IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 4 June 2026, accepted 4 July 2026, date of publication 9 July 2026, date of current version 16 July 2026.

Digital Object Identifier 10.1109/ACCESS.2026.3711834

RESEARCH ARTICLE

Comparing the Performance of Off-Policy Deep Reinforcement Learning and Neuroevolution in Stealth Game Problems

PEDRO F. SILVA[1], JACINTO ESTIMA[1], INÊS NOBRE[2], AND EDIRLEI SOARES DE LIMA[3]
[1]Department of Informatics Engineering, CISUC/LASI, University of Coimbra, 3004-531 Coimbra, Portugal
[2]CIPER, Faculty of Human Kinetics, University of Lisbon, 1499-002 Oeiras, Portugal
[3]Academy for AI, Games and Media, Breda University of Applied Sciences, 4817 JS Breda, The Netherlands

Corresponding author: Pedro F. Silva (pedrosilva333@outlook.pt)

This work was supported in part by the National Funds through Foundation for Science and Technology (FCT), I.P., within the Scope of the Research Unit UID/00326—Centre for Informatics and Systems, University of Coimbra (https://doi.org/10.54499/UID/00326/2025)

ABSTRACT Deep Learning is a rapidly evolving field with the potential to revolutionize the video game industry. However, its application remains limited, particularly in unexplored genres such as stealth games. Stealth games present unique challenges, including sparse rewards, long-term planning, and complex state dependencies. This paper presents a comparative evaluation of off-policy Deep Reinforcement Learning (DRL) and Neuroevolution (NE) algorithms in stealth game problems. To conduct this study, three distinct stealth game scenarios were developed using the Unity game engine. Eighteen algorithms were implemented and systematically optimized through hyperparameter tuning. Their performance was evaluated based on learning stability, adaptability, and task completion effectiveness. The results show that, in most cases, DRL methods and NE approaches were able to learn stealth-oriented behaviors, although differences in performance were observed. Among DRL approaches, Rainbow DQN achieved the most consistent and robust results, while Genetic Algorithms outperformed other methods within the NE category. These findings provide new insights into the relative strengths of learning-based approaches in stealth game environments.

INDEX TERMS Deep learning, deep reinforcement learning, neuroevolution, stealth games, video games.

I. INTRODUCTION

Over the last decade, Deep Learning (DL) has become one of the most popular topics in Artificial Intelligence (AI) and Machine Learning (ML) due to its success in solving regression and classification problems. It can generalize large amounts of data with the use of Artificial Neural Networks (ANN).

Traditionally, game developers have relied on classic game AI techniques like finite state machines and behavior trees, which require manually scripting all intended behaviors. This process demands extensive planning, as developers must account for every possible interaction an AI agent might have. As these AI systems grow in complexity, they become increasingly difficult to manage. Recognizing these challenges and the significant advancements in DL, game companies, such as Electronic Arts [1] and Ubisoft [2], began exploring how DL could enhance AI in video games.

Automated learning in game environments has gained traction in recent years, particularly through Reinforcement Learning (RL). RL models enable AI agents to analyze their environment and take actions that maximize future rewards. When combined with DL, this approach forms the field of Deep Reinforcement Learning (DRL), which has shown state-of-the-art results across various video game genres [3].

Another approach to learning in video games is Neuroevolution (NE), which optimizes ANNs using evolutionary algorithms. NE has a long history of success in training AI to solve complex video game challenges [4].

However, DRL and NE models are often tested within standardized research platforms, such as Gymnasium [5], which primarily feature a limited set of game environments.

The associate editor coordinating the review of this manuscript and approving it for publication was Domenico Rosaci.

While these platforms facilitate benchmarking and comparison among researchers, they do not fully represent the diversity of video game genres. For example, stealth games – a widely recognized yet underrepresented genre –are largely absent from these platforms.

Stealth games share common core mechanics despite their variations. At their core, they involve balancing two opposing states: sneaking, where the player must navigate undetected, and detection, which happens when the player is compromised/detected and must evade enemies until the game returns to the sneaking state [6], [7]. Other common traits in stealth games involve level exploration to achieve objectives, the freedom to employ different strategies (e.g., sneak past or risk disabling enemies), and limited knowledge of the environment (e.g., level layout, uncertainty regarding enemy patrol routes and behaviors). These traits introduce challenges commonly encountered when training DRL and NE agents, such as the exploration-exploitation dilemma, sparse or deceptive reward structures, partial observability, and long-term planning [8], [9].

Given the unique challenges of this genre, investigating the effectiveness of DRL and NE in stealth-based scenarios presents a valuable research opportunity. In this work, we evaluate the performance of off-policy DRL algorithms, such as Deep Q-Networks (DQN) and its variants, and NE models within stealth video game environments, focusing on their ability to explore and navigate environments while minimizing enemy detection.

The remainder of this paper is organized as follows. Section II reviews related work and describes the contributions of this paper. Section III provides an overview of the deep learning methods used in this study. Section IV outlines the research methodology, including the experimental setup. Section V presents and discusses the results, and Section VI concludes the paper and suggests potential directions for future research.

The full project and code created for this research can be accessed at: https://github.com/rorix14/Off-Policy-DRL-and-NE-in-Stealth-Game-Problems. A permanently archived version of the repository is also available via Zenodo at https://doi.org/10.5281/zenodo.20418839

II. RELATED WORK

Over the past decade, DL research applied to video games has increased significantly. Works such as [4], [10], and [11] showed that DL algorithms can be used to train AI agents capable of playing certain video games at a superhuman level. These results suggest that DL has the potential to generalize across various game environments, making it a valuable tool for both video game development and research.

DL research in video games currently spans a broad range of application domains. Prominent areas include procedural content generation (PCG), 3D navigation, and automated gameplay testing. In PCG, DL algorithms are used to generate game content, such as levels, textures, and characters [12], [13]. In 3D navigation, AI agents are trained to achieve realistic navigation in complex and realistic video game environments [2], [14]. Automated gameplay testing uses DL models to explore game environments to detect bugs and evaluate game mechanics [1], [15], [16], [17]. Beyond these core areas, DL has also been applied to player behavior modelling [18], dynamic difficulty adjustment [19], general game playing [3], [20], among others.

To support these research areas, several platforms have been developed to simulate different video game environments across many genres. For example, Gymnasium [5] provides several 2D arcade-style video games, while ViZDoom [21] offers 3D environments in a first-person shooter setting. Additionally, other platforms implement a range of different environments, such as car racing, 3D platformers, and puzzle games, using the Unity and Godot game engines [22], [23], [24]. These platforms, combined with open-source DRL and NE libraries, such as CleanRL [25], Stable-Baselines3 [26], and RLlib [27], have facilitated the development and evaluation of DL-based AI agents in video games.

Despite these advancements, some video game genres, such as stealth games, remain underrepresented in DL research, as mentioned in Section I. Thus, the effectiveness of DL algorithms in solving stealth game environments remains largely unexplored.

While reviewing the literature, we identified only two research articles that apply DL in the context of stealth video games. The first combines behavior trees with RL to train agents in a stealth environment, focusing on automated gameplay testing rather than on evaluating learning algorithms in stealth-specific scenarios, and considers only a single on-policy algorithm constrained by hand-programmed behavior structures [28]. The second applies genetic algorithms exclusively to evolve reward functions for a tabular RL agent in a simplified 2D stealth simulation, without employing deep learning methods or comparing multiple learning approaches [29].

Consequently, no systematic evaluation of state-of-the-art off-policy DRL and NE algorithms in dedicated stealth game environments currently exists in the literature, and the relative strengths and weaknesses of these two learning paradigms in the context of stealth-specific challenges, such as sparse rewards, long episodes, partial observability, and strategic trade-offs, remain unknown.

