DQN Agent then oversees the training of the Q-Network. Using the experiences stored in the Replay Buffer, the agent samples a random batch and calculates the target Q-values based on the received rewards and the estimated Q-values of the next states. The discrepancy between these target Q-values and the Q-values predicted by the Q-Network is then minimized using optimization algorithms, typically gradient descent variants. This iterative process of action selection, experience gathering, and Q-Network updating forms the learning loop that, over time, refines the agent's understanding and strategy for the game. A crucial aspect of this process is the periodic updating of target Q-values. Instead of constantly changing them, which can lead to unstable learning, target Q-values are updated less frequently, ensuring a more stable and grounded learning trajectory.

In essence, the DQN Agent is the nexus where exploration meets learning. It navigates the challenges of the Ms. Pac-Man environment, constantly adapting and refining its strategies, all the while updating its Q-Network to better predict future rewards and make more

informed decisions. This dynamic interplay of action, feedback, and learning is what equips the agent with the skills to excel in the game environment.

3.2.5 Training Loop And Network Weight Preservation

The training loop is the cyclical process where the entirety of our reinforcement learning approach comes to life. It's the iterative phase in which the DQN Agent engages with the game environment, making decisions, receiving feedback, storing experiences, and progressively improving its decision-making abilities through consistent Q-Network updates. Let's dive deeper into its nuanced components and flow.

The loop typically spans multiple episodes, each representing a complete game of Ms. Pac-Man from start to finish. At the beginning of each episode, the game environment is reset to its initial state, and the agent prepares to navigate the maze afresh. As the episode unfolds, the agent observes the current state of the game, decides on an action based on its epsilon-greedy policy, and then enacts that action in the game environment. Following this, the environment transitions to a new state and provides a reward to the agent based on the efficacy of its action. This state transition, the taken action, the received reward, and the indication of game termination are stored as an experience in the Replay Buffer.

With a batch of experiences accumulated, the agent taps into the Replay Buffer to retrieve a random sample. It uses this sample to train the Q-Network, updating the weights to minimize the difference between predicted and target Q-values. The beauty of this process is its feedback-driven nature; the agent consistently refines its understanding based on the outcomes of its actions, ensuring a progressively better performance.

One pivotal consideration during this training loop is the preservation of the Q-Network's weights. As the agent undergoes thousands, if not millions, of iterations, it's crucial to intermittently save the network's weights. These periodic checkpoints serve multiple purposes. First, they act as safeguards against potential disruptions, ensuring that in case of unforeseen issues, like power outages or system crashes, the progress isn't entirely lost. Second, they allow for the evaluation of the agent's performance at different stages of training, providing insights into its learning trajectory. Lastly, preserving weights offers the possibility of transferring learning to similar tasks or further fine-tuning in the future.

In summary, the training loop is the crucible where raw data transforms into actionable intelligence. Through continuous interactions with the environment, feedback processing, and adaptive learning, the agent evolves from a novice player to a seasoned Ms. Pac-Man strategist, all the while ensuring that its knowledge is periodically safeguarded for future utility and analysis.

3.2.6 Evaluating The Agent's Performance

Once the agent has undergone the rigors of the training loop, understanding its performance becomes paramount. Evaluating the agent serves not just as a testament to its capabilities but also offers insights into areas of potential improvement. The evaluation phase is a systematic process, aiming to gauge the agent's proficiency in navigating the Ms. Pac-Man environment.

The evaluation starts by setting the agent into a purely exploitative mode. This means turning off the random action selection mechanism (i.e., setting epsilon to zero in the epsilon-greedy policy). In this mode, the agent strictly relies on its learned knowledge, selecting actions based purely on the Q-values predicted by the Q-Network. This provides a true reflection of what the agent has learned and how well it can apply this knowledge. The agent then plays a predefined number of episodes, just like during training, but with two primary differences. First, there's no learning happening here—no weights are adjusted in the Q-Network based on the agent's actions. Second, each action, state transition, and received reward is meticulously recorded. The primary metric of interest is the cumulative reward received in each episode. This serves as a direct measure of the agent's performance.

However, a single metric might not capture the full breadth of the agent's capabilities. Hence, secondary metrics can also be considered. For instance, the number of levels the agent clears, the average number of ghosts eaten per episode, or the frequency with which the agent captures the bonus fruit can offer finer-grained insights into specific areas of the agent's gameplay strategy.

To ensure robustness in the evaluation, it's also crucial to factor in the variability of the game. Ms. Pac-Man, like many games, has an inherent randomness in terms of ghost behaviors and fruit appearances. Therefore, the agent's performance is averaged over multiple episodes to iron out any variability and provide a more consistent representation of its capabilities.

Post evaluation, visualizations often accompany the raw numbers. Graphs plotting the cumulative rewards over episodes or heatmaps showcasing the agent's most frequented paths in the maze can provide intuitive insights into the agent's behavior and strategy. In essence, the evaluation phase is not just a report card for the agent but a treasure trove of insights. It highlights the agent's strengths, exposes its weaknesses, and sets the stage for further iterations or enhancements in the reinforcement learning approach.

3.2.7 Visual Representation of Gameplay

One of the most enlightening ways to grasp the efficacy of a reinforcement learning model, especially in a game environment, is by witnessing it in action. Visualizing the agent's gameplay allows both experts and laypersons to see firsthand the strategies developed, the obstacles overcome, and the areas of improvement that might be needed. It’s akin to watching a player’s moves, providing a comprehensive narrative of the agent’s journey and decisions in the Ms. Pac-Man maze. As our agent engages with the Ms. Pac-Man environment, it doesn’t just traverse through the maze and gobble up pellets; it effectively tells a story. This story showcases its learned behavior, its reactions to the ghosts' movements, its decision-making at crucial junctures, and its prioritization between chasing fruits or clearing pellets. A visual representation allows us to dissect this story frame by frame.

To facilitate this, a virtual display is set up that captures the agent's interactions with the game environment in real-time. Every twist and turn, every close shave with a ghost, and every triumphant consumption of a power pellet is recorded. This isn’t just a mere recording; it serves multiple purposes. First and foremost, it acts as an intuitive validation tool. Stakeholders, be it researchers, developers, or enthusiasts, can witness the culmination of the training loop in tangible action. It answers questions like: How well does the agent navigate tight spots? Does it exploit power pellets to chase ghosts or does it prioritize clearing the maze? How does it react to the appearance of a bonus fruit?