To address this gap, the main contributions of this paper are twofold:

1) The design of stealth-based test environments that capture core challenges of the genre, including domain knowledge integration, state representation, and reward structure formulation, providing a dedicated benchmark for evaluating DL algorithms in stealth game scenarios.
2) A comparative evaluation of off-policy DRL and NE algorithms across the proposed stealth scenarios, providing insights into the relative strengths and

weaknesses of each learning paradigm under the specific conditions that stealth games impose.

III. DEEP LEARNING METHODS

In this section, we provide an overview of the DL algorithms applied in our research to address challenges commonly found in stealth video games. These algorithms encompass various advancements in off-policy DRL and NE methods.

A. OFF-POLICY DEEP REINFORCEMENT LEARNING

RL is inspired by the idea of learning through trial and error from performing actions in an environment, similar to how animals and humans learn. For example, when training a dog, a common approach is to reward it with treats when it performs the desired action in response to a command, reinforcing the association between commands and actions. Similarly, RL aims to enable an artificial agent to improve its performance on a given task over time. Since ANNs are powerful function approximators, DRL extends RL techniques to train them, allowing agents to learn policies with better generalization capabilities.

DRL methods are broadly categorized into two main approaches: on-policy and off-policy learning. On-policy methods learn exclusively from data generated by the current policy, requiring continuous interaction of the current policy with the environment, discarding experiences once the policy is updated. In contrast, off-policy methods can learn from experiences generated by multiple policies, including older versions of the current policy or random exploration, enabling the reuse of previously collected transition data across training iterations.

In this research, we focus exclusively on off-policy algorithms, specifically the Deep Q-Network (DQN) and several of its extensions. This choice is motivated by key characteristics of stealth game environments that align well with the strengths of off-policy learning. First, stealth environments typically involve long episodes with sparse reward signals, where meaningful outcomes such as successfully reaching the goal are rarely encountered. The ability of off-policy methods to store and repeatedly learn from such rare but informative transitions, via a replay buffer, makes them particularly well suited to this setting. Second, stealth gameplay inherently involves risk-reward trade-offs, where an agent must decide between safer stealth navigation and riskier but potentially higher-rewarding actions such as disabling enemies. DQN-based algorithms are well-positioned to handle this through their direct optimization toward the optimal policy, which can encourage the pursuit of riskier strategies. Also, their use of an ε -greedy exploration strategy ensures random exploration throughout training, which can potentially lead to the discovery of higher reward-states that involve risk-taking policies.

1) DEEP Q-NETWORKS

As a value-based RL method, the DQN algorithm extends traditional Q-Learning by using ANNs to approximate the Q-value function, denoted as $Q_\theta(s, a)$, where s and a represent the current state and action, respectively, while θ represents the network's weights and biases [10]. The objective is to adjust the θ parameters to minimize the difference between the predicted Q-values and the target Q-values, as expressed by the gradient descent update step:

$$\theta = \theta - \alpha(r + \gamma \max_{a'} Q_\theta(s', a') - Q_\theta(s, a))\nabla Q_\theta(s, a). \quad (1)$$

In (1), α is a hyperparameter called Learning rate (Lr), $r + \gamma \times \max_{a'} Q_\theta(s', a')$ represents the target value y, r denotes the reward, γ is the discount factor, and s' and a' are the subsequent state and action, respectively.

To enhance ANN training stability and reliability, DQN incorporates two key techniques. The first is a replay memory buffer, which ensures that training data is independent and identically distributed. This buffer stores agent transition data (states, actions, rewards), and allows the network to sample random batches for training [10]. Once the buffer reaches its capacity, older transitions are replaced with newer ones, preventing the model from overfitting to older experiences. The second technique involves using two separate networks: an online network, which interacts with the environment and updates continuously, and a target network, which estimates target values y. The target network is updated every n steps by copying the weights and biases of the online network. This approach mitigates the moving target problem by keeping the target values stable over short periods, reducing instability over training.

Under this strategy, the agent selects the action with the highest estimated Q-value with probability 1 - ε, while choosing a random action with probability ε [30]. To reduce excessive exploration during later stages of training, the value of ε is typically decayed over time. This gradually shifts the agent's behavior from exploration toward exploitation, facilitating convergence to a stable and near-deterministic policy.

2) DOUBLE DQN

By using the target network to calculate the y value, the DQN algorithm can overestimate the Q-values. In the original Double DQN paper [31], the authors state that the overestimation comes from the max operation in the Bellman equation, as it uses the same values to both select and evaluate an action. The authors suggest decoupling the selection from the evaluation to prevent this overestimation. This can be achieved by using the online network Q_θ to select the action, but having the target network $\hat{Q}_{\theta'}$ evaluate it. Now the target Q-value expression can be written as

$$y = r + \gamma \times \hat{Q}_{\theta'}(s', \operatorname*{argmax}_{a'} Q_\theta(s', a')). \quad (2)$$

3) DUELING DQN

The Dueling DQN architecture comes from the idea that the Q-values that a DQN model tries to approximate can

be divided into two properties, the state value $V(s)$, and the state-action advantage value $A(s, a)$ [32]. By doing this split, the model improves training stability and accelerates convergence. The state value $V(s)$ represents a state's expected reward, and the advantage value $A(s, a)$ says how much of an improvement an action value is over the expected state reward.

To predict these two different functions (the value and advantage) the authors suggest an architecture that contains two streams, one that predicts the value function, and another that predicts the advantage. The sum of these two streams forms the Q-value function that can now be described as

$$Q(s, a) = V(s) + A(s, a) - \frac{1}{|A|} \sum_{a'} A(s, a'). \quad (3)$$

4) N-STEP DQN

The concept of N-step was first introduced by Sutton, it combines Temporal Difference methods that learn by doing a single step at a time, and methods that use the complete trajectory to learn, like Monte Carlo methods [33]. The N-step method serves as a middle ground between these two approaches, as it can unroll the trajectory for any number of steps. The intuition is that all Q-values will converge faster the more n-steps there are. Because the DQN algorithm does some random exploration, having a small n-step n value is better for stable convergence [34]. The N-step DQN algorithm generalizes the $Q(s, a)$ function for any size of n as

$$Q(s_t, a_t) = \gamma^n \times \max_{a'} Q\left(s_{t+n}, a'\right) + \sum_{k=0}^{n-1} \gamma^k r_{t+k+1}. \quad (4)$$

5) DQN WITH PRIORITIZED EXPERIENCE REPLAY

DQN with Prioritized Experience Replay (PER) improves the learning efficiency of the DQN algorithm by prioritizing important experiences during the training process [35]. PER assigns a priority value P to each experience i, defining its probability of being sampled. The priority value of each experience is based on how well the network's Q-values predicted the observed reward, which means that experiences with high errors are assigned higher priority since they are more likely to improve performance. Thus, each experience priority value is calculated as

$$P(i) = \frac{p_i^\eta}{\sum_k p_k^\eta}, \quad (5)$$

where p_i is the obtained Q loss for i-th experience, and η is a hyperparameter within the range [0, 1].

Because the sampling of experiences is influenced by the priority values, it will introduce biases during training since batches are no longer independent and identically distributed. To correct this bias, the original authors [35] suggest multiplying importance-sampling weights w_i with the Q loss/error of the experience, as in

$$w_i = (N * P(i))^{-\beta}, \quad (6)$$

$$w_i p_i \nabla Q_\theta(s_i, a_i), \quad (7)$$

with N signifying the total number of experiences stored in the replay memory buffer, while β is a hyperparameter within the range [0, 1], which slowly increases to 1, fully compensating for the bias introduced.

6) NOISY NETWORKS DQN

Noisy DQN introduces a new form of exploration that replaces the traditional ε-greedy strategy [36]. While ε-greedy exploration is effective in simpler environments with short episodes, it often struggles in environments with sparse rewards, where a large sequence of actions is required to achieve a reward. To address this limitation, the Noisy DQN algorithm introduces noise directly into the network's weights w and biases b, adding a random component to the network's output. Specifically, the Noisy DQN algorithm adds new sigma σ weights and biases parameters, to the equation of an ANN linear layer. Each of these new parameters gets multiplied by random numbers ε taken from a Gaussian distribution. The function for a noisy linear layer now becomes

$$y = \left(w + \sigma^w \varepsilon^w\right) x + b + \sigma^b \varepsilon^b. \quad (8)$$

The sigma σ parameters are also updated during backpropagation, just like the network's other parameters. This allows the network to learn how to use the noise effectively to explore the state-action space, while also learning to process the Q-values.

7) CATEGORICAL DQN

In the traditional DQN approach, the goal is to predict the expected return for each state-action pair. The Categorical DQN (C-51) algorithm, which belongs to the subfield of distributional RL, tackles this limitation by representing the value function as a discrete distribution over fifty-one possible return values [37]. This approach takes uncertainty into account, leading to more informed decision-making. The C-51 algorithm estimates reward distribution by defining a set of 'atoms' within a specified range of minimum and maximum possible reward values. These atoms, evenly spaced to form a support vector, are assigned probabilities by the model, capturing the full distribution of possible return values. In practice, instead of outputting a single scalar value, for every action the C-51 algorithm outputs a vector of probabilities of the same size as the support vector. Each of these probabilities corresponds to the probability of the reward value of the same index in the support vector.

To accommodate these changes, the target value y is now the distribution of the max target network distribution projected over the current reward value onto the support vector. Since both predictions and targets are probability distributions, the loss function is given by the categorical cross-entropy between them.

8) RAINBOW DQN

Rainbow DQN is one of the most advanced variants of the DQN algorithm, it combines all the previously mentioned

DQN extensions to improve the stability and performance of the algorithm during training [34]. Rainbow DQN was introduced by researchers at DeepMind in 2017, where it managed to produce state-of-the-art results on the Atari 2600 benchmark, both in terms of data efficiency and final performance.

B. NEUROEVOLUTION

NE has a rich history of application in the video game industry [4], and has demonstrated to be a competitive alternative to DRL for solving game environments [38], [39]. NE optimizes ANNs using techniques from Evolutionary Algorithms (EAs), its objective is to optimize a target function $f(x)$ (or Fitness function in the context of NE) in a closed-box manner, without making assumptions about its internal structure. Unlike DRL, NE is only concerned with the output of f, not its underlying mechanics. This allows NE to optimize a wide range of functions, including those that are non-differentiable, setting it apart from DRL.

NE also diverges from DRL in its approach to training ANNs by drawing inspiration from biological evolution rather than relying on back-propagation. NE adopts a training paradigm modeled after natural selection, which gives rise to a structured cycle encompassing three main steps, population initialization, fitness evaluation, and selection:

1) Population initialization creates a population of individuals, represented in this research by ANNs. First-generation individuals are created with an identical network structure, but with random weight and bias values, while subsequent generations are derived according to the selection criteria.
2) Fitness evaluation assigns a performance score to each individual after running a full episode. Unlike the DRL algorithms presented in this paper, which learn from the individual samples they collect during training, NE algorithms evaluate their agents based on episode fitness, which in this research is represented by the total rewards collected by an individual during an episode.
3) Selection determines which and how current individuals contribute to the creation of the next generation. This process always takes into consideration the fitness scores assigned to individuals, but its mechanics vary among algorithms. Most commonly, the selection process is followed by two methods: crossover and mutation. Crossover combines two selected individuals to generate a new one, while mutation introduces random alterations to the new individual. In this research, mutations are applied by perturbing weights and biases with Gaussian noise, where the Noise std parameter controls the standard deviation of the distribution and thus the magnitude of these perturbations.

These steps are repeated until a termination condition is met, such as an individual reaching a predefined fitness threshold or a generation limit being reached.

In the context of this work, the inclusion of NE is further motivated by the specific challenges posed by stealth game environments. As previously mentioned, these environments are characterized by sparse and delayed rewards, as well as potentially deceptive reward structures, where locally optimal behaviors, such as consistently avoiding enemies, may prevent the discovery of globally optimal strategies that require risk-taking. Such conditions are known to be challenging for DRL methods, which rely heavily on reward signals to guide policy updates. In contrast, NE algorithms learn based on the total return of a complete episode, making them invariant to the specific timing of rewards, while also maintaining a population of candidate solutions, enabling a broader exploration of the policy space. These characteristics allow NE methods to potentially escape local optima and discover diverse behavioral strategies.

1) RANDOM SEARCH

Random Search (RS) algorithms can be implemented in various ways, but for this research, we adopted a method that loosely follows NE concepts. Our version of RS starts the training process by creating a single ANN, from which a population is derived by applying small perturbations to its network weights and biases parameters. After evaluation, the network with the best fitness score is used to produce the next population of perturbed/noisy networks. This process repeats until a termination condition is met. This variant provides a consistent baseline while enabling a more controlled exploration of the search space, compared to other RS methods such as always initializing individuals with fully random weights.

2) COVARIANCE MATRIX ADAPTATION EVOLUTIONARY STRATEGY

The Covariance Matrix Adaptation Evolutionary Strategy (CMA-ES) algorithm belongs to a subfield of EAs called Evolutionary Strategies. CMA-ES is similar to the RS algorithm described previously, but with a few key differences. Specifically, it samples its perturbation noise ε from a Gaussian distribution, with each weight and bias θ being summed with the corresponding noise, scaled by a noise standard deviation (*Noise std*) τ [4]. This transforms the linear network equation into a similar one used by Noisy DQN (8). Following the evaluation phase, the CMA-ES algorithm updates the original weights and biases values by factoring in the fitness score of each individual i, multiplied by the corresponding noise, as in

$$\theta = \theta + \alpha \frac{1}{n\tau} \sum_{i=1}^{n} f\,(\theta + \tau\varepsilon_i)\varepsilon_i. \tag{9}$$

The intuition behind this is to steer the network's weights and biases toward the direction of the best-performing individuals. According to other research [38], CMA-ES can make use of two other techniques for improved performance. These are mirrored sampling, used for sampling noise when creating a new population, and rank transformation, used to normalize the fitness scores within a population.

3) GENETIC ALGORITHM

Genetic algorithms (GAs) are among the more versatile NE algorithms due to the numerous variants available [40], [41], [42]. In the context of this research, the following methods were used for the selection, crossover, and mutation steps: tournament selection, single-point crossover, and noisy injection. Tournament selection is used to select which individuals perform the crossover operation, it does this by randomly choosing several individuals and selecting the two that have the best fitness values from that group. Single-point crossover combines the list of ANN parameters (θ) per layer of two individuals at a random crossover point to create an offspring. Noisy injection selects a small percentage of an offspring's genes (weights or biases) and adds to them a random value sampled from a Gaussian distribution.

Additionally, the GA algorithm used in this work also makes use of two other techniques, elitism, and adaptive mutation. Elitism ensures a few top performers advance to the next generation, stabilizing GA performance. Adaptive mutation adjusts the mutation rate of an offspring based on the fitness of its parents: offspring from fitter parents receive a lower mutation rate, whereas offspring from less fit parents receive a higher mutation rate. This promotes exploration, while preserving high-quality solutions [43].

4) NEUROEVOLUTION OF AUGMENTING TOPOLOGIES

While GAs and CMA-ES try to find the optimal network weights and biases through evolutionary methodologies, they are unable to change their network structure/topology during training. Neuroevolution of Augmenting Topologies (NEAT) addresses this limitation by introducing a set of rules that allow it to evolve network topologies through the training process, alongside all the other ANN parameters, enabling incremental complexification of architectures alongside parameter optimization [44].

The main technique that NEAT employs to evolve ANNs is through expanding its mutation methods to not only alter the values of existing ANN weights but also modify their topology by adding new neurons and connection weights.