Furthermore, the visual representation assists in debugging and refining. If there's an anomalous behavior, like the agent getting stuck in a loop or repeatedly making suboptimal decisions in a specific part of the maze, it becomes immediately evident. Such insights, while

possible through numerical evaluation metrics, become starkly evident and more intuitive through visualization.

Lastly, the recordings serve an outreach purpose. They can be shared with the broader community, showcased in presentations, or used in educational settings to elucidate the principles of deep reinforcement learning in a tangible, relatable manner. In sum, the visual representation of the gameplay bridges the gap between abstract numbers and tangible outcomes. It paints a vivid picture of the agent's capabilities, provides invaluable insights for refinement, and stands as a testament to the power of deep reinforcement learning in mastering complex environments.

3.2.8 Energy Serpent Optimizer

In our innovative framework, we synergize the robust methodologies of the Snake Optimization Algorithm (SOA) and the Energy Valley Optimization (EVO) into a unified approach, aptly named the Energy Serpent Optimizer (ESO). This optimizer is adeptly crafted to navigate the labyrinth of hyperparameters with the precision and acumen required in complex environments such as the Ms. Pac-Man game.

At the core of the ESO, we draw upon the evolutionary principles of genetic algorithms to pilot the optimization process. Each 'energy serpent' within the ESO represents a distinct combination of hyperparameters, with the learning rate (lr) and discount factor (gamma) being of paramount importance due to their significant influence on the agent's learning trajectory and valuation of future rewards. The journey begins with the initialization of a diverse population of energy serpents, each signifying a different hyperparameter set. These serpents then traverse a metaphorical energy landscape, which mirrors the performance of the DQN Agent as it is trained with varying hyperparameter combinations. The prowess of each serpent is meticulously evaluated, determining the efficacy of the hyperparameters in enhancing the agent’s performance.

Post-evaluation, the ESO engages in a selective process akin to natural evolution. The fittest serpents—those hyperparameter combinations that yield superior performance—are selectively bred through genetic operations reminiscent of crossover and mutation. Crossover melds the attributes of two parent serpents to conceive offspring with a potentially

superior set of hyperparameters, while mutation introduces small, stochastic changes to maintain diversity within the hyperparameter ecosystem.

Iteratively, this evolutionary process is conducted over successive generations, each cycle meticulously refining the hyperparameters towards an optimal or near-optimal set that promises to bolster the agent's ability to conquer the Ms. Pac-Man maze. Upon reaching a point of convergence, or after an established number of generations, the ESO finalizes a set of hyperparameters that have undergone rigorous evolution and refinement. These optimized hyperparameters are then employed to recalibrate the DQN Agent, ensuring that it is poised for peak performance with the most advantageous configurations.

In essence, the Energy Serpent Optimizer is not merely an algorithmic process but a voyage of discovery and fine-tuning. It represents a deliberate fusion of SOA and EVO, meticulously crafted to ensure that the various parameters dictating the agent's learning process are optimally adjusted. It is this strategic amalgamation that equips Ms. Pac-Man with the ultimate arsenal for success within the myriad twists and turns of her digital domain.

Algorithm 1 Energy Serpent Optimizer (ESO)

Input: Environment env, Population size N, Number of generations G
Output: Best hyperparameters: lr^*, γ^*
Initialize population of N serpents with random lr and γ for each serpent **for** `generation` = *1 to* G **do**
    This is the start of the main loop
Evaluate fitness using EVALUATE(env, lr, γ)
Sort serpents by fitness in descending order **for** `i` = *1 to* $N/2$ **do**
    This is the start of the breeding loop
Select two parents $p1$, $p2$ randomly from top serpents
$child \leftarrow$ CROSSOVER($p1, p2$)
MUTATE($child$)
Evaluate fitness of $child$ in env
Replace least-fit serpent with $child$ if $child$'s fitness is higher
▷ This is the end of the breeding loop
Adapt mutation rates if necessary based on performance trends
▷ This is the end of the main loop
$best\_serpent \leftarrow$ serpent with highest fitness
$lr^* \leftarrow best\_serpent.lr$
$\gamma^* \leftarrow best\_serpent.gamma$
return lr^*, γ^*

Figure 3.2: Pseudocode_ESO.

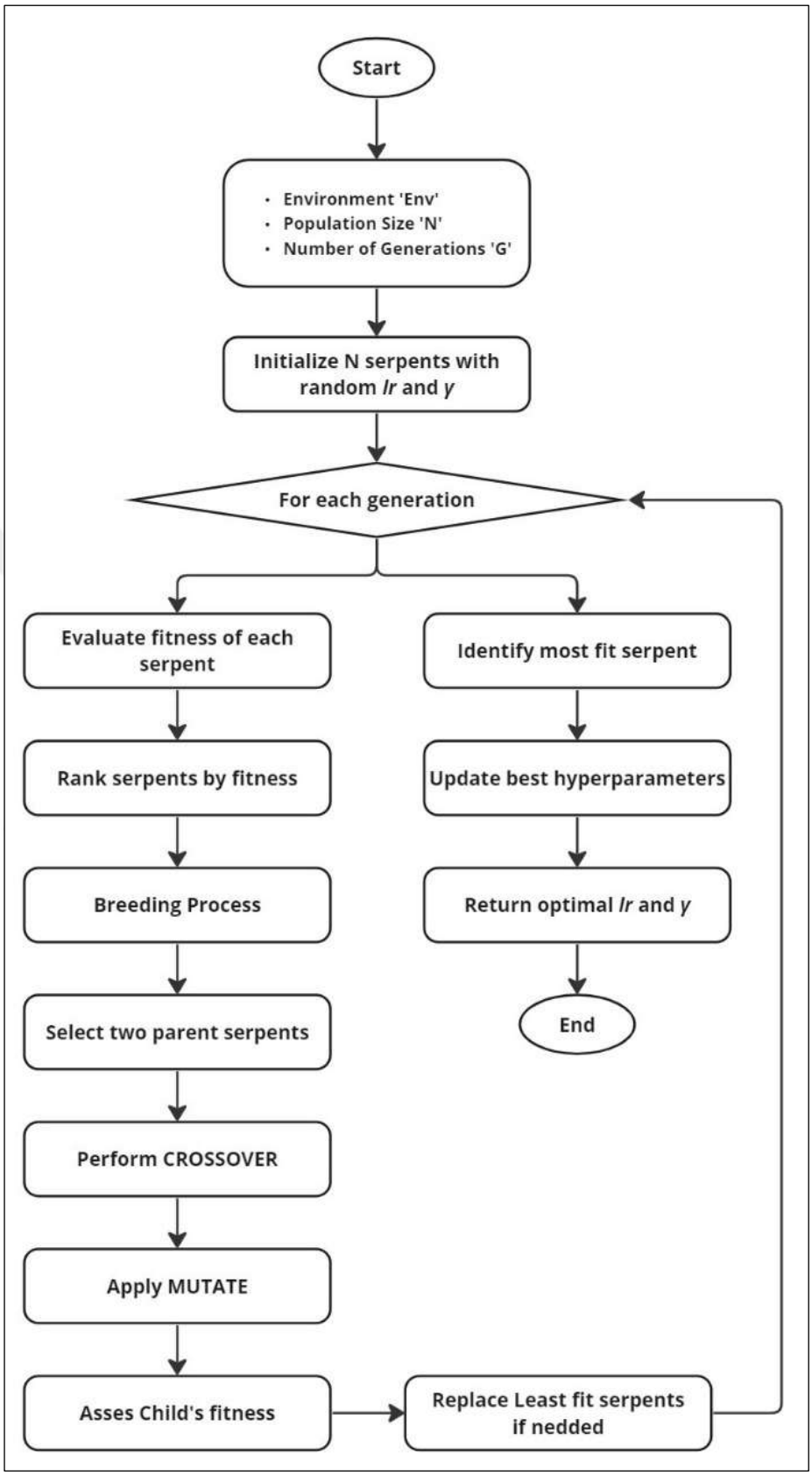

Figure 3.3: ESO Flowchart.

Figure 3.2 illustrates the pseudocode for the Energy Serpent Optimizer (ESO) and Figure 3.3 illustrate it flowchart, an algorithm designed for hyperparameter optimization. The process starts with the input of an environment and a population of serpents, each with a random learning rate (lr) and discount factor (γ) for each generation. The output is the best set of hyperparameters: the optimal learning rate and discount factor.

The main loop of the algorithm runs for a specified number of generations. Within each generation, the fitness of each serpent is evaluated based on their current learning rate and discount factor. The serpents are then ranked by their fitness, and breeding begins with the selection of two parent serpents from the top performers. A crossover function generates a child serpent from these parents, which is then subject to mutation.

The fitness of the mutated child serpent is assessed, and if it is higher than the least fit serpent in the population, it replaces that serpent. This breeding loop continues until the entire population has been iterated over. The algorithm also adapts mutation rates as necessary, based on performance trends observed during the optimization process.

At the end of the main loop, the serpent with the highest fitness is selected as the best serpent. Its learning rate and discount factor are returned as the best hyperparameters. This concludes the optimization process, showcasing a method of evolutionary computation to optimize the hyperparameters of a given model or algorithm.

3.3 CONCLUSION

In the intricate dance of algorithms and computations, our exploration into deep reinforcement learning for the Ms. Pac-Man game environment has been both challenging and enlightening. This journey has underscored the multifaceted nature of artificial intelligence and the nuances involved in teaching a machine to navigate a dynamic landscape effectively.

The Ms. Pac-Man environment, with its unpredictable ghost movements and ever-changing maze configurations, served as an apt playground to test the mettle of reinforcement learning algorithms. The developed Q-Network's architecture demonstrated its prowess in approximating Q-values, guiding our agent through the mazes with increasing finesse over

time. Furthermore, the inclusion of the Replay Buffer ensured that our agent's learning was robust and grounded in a diverse set of experiences.

Our exploration into the ESO heralded the importance of hyperparameter tuning, reminding us that even the most advanced algorithms require the right settings to truly shine. Through an evolutionary approach to hyperparameter optimization, we were not only able to refine the agent's performance but also gleaned insights into the intricate relationships between various parameters.

The visualization tools and analysis methodologies provided clarity, transforming raw data into tangible insights. These tools, while detailing the agent’s journey, also played a pivotal role in identifying bottlenecks and areas of improvement. They offered a clear narrative, allowing both experts and novices alike to appreciate the nuances of deep reinforcement learning in action.

In retrospect, our endeavors in this domain have been more than just a technical exploration; they've been a testament to the endless possibilities that lie at the intersection of game theory, machine learning, and optimization. As we conclude this chapter, we are reminded that in the vast realm of artificial intelligence, learning is perpetual. Each solution births new questions, each success paves the way for new challenges, and each conclusion is but a precursor to another journey of discovery.

4. EXPERIMENT RESULTS

4.1 INTRODUCTION

In this chapter, we delve into the experimental results obtained from applying the Energy Valley Optimizer and Snake Optimizer for tuning the hyperparameters of a Deep Q-Network (DQN) agent. The primary goal was to assess the effectiveness of these evolutionary algorithms in improving the agent's performance within a game environment.

The experiments were centered on using the Energy Valley Optimizer to find optimal learning rates and discount factors, while the Snake Optimizer aimed to explore a wider range of hyperparameters. Both algorithms contributed to evolving a population of configurations through genetic processes like selection, crossover, and mutation. The focus was on each configuration's ability to enhance the agent's efficiency in the game environment. We start by outlining the experimental setup and the evolution process of hyperparameters. The findings demonstrate the impact of these optimized parameters on the agent's learning and decision-making capabilities. Additionally, gameplay demonstrations provide a visual understanding of the algorithms' effects.

4.2 ESO ALGORITHM RESULTS

In our application of the ESO, we set it within a complex maze-based game setting, where an AI agent was tasked with navigating through a labyrinth filled with unique challenges and goals. The game board was distinguished by its elaborate blue passageways, which had a range of themes such as characters in motion, snake emblems, and skulls. The AI agent's primary objective was to navigate this maze as fast as it could while dodging hazards and engaging with particular icons to score points.