Evolving ANN topologies makes applying crossover methods more complex than in fixed-topology algorithms, because parent networks may have different architectures. Unlike GAs, there is no one-to-one similarity between their genes: a weight connection or neuron/node present in one parent may not exist in the other. NEAT solves this issue by introducing innovation numbers, which uniquely label every new gene (nodes and connections) when created through mutation. These labels allow the algorithm to keep track of the historical origin of genes, making it possible to align corresponding genes between parents during crossover.

NEAT begins training with minimal ANNs, consisting only of input and output nodes without any hidden nodes or extra connections. This starting point allows the algorithm to explore the search space gradually, adding complexity when the structural mutations prove beneficial through improved fitness score. By starting simple, NEAT avoids initially searching in large dimensions, ensuring that complexity is introduced progressively as needed, while also providing a computational performance advantage when compared to fixed topology algorithms.

When a new ANN topology is introduced, it often performs poorly at first, even though it may eventually surpass older topologies. This initial drop in performance usually leads to its elimination during selection. To address this challenge, the NEAT algorithm implements the concept of speciation. Speciation groups topologies into species using a compatibility distance metric that measures topology similarity. At the start of each generation, each topology is assigned to a species based on its similarity to a representative network. Within each species, NEAT applies explicit fitness sharing, where an individual's fitness is divided among the members of its species. The resulting shared score determines how many offspring the species will produce in the next generation. This mechanism prevents a single large, well-performing species from dominating the population and ensures that smaller, newer species have a chance to reproduce, reducing the risk of prematurely discarding promising but underdeveloped innovations.

5) NOVELTY SEARCH

Typically, fitness functions employed in NE are tied to the mechanics of the environment they aim to solve. These types of fitness functions are commonly referred to as goal-oriented or objective functions [45]. However, this type of fitness function can prove to be deceptive, depending on the environment dynamics. Deceptive fitness functions create search landscapes where the optimization process is misled toward suboptimal solutions, often due to local optima. An example is the Hard Maze problem [46], where the fitness function rewards agents based on their proximity to the goal. This can cause agents to get stuck in dead ends or deceptive paths within the maze. Without strong exploration pressure, agents may never discover the true path to the goal.

To achieve better performance in a deceptive environment, a different type of fitness function should be used. Novelty Search (NS) is a type of fitness function that rewards agents for generating novel behaviors that have not been previously encountered throughout the evolution process. To assess this, NS uses a novelty metric to understand the uniqueness of an individual's behavior, this metric can quantify the dissimilarity between an individual's behavior with that of any other behavior. The novelty score of a behavior is determined by computing the mean novelty metric across the k-nearest neighbors, which are sorted based on their behavioral distance from that specific behavior [46]. The higher the novelty score, the more unique the behavior. During the evolutionary process, the most novel behaviors are stored in an archive, allowing them to be used in calculating the novelty score of future individuals. In practice, the behavior of an agent is domain-specific, as such its characterization must be defined

by the practitioner. For example, in maze-like environments, an agent's behavior can be defined by its final $\{x, y\}$ position after completing an episode [46]. In this case, the Euclidean distance function can be used to measure the behavioral distance between two agents.

Despite the success of NS [39], further research suggests that combining NS with the objective fitness function leads to better results [47], [48], [49], [50]. While this can be done in different ways, this research adopts the method introduced in [47], due to its strong performance when compared to other approaches [50]. This method consists of linearly combining the normalized fitness score with the normalized novelty score, with a parameter novelty relevance λ, within the range [0, 1], serving to govern the impact of the fitness and novelty in the algorithm, as seen in

$$score\,(i) = (1 - \lambda) \times \overline{fit}\,(i) + \lambda \times \overline{nov}\,(i)\,. \qquad (10)$$

In this research, we combine NS with the other NE algorithms presented in this section (RS, CMA-ES, GA, NEAT). The combination is controlled by the value of λ, where $\lambda = 1$ corresponds to pure NE and $\lambda = 0$ to the baseline algorithm. For clarity, we denote combined methods by appending "-NS" to the algorithm name (e.g., GA-NS), while the original abbreviation is used when $\lambda = 0$.

IV. METHODOLOGY

A. TESTING ENVIRONMENT AND SCENARIO DESIGN

Stealth game environments pose a combination of challenges not typically captured by standard RL benchmarks such as Gymnasium [5]. While Gymnasium and similar platforms provide a wide range of environments, they lack dedicated stealth-game scenarios and fail to replicate the specific reward sparsity and exploration dynamics this genre demands. In stealth games, meaningful rewards are infrequent and require agents to sustain coherent behavior over significantly longer episodes than those found in most common benchmarks, placing greater demands on long-term planning. Furthermore, the state representation in these environments is inherently dynamic (e.g., enemy cones of vision and their occlusion by the level layout continuously alter what information is available to the agent). Finally, the balance between cautious stealth navigation and the risk of disabling enemies for higher rewards introduces strategic trade-offs that require agents to reason about long-term consequences. Together, these characteristics create reward dynamics that can incentivize fundamentally different behaviors depending on each algorithm's exploration mechanics, making stealth games a particularly valuable and underexplored testbed for evaluating DRL and NE methods. Despite this, designing environments with these dynamics requires a high degree of environmental flexibility to be modeled properly.

While platforms such as the Atari Learning Environment [5] have played an important role in benchmarking DRL methods, they provide limited flexibility for designing custom and complex environments. By developing the testing environments within a commercial-grade game engine, like Unity, we gain greater control over experimental parameters and environmental detail [24]. This approach allows us to build state and reward structure representations directly from in-game data that capture the stealth-specific dynamics described above. As such, to evaluate the performance of the DRL and NE algorithms discussed in Section III, three distinct stealth levels were designed and implemented in the Unity game engine, as shown in Fig. 1.

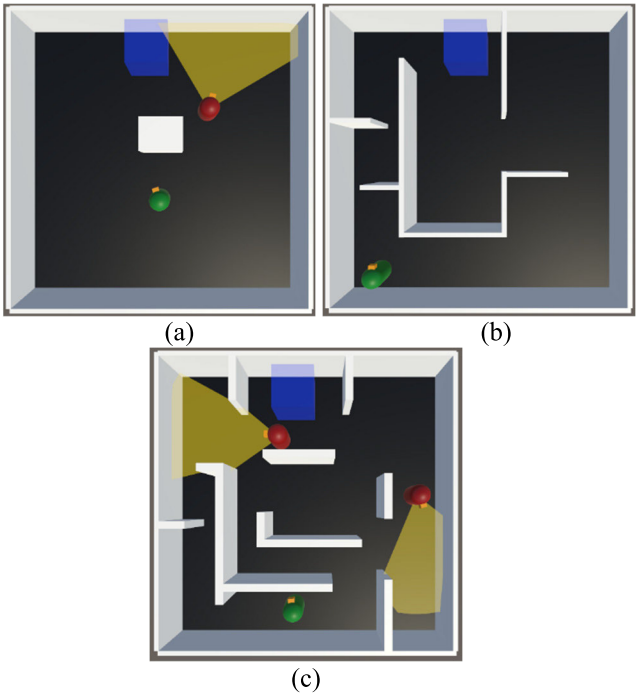

FIGURE 1. The three stealth game levels: level one (a), level two (b), level three (c). The learning agent is in green, enemies in red, enemy cone of vision in yellow, level goal in blue, and walls in white. The images show the starting states of each level.

The game adopts a 3D top-down perspective, where an agent must reach the goal of the level within a time limit while avoiding detection by patrolling enemies. The agents' available actions are limited to movement and "disabling" enemies. Movement can be executed in eight directions: left, right, forward, backward, left-forward, left-backward, right-forward, and right-backward, simulating keyboard-based navigation. The "disable" action can only be performed when the agent character is undetected and near an enemy. The agent might also choose to take no action, bringing the total number of possible actions to ten.