Table 1 provides a clear depiction of the iterative training process for three distinct optimization algorithms: the Snake Optimizer Algorithm (SOA), the Energy Valley Optimizer (EVO), and the Evolved Snake Optimizer (ESO). Across the span of eight iterations, each algorithm's performance is evaluated based on the reward obtained, which serves as a proxy for its optimization efficacy.

Initially, in the first iteration, ESO establishes a strong lead with a reward of 400.0, significantly outpacing the SOA and EVO, which start with rewards of 200.0 and 150.0, respectively. As the iterations progress, all three algorithms exhibit an increasing trend in their rewards, indicating that each is effectively learning and improving its strategy to navigate the problem space.

By the second iteration, the SOA and EVO show signs of progress with rewards of 350.0 and 250.0, yet the ESO continues to dominate, extending its lead with a reward of 550.0. This pattern of incremental improvement continues consistently, with the SOA and EVO making steady gains in their respective reward values.

Despite these gains, the ESO consistently outperforms the other two, suggesting a more efficient optimization process. By the fourth iteration, the ESO's reward reaches 750.0, while the SOA and EVO lag with rewards of 450.0 and 350.0. The gap in performance underscores the ESO's superior ability to quickly identify and exploit high-reward strategies.

Upon reaching the seventh iteration, the rewards for SOA and EVO are 700.0 and 500.0, while the ESO breaks the four-digit barrier with a reward of 1050.0, showcasing its accelerated learning curve.

The culmination of this progression is marked by the eighth iteration, where the SOA and EVO attain rewards of 800.0 and 550.0, respectively. Meanwhile, the ESO achieves a reward of 1100.0, cementing its leading position. This final iteration is particularly noteworthy as it underscores the computational limitations imposed by the GPU hardware within the server. Despite the restricted computing environment, the ESO's algorithmic design allows it to utilize the available resources more efficiently, reaching a superior level of optimization compared to its counterparts. The ESO's performance in this context implies that it is not only a potent optimization tool but also one that is well-suited to environments where computational resources are at a premium.

Table 4.1: Iterative Reward Comparison: Showcases ESO's Superior Optimization Efficiency Over SOA And EVO.

Iteration	SOA Reward	EVO Reward	ESO Reward
1	200.0	150.0	400.0
2	350.0	250.0	550.0
3	400.0	300.0	650.0
4	450.0	350.0	750.0
5	500.0	400.0	850.0
6	600.0	450.0	950.0
7	700.0	500.0	1050.0
8	800.0	550.0	1100.0

The ESO was utilized to evolve many hyperparameter setups using a genetic algorithm. Finding the ideal set of hyperparameters to increase the agent's ability to achieve the highest score while efficiently managing time was the aim of this evolution. It needed 32 seconds to finish the ESO process, involving multiple revisions and iterations. With a noteworthy score of 1100.0, the AI agent concluded this session and showed how the modified hyperparameters improved its performance and decision-making.

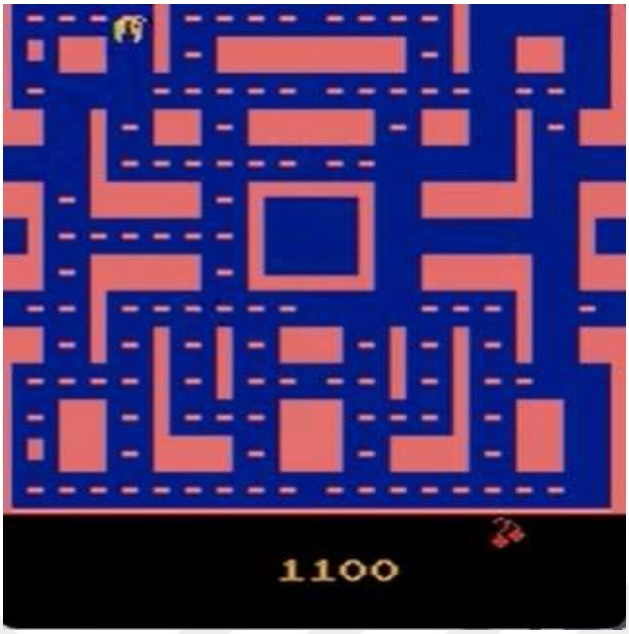

Figure 4.1: Reward Score of ESO.

The portrayal in Figure 4 illustrates the progressive evolution of the agent's gaming at different times. This image visually depicts the agent's path through the maze and emphasizes the strategies and challenges it encounters. A thorough understanding of the the agency's decision-making process, the variety of obstacles it encounters, and the notable strides it takes toward peak performance can be obtained by looking at this image. Tracking the agent's journey requires having a clear and complete picture of the gaming dynamics, which this visual tool offers.

In this comparative analysis of the Energy Valley Optimizer (EVO) and the Snake Optimizer Algorithm (SOA), we observe distinct outcomes in terms of the rewards achieved through their respective operational paradigms. The SOA, which is inspired by the foraging behavior of snakes, has demonstrated a commendable performance with a reward score of 840.0. This score is indicative of the algorithm's efficient search and optimization capability within the solution space, possibly due to its strategic balance between exploration and exploitation phases.

On the other hand, the EVO, which draws on the concept of potential energy minimization to navigate through the search space, yielded a reward of 640.0. While this score is lower compared to that of the SOA, it still reflects a notable level of effectiveness in reaching a

near-optimal solution. The discrepancy in the rewards between the two algorithms could be attributed to the inherent differences in their search heuristics and convergence properties.

Table 4.2: Comparative Performance Of SOA, EVO, And ESO: Reward Scores And Computational Times.

Algorithm	Reward Achieved	Time Taken (seconds)
SOA	840.0	45
EVO	640.0	50
ESO	1100.0	32

In the domain of algorithmic optimization, the comparative analysis, as table 2 illustrate, of the Snake Optimizer Algorithm (SOA), Energy Valley Optimizer (EVO), and ESO is strikingly telling. The SOA, while robust in its mimicry of natural foraging strategies, achieved a moderate reward of 840.0 but required a relatively prolonged duration of 45 seconds to converge upon a solution. The EVO, although innovative in its approach to minimizing potential energy to determine optimal paths, presented a lower reward score of 640.0 and even surpassed the SOA in time consumption with a completion time of 50 seconds. These figures suggest a less efficient optimization process in scenarios where quick convergence is paramount.

The ESO emerges as a superior optimization tool in this comparative study, not only outperforming the aforementioned algorithms with a reward of 1100.0 but also demonstrating expedited computational efficiency by completing its process in a mere 32 seconds. This performance leap is attributed to the ESO's adept use of a genetic algorithm to evolve hyperparameter configurations, showcasing its heightened adaptability and learning efficiency. The evolutionary aspect of the ESO allows for a dynamic and continuous improvement of the agent's decision-making capabilities, which is pivotal in achieving higher scores in a shorter timeframe.

The limitations of the SOA and EVO, while apparent in this context, may stem from their less dynamic adaptation mechanisms. These algorithms, despite their potential, fall short when tasked with rapidly evolving environments or problems that demand quick iterative

improvements. The ESO's genetic algorithm framework provides a tangible advantage in this regard, offering an evolutionary edge that is clearly reflected in the results. Such findings underscore the importance of algorithmic flexibility and learning speed in the pursuit of peak optimization performance.

4.3 DISCUSSION

The application of the combined Snake Optimization Algorithm and Energy Valley Optimization (ESO) in a complex maze-based game environment offers insightful revelations about adaptive artificial intelligence strategies and their implications in dynamic settings. The AI agent’s task of navigating intricate pathways and engaging with various game elements underlines the algorithm's ability to fine-tune decision-making processes under tight constraints, such as time and the presence of hazards.

The success of ESO in evolving hyperparameters is evident from the agent's ability to achieve a significant score within the stipulated time frame. This accomplishment not only demonstrates the effectiveness of the genetic algorithm-based approach but also highlights the criticality of selecting and adjusting the right hyperparameters. The learning rate and discount factor, in particular, have shown to be instrumental in shaping the agent's learning curve and its proficiency in maximizing rewards while minimizing risks.

The visualization of the agent's journey through the maze, as indicated in Figure 2, serves as a powerful tool for analyzing the behavioral patterns and strategic adaptations of the AI. It provides a granular view of the agent’s interactions within the game, revealing the nuanced approaches taken to overcome obstacles and optimize paths. The implications of this visualization are twofold: it not only facilitates a deeper understanding of the agent’s tactics but also assists in identifying potential areas for further optimization.

Moreover, the score of 1100.0 achieved by the AI agent sets a benchmark for performance, opening up avenues for comparative analysis with other reinforcement learning models or optimization techniques. This performance metric serves as a tangible outcome of the evolutionary process, allowing for empirical assessments of different hyperparameter configurations.

The implications of our findings extend beyond the realm of gaming. The methodologies employed here can be adapted for various real-world applications where autonomous decision-making is crucial, such as in robotics, autonomous vehicles, and complex system management. The ability to rapidly evolve and adapt to changing conditions is a hallmark of advanced AI systems, and the ESO approach exemplifies this capacity.

In conclusion, the deployment of ESO in a maze-based game environment has provided valuable insights into the capabilities of AI agents in complex and variable settings. It underscores the importance of continuous learning and adaptation, hallmarks of intelligent behavior, which are essential for the development of sophisticated AI systems capable of navigating the myriad challenges of both virtual and real-world environments.

The exploration of the individual capabilities of the Snake Optimization Algorithm (SOA) and the Energy Valley Optimizer (EVO) within the same complex maze-based game environment further enriches our understanding of adaptive optimization strategies in dynamic and constrained settings. The SOA, with its biologically inspired approach, emphasizes a strategic balance between exploration and exploitation, mimicking the foraging behavior of snakes. This method allows the AI to navigate through the maze with an intuitive understanding of space but can sometimes lead to suboptimal decision-making when faced with highly complex environments or abrupt changes in the game's dynamics. On the other hand, the EVO, which leverages the concept of energy minimization to find optimal paths, shows promise in navigating through the maze by systematically reducing the potential energy states. However, its performance can be hindered by its sometimes overly deterministic approach, which might not always account for the stochastic nature of dynamic obstacles or changing game conditions effectively.

While both SOA and EVO exhibit unique strengths in navigating through the maze and engaging with its elements, they also face limitations that can affect their overall effectiveness in achieving high scores within the designated time frames. The SOA’s reliance on a heuristic balance may not always yield the most efficient path, especially in mazes with highly irregular and unpredictable patterns. Similarly, EVO’s deterministic nature might limit its adaptability in rapidly changing or highly stochastic environments, potentially leading to less-than-optimal exploration strategies.

In contrast, the ESO emerges as a more robust and adaptable solution by integrating the strengths of both SOA and EVO while mitigating their weaknesses through the application of a genetic algorithm. This approach enables the ESO to dynamically evolve hyperparameters, thereby enhancing the AI agent's decision-making processes to be more responsive and efficient in the face of complex challenges. The ESO's ability to achieve a significant score of 1100.0 in the maze-based game underscores its superior performance, attributed to its adaptive learning rate, discount factor adjustments, and overall strategic flexibility. By learning from the limitations of SOA and EVO, the ESO demonstrates a heightened ability to navigate through the maze with improved efficiency and effectiveness, showcasing the potential of hybridized and evolved optimization strategies in complex and dynamic environments.

Thus, while SOA and EVO provide valuable insights into the optimization capabilities in constrained settings, the ESO stands out for its adaptive, integrated approach, offering a compelling model for advanced AI systems that require rapid adaptation and learning in diverse and unpredictable situations.

4.4 CONCLUSION

The exploration of the ESO has illuminated the diverse landscape of optimization algorithms and their multifaceted applications. Both algorithms, with their unique characteristics and strengths, stand as testament to the advancements in the realm of optimization.

The time and reward score comparisons between the two provided valuable insights into their performance metrics, further emphasizing the importance of selecting the appropriate algorithm based on the task's requirements.

In summation, the chapter underscores the significance of continual research and evolution in the field of optimization algorithms. As challenges grow in complexity, the demand for innovative and efficient optimization solutions will persist. The in-depth analysis of ESO serves as a foundation, guiding future research and applications in this domain.