Each level has three possible terminal states: the agent reaches the level goal, is detected by an enemy, or runs out of time. Levels states are represented by the following features: the x and z coordinates of the level goal, the agent's x and z coordinates, an array of player-to-wall x and z coordinates captured at 30 incremented 12° angle centered around the agent, where 12° increments were chosen to provide full 360° coverage while providing strong environment representation, the x and z coordinates of all enemies, and the x and z coordinates corresponding to the endpoints of each enemy's

line of sight, for a total of 15 different points per enemy. All feature values are normalized within the range $[-1, 1]$.

To help the agent assess its performance, this research employed four different reward signals: $+50$ for reaching the goal, $+15$ for disabling an enemy, -20 for being detected by an enemy, and -0.01 for executing actions that do not lead to any of the above states. This reward structure follows a sparse shaping strategy, wherein meaningful signals are deliberately restricted to a small number of outcome-defining transitions, while all remaining timesteps receive only the small step penalty. The aim of this design choice is to reflect the mechanics of stealth games, where consequential events are infrequent.

The relative magnitudes of the reward components were chosen to reflect the relative importance of each outcome: the goal reward of $+50$ establishes task completion as the primary objective; the disable bonus of $+15$ positions enemy disabling as a beneficial but secondary strategy; the detection penalty of -20 is set at a magnitude sufficient to discourage reckless behavior while preserving the viability of risk-taking strategies; and the step penalty of -0.01 is kept deliberately small to promote efficiency without discouraging exploration. Together, this reward structure encourages agents to complete levels as swiftly as possible while avoiding bias toward a specific playstyle, by rewarding both stealthy and aggressive strategies, thereby enabling varied observations of how different algorithms learn.

B. PROBLEM FORMULATION

The proposed stealth game environments are modeled as a Markov Decision Process (MDP), defined by the tuple: (S, A, P, R, γ), where S represents the state space, A the action space, P the transition function, R the reward function, and γ the discount factor.

1) STATE SPACE

The state space $s_t \in S$ is defined as a continuous, normalized vector of environmental features, with all values scaled to the range $[-1, 1]$. The state vector s_t at timestep t is formally defined as

$$s_t = [g_x, g_z, p_x, p_z, w_1^x, w_1^z, \ldots, w_{30}^x, w_{30}^z, e_1^x, e_1^z, \ldots, e_n^x, e_n^z, l_1^x, l_1^z, \ldots, l_{15n}^x, l_{15n}^z]. \quad (11)$$

where (g_x, g_z) are the coordinates of the level goal, (p_x, p_z) are the coordinates of the agent, (w_i^x, w_i^z) for $i = 1, \ldots, 30$ are the endpoints of the agent's ray-casting wall perception vector (explained in the previous section), (e_j^x, e_j^z) for $j = 1, \ldots, n$ are the coordinates of each enemy, (l_k^x, l_k^z) for $k = 1, \ldots, 15n$ are the coordinates of the endpoints of each enemy's line-of-sight, with 15 points defined per enemy, and n denotes the number of enemies present in the level.

An episode begins at a fixed initial state s_0 and proceeds until a terminal state is reached, corresponding to one of three conditions: the agent reaches the level goal, is detected by an enemy, or exhausts the maximum number of allowed steps.

2) ACTION SPACE

The action space A is discrete and consists of ten possible actions, $A = \{a_1, a_2, \ldots, a_{10}\}$. Actions a_1 through a_8 correspond to movement in eight directions, a_9 corresponds to the disable action, and a_{10} corresponds to the no-operation action.

3) TRANSITION FUNCTION

The transition function defines the probability of transitioning to state s' from state s taking action a, expressed as $T(s, a, s') = P(s_{t+1} = s' \mid s_t = s, a_t = a)$. All levels in this work are fully deterministic, meaning that for any given state-action pair (s, a), the resulting next state s' is uniquely determined.

4) REWARD FUNCTION

The reward function defines the scalar reward signal received by the agent at each timestep as a result of taking action a in state s. It is defined as a piecewise function that captures all possible outcomes of agent-environment interaction

$$R(s, a) = \begin{cases} +50 & \text{if } s' \in S_{goal} \\ +15 & \text{if } s' \in S_{disable} \\ -20 & \text{if } s' \in S_{detected} \\ -0.01 & \text{otherwise} \end{cases} \quad (12)$$

where $S_{\text{goal}} \subset S$ is the set of states in which the agent reaches the level goal, $S_{\text{disable}} \subset S$ is the set of states in which the agent successfully disables an enemy, $S_{\text{detected}} \subset S$ is the set of states in which the agent is detected by an enemy, and the final case applies to all remaining non-terminal transitions.

It is worth noting that the reward components are not mutually exclusive across an episode. An agent may accumulate multiple disable rewards ($+15$) before reaching the goal ($+50$), making the maximum achievable return per episode dependent on the number of enemies present in each level. Specifically, the theoretical maximum cumulative reward is $R_{\max} = 50 + 15n - 0.01t$, where n is the number of enemies disabled and t is the number of timesteps elapsed before reaching the goal.

C. TRAINING PROCEDURE AND EVALUATION METRICS

Effective training of deep learning algorithms requires careful hyperparameter selection. In this study, hyperparameters were tuned manually through an iterative process in which parameters were adjusted, performance was evaluated, and refinements were made until no meaningful improvements were observed. Automated optimization methods (e.g., grid search or Bayesian optimization) were not employed due to the large number of algorithms evaluated in this work and the associated computational cost.

A unified tuning strategy was not feasible because the evaluated methods belong to different learning paradigms. Off-policy DRL algorithms rely on gradient-based updates, whereas NE methods use population-based evolutionary mechanisms, sharing few hyperparameters. As a result, each

algorithm family required a tailored tuning approach. Despite these differences, a common baseline training protocol was applied to ensure fair comparison:

1) Each episode on any tested level was limited to a maximum duration of 700 steps. Episodes exceeding this limit were terminated to prevent stagnation while allowing sufficient time for exploration and task completion.
2) Each training session consisted of 300 episodes. For NE algorithms, one episode corresponds to a full population evaluation. Thus, 300 episodes correspond to 300 generations. This configuration provides agents with enough training experience to learn medium- to long-term behaviors.
3) Levels are deterministic, meaning that environmental conditions, including agent starting positions, enemy starting positions, and patrol routes, were held constant across level restarts. This ensures a fair comparison between DRL and NE algorithms, as randomizing the environment between generations could disadvantage the NE methods used in this research.
4) All algorithms were trained from scratch with randomly initialized network parameters.
5) To evaluate performance on a specific level, each algorithm was trained independently in ten separate sessions, which we refer to as a full training session.
6) All algorithms were trained using the same set of ten fixed random seeds, one per training session, ensuring that observed performance differences reflect algorithmic behavior rather than initialization variability.
7) The best-performing hyperparameter combination for each algorithm was determined by comparing the mean episode reward averaged over a full training session, with the configuration yielding the highest mean reward being selected for further evaluation.
8) All algorithms were systematically tuned and evaluated on levels one (a) and two (b) to assess performance across different stealth-related challenges.

This baseline training structure ensures a fair comparison between algorithms.

To gain deeper insights into the learning capabilities of off-policy DRL and NE algorithms in the context of stealth games, it is essential to evaluate their performance in increasingly complex scenarios. While levels one (a) and two (b) each present distinct challenges, they are relatively straightforward to solve for most human players. In contrast, level three (c) provides a significantly more challenging scenario that could pose initial difficulties even for some human players. For this reason, level three (c) was chosen to assess algorithm performance under more demanding conditions, using the best hyperparameter configurations identified during tuning.

Due to the heightened complexity of level three (c), two adjustments were made to the baseline training structure. The maximum number of steps per episode increased from 700 to 1400, and the number of episodes per training session doubled from 300 to 600.

TABLE 1. Best-found DQN hyperparameters.

Name	*Value*
Layer number	3
Neurons per layer	128
Lr & decay	0.0001; 1e-7
Update target network	50
Batch size	32
Gamma (γ)	0.99

TABLE 2. Best-found hyperparameters for the used NE algorithms.