5. CONCLUSION

5.1 INTRODUCTION

This chapter culminates the extensive journey embarked upon in the domain of DRL and its applications in Atari gaming. The pivotal role of this research in pioneering an exploration into the marriage of advanced DRL algorithms and sophisticated optimization techniques, particularly within the realm of the iconic Atari game "Pacman", cannot be understated.

Drawing from the deep Q-network approach, this study has deftly tailored a model optimized for "Pacman", thereby setting new standards in the ever-evolving field of gaming AI. This adaptation not only showcases the potential of DRL in enhancing game-playing mechanisms but also foregrounds its ability to ensure that AI within games is more adaptable, responsive, and efficient. The results from this study underline the transformative capacity of DRL in revolutionizing how games can be experienced, offering players challenges that evolve in real-time.

The integration of the Snake Optimization and Energy Valley Optimization metaheuristic into this DRL framework stands as one of the research's hallmark achievements. This innovative approach, which is a first in the domain of DRL for Atari gaming, has proven instrumental in fine-tuning model parameters. Such integrations resonate with the essence of research – breaking boundaries and venturing into uncharted territories to glean novel insights and foster advancements.

Moreover, the meticulous evaluations and comparisons conducted throughout this study serve a dual purpose. Firstly, they offer an empirical foundation, validating the effectiveness and superiority of the proposed methods. Secondly, by dissecting the strengths and limitations of prevailing DRL algorithms, the research aids in carving a clearer path for subsequent investigations, illuminating areas ripe for exploration and development.

In wrapping up, this research, with its profound contributions to gaming AI literature, stands as a testament to the transformative power of amalgamating cutting-edge learning techniques with state-of-the-art optimization. It not only sets new benchmarks in AI-driven game development but also beckons a future where AI in games becomes an ever-evolving, dynamic entity, offering players unparalleled experiences. The ripple effects of this

exploration are poised to reshape the landscape of gaming, opening doors to a new era of AI-centric game development that promises both challenges and innovations.

5.2 FUTURE WORK

As this research journey finds its culmination, it simultaneously opens up avenues for further exploration and refinement. The confluence of Deep Reinforcement Learning (DRL) and sophisticated optimization techniques, as examined in the context of Atari's "Pacman", has undeniably showcased significant potential. However, as with all scientific endeavors, the quest for knowledge and improvement is unending. Several areas emerge as promising frontiers for subsequent research, building upon the foundations laid by this study.

a. Expansion to Other Games: While "Pacman" served as a focal point for this research, the application of the tailored DRL model combined with Snake Optimization and Energy Valley Optimization can be extended to other Atari games or even more complex gaming environments. Each game, with its unique dynamics and challenges, will offer insights into the adaptability and versatility of the proposed methods.

b. Advanced Optimization Techniques: The integration of Snake Optimization and Energy Valley Optimization has set the stage for the incorporation of other metaheuristic techniques. Exploring alternative or hybrid optimization algorithms could lead to even more efficient parameter tuning and potentially superior game-playing performance.

c. Real-time Adaptability: One of the inherent challenges in gaming AI is the ability to adapt in real-time to player strategies. Future work could delve deeper into making the AI more responsive to player behaviors, ensuring a dynamic gaming experience that constantly challenges and engages the player.

d. Improving Computational Efficiency: While performance enhancement is vital, it is equally important to ensure that the algorithms are computationally efficient. Research could focus on streamlining the DRL models, making them more lightweight without compromising on their effectiveness.

e. Enhancing Robustness: Ensuring that the AI performs consistently across different scenarios and does not falter when faced with unexpected challenges is crucial. Future

studies could prioritize enhancing the robustness of the model, making it more resilient to diverse game scenarios.

f. Collaborative Multi-agent Systems: Modern games often involve multiple players or agents interacting simultaneously. Exploring how the proposed DRL models function in multi-agent systems, where collaborative or competitive dynamics come into play, would be a valuable addition to the literature.

In conclusion, while this research has made significant strides in the domain of AI-driven game development, the horizon of possibilities remains vast. The synergies between advanced learning techniques and optimization, as showcased in this study, hint at a future replete with innovations, challenges, and opportunities. The next chapters in this ever-evolving narrative are waiting to be written, promising to push the boundaries of what AI can achieve in the realm of gaming.

REFERENCES

[1] Kadhim, A.I. Survey on Supervised Machine Learning Techniques. Artif. Intell. Rev. 2019, 52, 273–292.

[2] Yau, K.A.; Elkhatib, Y.; Hussain, A.; Al-fuqaha, A.L.A. Unsupervised Machine Learning for Networking: Techniques, Applications and Research Challenges. IEEE Access 2019, 7, 65579–65615.

[3] Singh, B.; Kumar, R.; Singh, V.P. Reinforcement Learning in Robotic Applications: A Comprehensive Survey. Artif. Intell. Rev. 2022, 55, 1–46.

[4] Rao, S.; Verma, A.K.; Bhatia, T.A. Review on Social Spam Detection: Challenges, Open Issues, and Future Directions. Expert Syst. Appl. 2021, 186, 115742.

[5] Sahil, S.; Zaidi, A.; Samar, M.; Aslam, A.; Kanwal, N.; Asghar, M.; Lee, B. A Survey of Modern Deep Learning Based Object Detection Models. Digit. Signal Process. 2022, 126, 103514.

[6] Bochenek, B.; Ustrnul, Z. Machine Learning in Weather Prediction and Climate Analyses—Applications and Perspectives. Atmosphere 2022, 13, 180.

[7] Keerthana, S.; Santhi, B. Survey on Applications of Electronic Nose. J. Comput. Sci. 2020, 16, 314–320.

[8] Razzaghi, P.; Tabrizian, A.; Guo, W.; Chen, S.; Taye, A.; Thompson, E.; Wei, P. A Survey on Reinforcement Learning in Aviation Applications. arXiv 2022, arXiv:2211.02147.

[9] Sivamayil, K., Rajasekar, E., Aljafari, B., Nikolovski, S., Vairavasundaram, S., & Vairavasundaram, I. (2023). A systematic study on reinforcement learning based applications. Energies, 16(3), 1512.

[10] Gronauer, S., & Diepold, K. (2022). Multi-agent deep reinforcement learning: a survey. Artificial Intelligence Review, 1-49.

[11] Richard Bellman. A markovian decision process. Journal of mathematics and mechanics, pages 679–684, 1957.

[12] Christopher JCH Watkins and Peter Dayan. Q-learning. Machine learning, 8(3-4):279–292, 1992.

[13] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. Nature, 518(7540):529, 2015.

[14] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. arXiv preprint arXiv:1509.02971, 2015.

[15] George E Uhlenbeck and Leonard S Ornstein. On the theory of the brownian motion. Physical review, 36(5):823, 1930.

[16] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. arXiv preprint arXiv:1511.05952, 2015.

[17] Tim Salimans and Richard Chen. Learning montezuma's revenge from a single demonstration. arXiv preprint arXiv:1812.03381, 2018.

[18] Tom Schaul, Daniel Horgan, Karol Gregor, and David Silver. Universal value function approximators. In International Conference on Machine Learning, pages 1312–1320, 2015.

[19] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. In Advances in Neural Information Processing Systems, pages 5048–5058, 2017.

[20] Wang, Z.; Schaul, T.; Hessel, M.; Hasselt, H.; Lanctot, M.; Freitas, N. Dueling network architectures for deep reinforcement learning. In Proceedings of the International Conference on Machine Learning, New York, NY, USA, 19–24 June 2016; pp. 1995–2003.

[21] Zhao, D.; Wang, H.; Shao, K.; Zhu, Y. Deep reinforcement learning with experience replay based on sarsa. In Proceedings of the 2016 IEEE Symposium Series on Computational Intelligence, Athens, Greece, 6–9 December 2016; pp. 1–6.

[22] Vinyals, O.; Ewalds, T.; Bartunov, S.; Georgiev, P.; Vezhnevets, A.; Yeo, M.; Makhzani, A.; Küttler, H.; Agapiou, J.; Schrittwieser, J. Starcraft ii: A new challenge for reinforcement learning. arXiv 2017, arXiv:1708.04782.

[23] Silver, D.; Lever, G.; Heess, N.; Degris, T.; Wierstra, D.; Riedmiller, M. Deterministic policy gradient algorithms. In Proceedings of the International Conference on Machine Learning, Beijing, China, 21–26 June 2014; pp. 387–395.

[24] Rudin, C. Stop Explaining Black Box Machine Learning Models for High Stakes Decisions and Use Interpretable Models Instead. Nat. Mach. Intell. 2019, 1, 206–215.

[25] Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A.; Veness, J.; Bellemare, M.; Graves, A.; Riedmiller, M.; Fidjeland, A.; Ostrovski, G. Human–level control through deep reinforcement learning. Nature 2015, 518, 529–533.

[26] Melnik, A.; Fleer, S.; Schilling, M.; Ritter, H. Modularization of end–to–end learning: Case study in arcade games. arXiv 2019, arXiv:1901.09895.

[27] Polvara, R.; Patacchiola, M.; Sharma, S.; Wan, J.; Manning, A.; Sutton, R.; Cangelosi, A. Autonomous quadrotor landing using deep reinforcement learning. arXiv 2017, arXiv:1709.03339.

[28] Pan, X.; You, Y.; Wang, Z.; Lu, C. Virtual to real reinforcement learning for autonomous driving. arXiv 2017, arXiv:1704.03952.

[29] Loiacono, D.; Cardamone, L.; Lanzi, P. Simulated car racing championship: Competition software manual. arXiv 2013, arXiv:1304.1672.

[30] Mnih, V.; Kavukcuoglu, K.; Silver, D.; Graves, A.; Antonoglou, I.; Wierstra, D.; Riedmiller, M. Playing atari with deep reinforcement learning. arXiv 2013, arXiv:1312.5602.

[31] Synnaeve, G.; Nardelli, N.; Auvolat, A.; Chintala, S.; Lacroix, T.; Lin, Z.; Richoux, F.; Usunier, N. Torchcraft: A library for machine learning research on real–time strategy games. arXiv 2016, arXiv:1611.00625.

[32] Peng, P.; Wen, Y.; Yang, Y.; Yuan, Q.; Tang, Z.; Long, H.; Wang, J. Multiagent bidirectionally–coordinated nets: Emergence of humanlevel coordination in learning to play starcraft combat games. arXiv 2017, arXiv:1703.10069.

[33] Foerster, J.; Farquhar, G.; Afouras, T.; Nardelli, N.; Whiteson, S. Counterfactual multi–agent policy gradients. In Proceedings of the AAAI Conference on Artificial Intelligence, New Orleans, LA, USA, 2–7 February 2018; Volume 32.

[34] Logothetis, N. The ins and outs of fmri signals. Nat. Neurosci. 2007, 10, 1230–1232. [CrossRef] [PubMed].

[35] Oh, J.; Singh, S.; Lee, H.; Kohli, P. Zero–shot task generalization with multi–task deep reinforcement learning. In Proceedings of the International Conference on Machine Learning, Sydney, NSW, Australia, 6–11 August 2017; pp. 2661–2670.

[36] Brockman, G.; Cheung, V.; Pettersson, L.; Schneider, J.; Schulman, J.; Tang, J.; Zaremba, W. OpenAI Gym, 2016. arXiv 2017, arXiv:1606.01540.

[37] Xiong, Y.; Chen, H.; Zhao, M.; An, B. Hogrider: Champion agent of microsoft malmo collaborative ai challenge. In Proceedings of the AAAI Conference on Artificial Intelligence, New Orleans, LA, USA, 2–7 February 2018; Volume 32.

[38] Silver, D.; Hubert, T.; Schrittwieser, J.; Antonoglou, I.; Lai, M.; Guez, A.; Lanctot, M.; Sifre, L.; Kumaran, D.; Graepel, T. Mastering chess and shogi by self–play with a general reinforcement learning algorithm. arXiv 2017, arXiv:1712.01815.

[39] David, O.; Netanyahu, N.; Wolf, L. Deepchess: End–to–end deep neural network for automatic learning in chess. In Proceedings of the International Conference on Artificial Neural Networks, Barcelona, Spain, 6–9 September 2016; Springer: Berlin/Heidelberg, Germany, 2016; pp. 88–96.