Algorithm	*RS*	*CMA-ES*	*GA*	*NEAT*
Layer number	3	3	3	- -
Neurons per layer	32	256	64	- -
Noise std	0.03	0.015	0.015	0.015
Population size	76	50	76	50
Tournament size	- -	- -	10	- -
Elitism size	- -	- -	3	- -
Lr & decay	- -	0.01; 1e-5	- -	- -
Novelty relevance	0.2	0.5	0.5	0.2

This work evaluates various off-policy DRL algorithms, including DQN, Double DQN, Dueling DQN, N-step DQN, PER DQN, Noisy DQN, Categorical DQN, and Rainbow DQN. Since all these models are built upon DQN, many of their hyperparameters remain the same. For this reason, the tuning process first optimized the hyperparameters for DQN, and the best-performing configurations were then used as the base for fine-tuning the other algorithms in this group.

For NE, the algorithms tested include RS, CMA-ES, GA, and NEAT, along with their NS counterparts: RS-NS, CMA-ES-NS, GA-NS, and NEAT-NS. While these algorithms share many hyperparameters, the optimal values for one algorithm may not necessarily yield the best results for another, as each approach is unique. Therefore, shared hyperparameters were thoroughly tested for every algorithm, in contrast with the approach taken for the DRL group. A detailed view of all the hyperparameter settings used for training is available in our GitHub repository: https://github.com/rorix14/Off-Policy-DRL-and-NE-in-Stealth-Game-Problems

All tests and experiments were conducted over three months using an AMD Ryzen 9 7900X CPU and an NVIDIA GeForce RTX 3070 GPU, and 32G of RAM.

V. RESULTS AND DISCUSSION

A. HYPERPARAMETER TUNING RESULTS

The best-found hyperparameter configuration for the DQN algorithm is presented in Table 1. Alternative combinations of Lr and decay values were also evaluated, yielding comparable performance.

TABLE 3. Best full training sessions results by DRL algorithms on the implemented stealth game levels.

Algorithm	*Level one*	*Level two*	*Level three*
DQN	37.1 ±4.2	41.0 ±4.0	-8.7 ±10.5
Double DQN	41.0 ±1.8	38.9 ±3.7	-7.6 ±16.7
Dueling DQN	37.8 ±13.2	37.3 ±7.9	-7.7 ±14.6
N-step DQN	40.2±4.0	41.4 ±2.4	3.1 ±18.7
PER DQN	40.2 ±4.2	38.3 ±8.9	-0.7 ±18.6
Noisy DQN	37.6 ±9.8	31.0 ±14.2	-10.9 ±13.9
Categorical DQN	38.8 ±5.2	36.1 ±3.9	9.1 ±27.9
Rainbow DQN	**45.0 ±3.7**	**43.3 ±2.5**	**22.2 ±28.4**

TABLE 4. Best full training sessions results by NE algorithms on the implemented stealth game levels.

Algorithm	*Level one*	*Level two*	*Level three*
RS	50.5 ±4.1	21.9 ±20.8	55.5 ±4.3
CMA-ES	48.9 ±0.3	32.8 ±21.1	-13.7 ±0.3
GA	50.1 ±4.6	45.5 ±5.5	**58.1 ±4.6**
NEAT	50.6 ±3.9	32.7 ±16.6	49.5 ± 7.9
RS-NS	49.0 ±2.5	28.9 ±24.2	42.7 ± 15.2
CMA-ES-NS	48.5 ±0.5	35.0 ±7.1	34.5 ± 16.4
GA-NS	49.5 ±0.2	**47.0 ±1.6**	56.9 ± 2.9
NEAT-NS	**53.1 ±2.6**	34.0 ±17.4	57.3 ± 4.4

For DQN-based extensions, the hyperparameters in Table 1 were used as a baseline, with additional tuning performed for extension-specific parameters. The best-performing values were as follows: step size of 3 (N-step DQN), sigma value of 0.05 (Noisy DQN), atom min/max values of -23 and 63, respectively (Categorical DQN), alpha and beta values of 0.2 and 0.6, respectively (PER DQN), and noisy value network layers (Rainbow DQN).

NE algorithms share several common hyperparameters, although each method (with the exception of RS) also includes algorithm-specific parameters [4]. Due to computational constraints, tuning focused primarily on shared hyperparameters, while only a limited set of algorithm-specific parameters were explored. Table 2 summarizes the tuned parameters for each NE method and their corresponding best-performing values. The table also reports the best-performing values for the novelty relevance hyperparameter [49], used when combining NE algorithms with NS.

B. ALGORITHM PERFORMANCE ACROSS LEVELS

The experimental results show that the designed state representation and reward structure successfully allowed the DL algorithms to learn the created stealth game levels, with a few exceptions. While utilizing their best-found hyperparameter combinations, all algorithms successfully learned levels one (a) and two (b), while level three (c) presented more nuanced results.

1) level one (a): All algorithms were able to successfully avoid the enemy and reach the goal. However, only Rainbow DQN, RS, and NEAT-NS consistently maximized rewards by disabling the enemy before reaching the goal.
2) Level two b): All algorithms successfully completed the level, even with a reward structure that did not directly reward agents by their proximity to the goal.
3) Level three (c): This level proved challenging for most DRL algorithms but was relatively straightforward for NE algorithms, as their generation best individuals reach the goal early in training, as seen in Fig. 2. Most DRL algorithms struggled to consistently reach the goal with only N-step DQN, Categorical DQN, and Rainbow DQN averaging positive results. Despite this, all DRL algorithms always had at least one training session where they managed to average positive results by consistently reaching the goal, as can be seen in Table 5. Among NE algorithms, as seen in Fig. 2(b), only CMA-ES was unable to consistently reach the goal, not having a single session with a positive outcome, as seen in Table 5. Further analysis suggests that the current reward structure might not allow CMA-ES to successfully explore the level, as it occasionally finds the goal early in training, but forgoes going there instead preferring to avoid all enemies. We hypothesize that this behavior is derived from the CMA-ES update function (9), which adjusts network parameters based on the achieved fitness of each individual within a generation. Essentially, the CMA-ES update takes into account the experience of all the individuals that get caught trying to reach the goal, resulting in the described behavior. A modified reward structure would probably fix this behavior since the added exploration score to the overall fitness in the CMA-ES-NS algorithm results in behavior that consistently reaches the goal, while avoiding disabling enemies, as can be seen from Fig. 3(b).

C. CONTRASTS BETWEEN DRL AND NE ALGORITHM TUNING

The results from the hyperparameter tuning experiments indicate that the methodology used found combinations that lead to satisfactory performance on levels one (a) and two (b). As previously mentioned, utilizing the best-found hyperparameter combinations for DRL algorithms produced mixed results on level three (c). This suggests that even if a certain hyperparameter combination works well on levels one (a) and two (b), additional tuning might still be needed for more complex scenarios.

The tuning process also showed that the implemented DRL algorithms were sensitive to hyperparameter adjustments. Although small changes to parameters such as batch

TABLE 5. Level three best training sessions mean reward.

DRL method	*Reward*	*NE method*	*Reward*
DQN	19.0	RS	60.8
Double DQN	30.8	CMA-ES	-13.2
Dueling DQN	33.5	GA	61.2
N-step DQN	39.8	NEAT	58.5
PER DQN	30.0	RS-NS	54.9
Noisy DQN	28.6	CMA-ES-NS	43.6
Categorical DQN	50.4	GA-NS	60.2
Rainbow DQN	**51.8**	NEAT-NS	**61.9**

size and target network copy period had little impact, most other parameters would provide considerably different results even with minor modifications. Notably, variations in the ADAM optimizer's Lr and decay values led to vastly different outcomes in performance. In contrast, NE algorithms demonstrated greater robustness to hyperparameter changes, except for CMA-ES and CMA-ES-NS, which were sensitive to changes in the ADAM optimizer's Lr and decay values. These outcomes suggest that DQN-based algorithms may require a more robust hyperparameter tuning approach to fully optimize their performance, whereas NE algorithms appear to achieve strong results even with minimal tuning.