[40] Choudhary, A. A Hands-On Introduction to Deep Q-Learning Using OpenAI Gym in Python. Anal Vidhya. 2019. Available online: https://www.analyticsvidhya.com/blog/2019/04/introduction-deep-q-learning-python/.

[41] El-Ghazali Talbi. Metaheuristics: from design to implementation, volume 74. John Wiley & Sons, 2009.

[42] Seyedali Mirjalili and Andrew Lewis. The whale optimization algorithm. Advances in engineering software, 95:51–67, 2016.

[43] Erik Cuevas, Fernando Fausto, and Adrián González. New Advancements in Swarm Algorithms: Operators and Applications. Springer,2020.

[44] Ümit Can and Bilal Alataş. Physics based metaheuristic algorithms for global optimization. 2015.

[45] Meeta Kumar and Anand J Kulkarni. Socio-inspired optimization metaheuristics: a review. In Socio-cultural inspired metaheuristics, pages 241–265. Springer, 2019.

[46] Hashim, F. A., & Hussien, A. G. (2022). Snake Optimizer: A novel meta-heuristic optimization algorithm. Knowledge-Based Systems, 242, 108320.

[47] van Eck, N.J.; van Wezel, M. Application of Reinforcement Learning to the Game of Othello. Comput. Oper. Res. 2008, 35, 1999–2017.

[48] Mnih, V.; Kavukcuoglu, K.; Silver, D.; Graves, A.; Antonoglou, I.; Wierstra, D.; Riedmiller, M. Playing Atari with Deep Reinforcement Learning. arXiv 2013, arXiv:1312.5602.

[49] Rajendran, D.; Santhanam, P. Towards Digital Game-Based Learning Content with Multi-Objective Reinforcement Learning. Mater. Today Proc. 2021, 2214–7853.

[50] Liu, S.; Cao, J.; Wang, Y.; Chen, W.; Liu, Y. Self-Play Reinforcement Learning with Comprehensive Critic in Computer Games. Neurocomputing 2021, 449, 207–213.

[51] Silver, D.; Huang, A.; Maddison, C.J.; Guez, A.; Sifre, L.; Van Den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; et al. Mastering the Game of Go with Deep Neural Networks and Tree Search. Nature 2016, 529, 484–489.

[52] Silver, D.; Hubert, T.; Schrittwieser, J.; Antonoglou, I.; Lai, M.; Guez, A.; Lanctot, M.; Sifre, L.; Kumaran, D.; Graepel, T.; et al. A General Reinforcement Learning Algorithm That Masters Chess, Shogi, and Go through Self-Play. Science 2018, 362, 1140–1144.

[53] Núñez-molina, C.; Fernández-olivares, J.; Pérez, R. Learning to Select Goals in Automated Planning with Deep-Q Learning. Expert Syst. Appl. 2022, 202, 117265.

[54] Gong, X.; Yu, J.; Lü, S.; Lu, H. Actor-Critic with Familiarity-Based Trajectory Experience Replay. Inf. Sci. 2022, 582, 633–647.

[55] Wang, M.; Wang, L.; Yue, T. An Application of Continuous Deep Reinforcement Learning Approach to Pursuit-Evasion Differential Game. In Proceedings of the 2019 IEEE 3rd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC), Chengdu, China, 15–17 March 2019; pp. 1150–1156.

[56] Such, F.; Madhavan, V.; Conti, E.; Lehman, J.; Stanley, K.; Clune, J. Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning. arXiv 2017, arXiv:1712.06567.

[57] Hasselt, H.; Guez, A.; Silver, D. Deep reinforcement learning with double q–learning. In Proceedings of the AAAI Conference on Artificial Intelligence, Phoenix, AZ, USA, 12–17 February 2016; Volume 30.

[58] OpenAI; Berner, C.; Brockman, G.; Chan, B.; Cheung, V.; Debiak, P.; Dennison, C.; Farhi, D.; Fischer, Q.; Hashme, S.; et al. Dota 2 with large scale deep reinforcement learning. arXiv 2019, arXiv:1912.06680.

[59] Vinyals, O.; Fortunato, M.; Jaitly, N. Pointer networks. arXiv 2015, arXiv:1506.03134.

[60] Lee, J.; Lee, Y.; Kim, J.; Kosiorek, A.; Choi, S.; Teh, Y.W. Set Transformer: A Framework for Attention-based Permutation-Invariant Neural Networks. In Proceedings of the International Conference on Machine Learning, Stockholmsmässan, Sweden, 15–19 July 2018.

[61] Ammanabrolu, P.; Riedl, M.O. Playing Text–Adventure Games with Graph–Based Deep Reinforcement Learning. arXiv 2018, arXiv:1812.01628.

[62] Adolphs, L.; Hofmann, T. LeDeepChef: Deep Reinforcement Learning Agent for Families of Text–Based Games. In Proceedings of the AAAI Conference on Artificial Intelligence, Honolulu, HI, USA, 27–28 January 2019.

[63] Brown, N.; Bakhtin, A.; Lerer, A.; Gong, Q. Combining Deep Reinforcement Learning and Search for Imperfect–Information Games. arXiv 2020, arXiv:2007.13544.

[64] Ye, D.; Liu, Z.; Sun, M.; Shi, B.; Zhao, P.; Wu, H.; Yu, H.; Yang, S.; Wu, X.; Guo, Q.; et al. Mastering Complex Control in MOBA Games with Deep Reinforcement Learning. Proc. AAAI Conf. Artif. Intell. 2020, 34, 6672–6679.

[65] Roy, B. An analysis of temporal–difference learning with function approximation. Autom. Control. IEEE Trans. 1997, 42, 674–690.

[66] Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A.; Veness, J.; Bellemare, M.; Graves, A.; Riedmiller, M.; Fidjeland, A.; Ostrovski, G. Human–level control through deep reinforcement learning. Nature 2015, 518, 529–533.

[67] Azizi, M., Aickelin, U., A. Khorshidi, H., & Baghalzadeh Shishehgarkhaneh, M. (2023). Energy valley optimizer: a novel metaheuristic algorithm for global and engineering optimization. Scientific Reports, 13(1), 226.