D. PERFORMANCE ANALYSIS

Table 3 and Fig. 2(a) show that, across all the experiments conducted in this research, the DQN variant that consistently provided the best results was Rainbow DQN, which matches with current research [34]. In contrast, Noisy DQN consistently performed the worst, which does not match previous studies [36], mostly displaying risk-averse behaviors limiting exploration, as seen in Fig 3(a). It is worth noting that N-step DQN consistently provided some of the best results from the DRL family, which is somewhat surprising, as it is one of the least explored DQN variants from this list in the research field. This is relevant since the N-Step addition is less complex to implement and can require fewer computational resources than some of the other DQN extensions, including Categorical and Rainbow DQN.

For NE algorithms, Table 4 and Fig. 2(b) show that GA and GA-NS consistently yielded the best results. Further analysis shows that gradient-dependent algorithms (CMA-ES and CMA-ES-NS) tend to underperform compared to algorithms that rely only on natural selection mechanisms.

When comparing both algorithm families, it is clear that NE algorithms exhibit, on average, stronger performance than DRL algorithms across all implemented levels. This advantage likely stems from the characteristics of the level layouts and reward design, both of which require effective exploration for identifying high-quality solutions. In our experiments, NE methods demonstrated more consistent exploration behavior than the DRL approaches, even without the NS addition. We hypothesize that a different reward structure that reduces the reliance on exploration could improve the performance of DRL methods; however, investigating this is left for future work.

It is also important to point out that, when observing these algorithms during training, particularly on levels one (a) and three (c), the highest performing DRL algorithms seem to actually learn the level's dynamics, while the best NE algorithms seem to be mostly brute forcing a solution, by sheer number of solutions tried. These behaviors are aligned with recent works [51], as overall, DRL algorithms seem to solve problems by trying to generalize, while NE algorithms are more prone to overfitting.

To formally validate the observed performance differences, Wilcoxon signed-rank tests were conducted for each algorithm on all independent training session results from level three (c), as this level provided greater complexity and more pronounced performance differences between algorithms. Within the DRL family, the results confirmed that only Rainbow DQN and N-step DQN achieved statistically significant improvements over the baseline DQN algorithm ($p = 0.037$ and $p = 0.027$, respectively), further supporting their status as the strongest DRL performers in this study. Within the NE family, all algorithms showed statistically significant differences from their NS counterparts, with the exception of GA and GA-NS ($p = 0.230$), suggesting that the addition of Novelty Search does not provide a meaningful performance benefit when combined with GA in these environments. Additionally, GA did not show statistically significant differences from RS ($p = 0.064$) and NEAT-NS ($p = 0.769$), suggesting these three algorithms achieve comparable performance levels. When comparing across families, Rainbow DQN did not show a statistically significant difference from CMA-ES-NS ($p = 0.232$), and DQN did not show a statistically significant difference from CMA-ES ($p = 0.131$). All other cross-family comparisons yielded statistically significant differences.

E. SAMPLE EFFICIENCY ANALYSIS

Figure 4 illustrates the mean episode length over training for each algorithm on level three (c), serving as a proxy for sample efficiency. Algorithms that learn to complete episodes in fewer steps consume fewer environment interactions to achieve a given level of performance. Consistent with the reward results presented in Section V-B, algorithms that achieve stronger performance also tend to reduce their episode length more consistently over time, this is particularly evident in the DRL algorithm family, as can be seen from Figure 4(a).

Within the NE family, Figure 4(b) shows a consistent pattern where the non-NS variants of each algorithm demonstrated stronger reductions in mean episode length over training compared to their NS counterparts, with the exception of CMA-ES. This is to be expected, as the NS component combines a novelty score with the fitness score, rewarding the

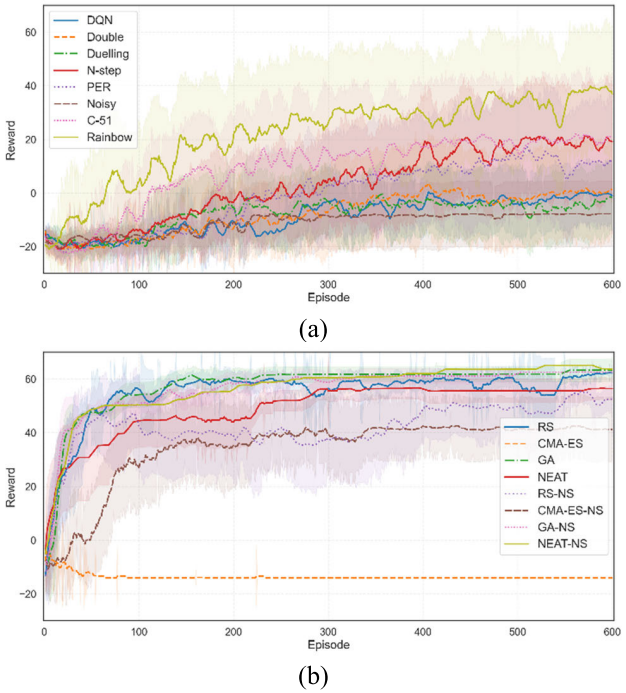

FIGURE 2. (a) Mean performance of each DRL algorithm analyzed in this work over all training sessions on level 3, with a 10-period moving average; (b) Mean performance of the best individuals of each NE algorithm analyzed in this work over all training sessions on level 3, with a 10-period moving average. Shaded regions around each curve represent the 95% confidence interval around the mean.

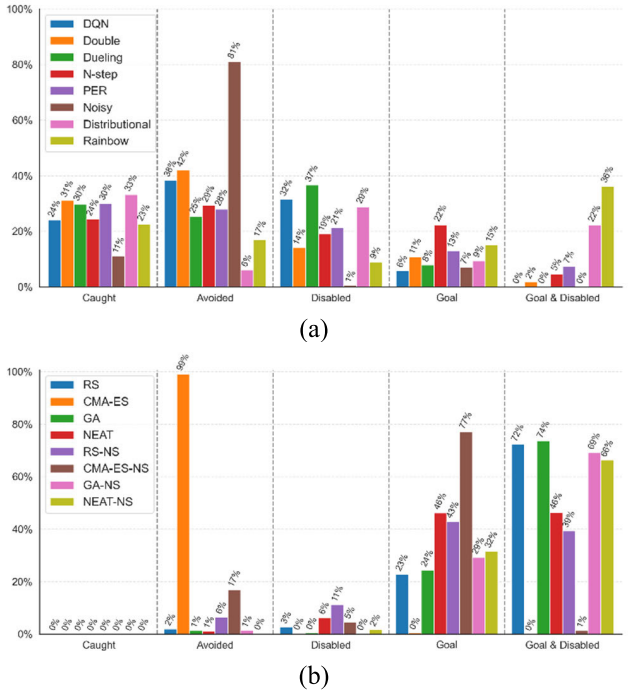

FIGURE 3. DRL (a) and NE (b) algorithms mean end-of-episode behaviors over all training sessions on level 3. "Caught"- the agent was detected by an enemy; "Avoided"- the agent did not reach the goal but was not caught; "Disabled"- the agent disabled at least one enemy but did not reach the goal; "Goal"- the agent reached the goal; "Goal and Disabled"- the agent reached the goal and disabled at least one enemy.

discovery of novel behaviors and thus encouraging broader level exploration rather than pure task completion. It is worth noting, however, that while the non-NS variants proved more sample efficient in terms of episode length reduction, the NS variants tended to discover the level goal earlier in training on average, as evidenced by Figure 2(b). This suggests a meaningful trade-off between exploration breadth and sample efficiency within the NE family, where NS facilitates faster initial discovery of the goal at the cost of slower convergence toward efficient solutions.

A cross-family comparison of Figures 4(a) and 4(b) might initially suggest that NE algorithms are more sample efficient than DRL methods, as their average individual episode lengths are generally lower. However, this interpretation does not account for the population-based nature of NE algorithms. Since every individual in the population interacts independently with the environment at each generation, the true number of environment interactions per episode is obtained by multiplying the mean individual episode length by the population size. When this correction is applied, NE algorithms prove considerably less sample efficient than their DRL counterparts, despite achieving stronger reward performance.

To illustrate this, Rainbow DQN begins consistently reaching the goal around episode 142, by which point it has accumulated approximately 93,162 total environment interactions. In contrast, GA's best individual begins consistently reaching the goal as early as episode 24, which might appear to indicate superior sample efficiency. However, accounting for GA's population size of 76, the total environment interactions accumulated by that point amount to approximately 8,620,224, roughly 93 times more than Rainbow DQN.

It should be noted, however, that in the specific experimental setup of this work, this difference in sample efficiency had a limited practical impact. Environment interactions in the implemented Unity-based stealth levels are not computationally expensive, and NE evaluations are inherently parallelizable since all population individuals can interact with the environment simultaneously. These characteristics reduce the wall-clock cost of the additional interactions, meaning that the large numerical disparity in sample efficiency did not translate into a proportional difference in practical training cost in our setting.

F. LIMITATIONS

While the results presented in this work provide insights into the applicability of off-policy DRL and NE algorithms in stealth game environments, several limitations should be considered when interpreting these findings.

First, although implemented levels were designed to capture core stealth game mechanics and introduce progressively greater challenges, they inherently reflect specific design choices, such as level layout, enemy count, patrol patterns, and reward structure. As previously mentioned, the reward sparsity and exploration-heavy dynamics of the implemented

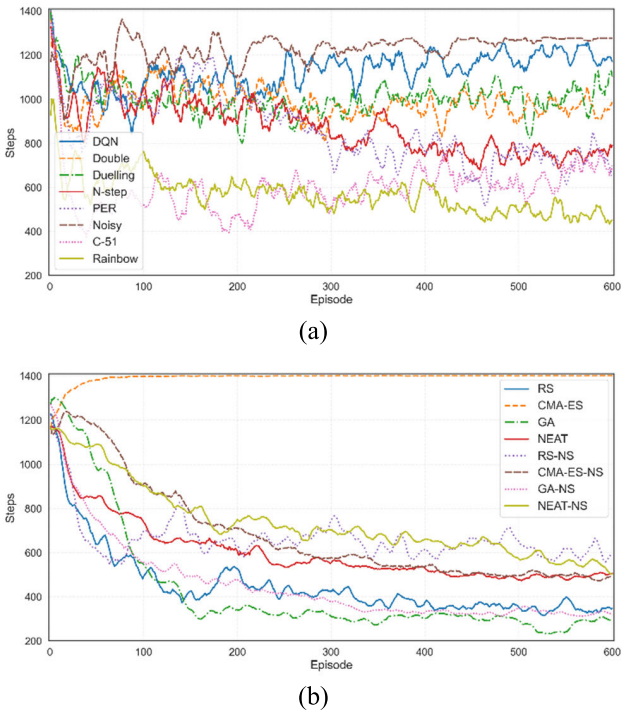

FIGURE 4. (a) Mean episode steps of each DRL algorithm analyzed in this work over all training sessions on level 3, with a 10-period moving average; Mean episode steps for an average individual of each NE algorithm analyzed in this work, over all training sessions on level 3, with a 10-period moving average.

levels may have structurally favored NE algorithms over DRL methods. A different level design or reward structure could potentially shift the relative performance of the two algorithm families.

Second, the environments tested in this work are relatively contained in terms of scalability. It remains an open question whether the findings reported here would scale to larger and more complex stealth scenarios, such as those featuring more enemies, larger maps, richer perceptual inputs, or continuous action spaces. DRL algorithms in particular, which have demonstrated strong performance in high-dimensional environments in other contexts, could potentially scale differently than NE methods under increased environmental complexity, potentially altering the conclusions drawn here.

Finally, this study focuses exclusively on stealth game environments, and the generalizability of these results to other game genres or problem domains cannot be assumed. The specific characteristics of stealth games represent a narrow subset of the broader landscape of game AI challenges, as such, the results obtained here may not transfer directly to environments with different reward dynamics or state representations.

VI. CONCLUSION

This work proposed three different stealth game environments, which introduced a distinctive combination of challenges that set them apart from conventional RL and NE benchmarks. Unlike standard environments, the implemented stealth levels feature sparse and delayed reward signals, requiring agents to sustain coherent behavior over significantly longer episodes. The state representation is inherently dynamic, incorporating geometry-based perception through ray-based environmental sensing and modeling of enemy line-of-sight and occlusion. Furthermore, the strategic trade-off between cautious stealth navigation and riskier but higher-rewarding actions such as disabling enemies, places greater demands on long-term planning. Under these conditions, agents must reason over extended time horizons, operate under continuously changing observations, and discover fundamentally different behavioral strategies.

These characteristics present significant challenges for DRL algorithms, particularly in terms of exploration, while NE algorithms benefited from their population-based exploration to more consistently find high-quality solutions, particularly on the most complex level (c).

Nevertheless, the overall positive results show that the applied state representation and reward structure – detailed in Section IV – could potentially be adapted for similar environments beyond those implemented in this work. While the results were positive, certain exceptions suggest that the developed state representation and reward structure could still be improved. As a potential improvement, we propose adding a small proximity-based reward (granted only when the agent has line-of-sight to the goal) to provide non-deceptive, incremental feedback about progress toward the goal. This addition could reduce dependence on pure exploration to occasionally stumble on the goal, which may benefit DRL in particular, while keeping strategy trade-offs (disabling vs stealth vs speed) meaningful. Nevertheless, the overall positive ability of the used DL algorithms in solving the implemented environments suggests that they are capable of learning various types of game environments, which is consistent with existing research [3], [4], [52].

The hyperparameter tuning experiments conducted in this research identified differences in tuning requirements for each algorithm, particularly when comparing DRL to NE. This process also showed the potential and effectiveness of each implemented DL algorithm in learning the different stealth game environments. However, due to time and computational resource limitations, the hyperparameter tuning methodology was not fully explored, meaning that some DL algorithms could still benefit from further fine-tuning. Nevertheless, the gathered results provide insights into future research on hyperparameter optimization.

The analysis in Section V provides a comparative evaluation of each algorithm within their respective family, DRL and NE, contributing to the ongoing research on the testing and comparison of DL algorithms [4], [10], [34], [39], [51], [53].

A natural extension of this work could involve designing other distinctive stealth challenges and training the implemented DL algorithms in those environments while using a similar state representation and reward structure. Another

promising direction would be to explore on-policy DRL algorithms, which have shown state-of-the-art performance, even surpassing off-policy approaches on some problems [3], [52].

In addition, hybrid approaches that combine DRL and NE algorithms, such as Evolutionary Reinforcement Learning, could be investigated [54]. Such methods have shown promise in leveraging the strengths of both paradigms and are an emergent promising research direction.

Lastly, future work could also explore alternative ANN architectures. Recurrent Neural Networks (RNN) and Long Short-Term Memory (LSTM) networks have proven to have strong performance when employed by some of the algorithms presented in this research [3], [52].

Nevertheless, the findings in this research still make progress toward understanding the potential applicability of DRL and NE in game development, particularly in the context of stealth games.

REFERENCES

[1] C. Gordillo, J. Bergdahl, K. Tollmar, and L. Gisslén, "Improving playtesting coverage via curiosity driven reinforcement learning agents," in *Proc. IEEE Conf. Games (CoG)*, Aug. 2021, pp. 1–8, doi: 10.1109/CoG52621.2021.9619048.

[2] E. Alonso, M. Peter, D. Goumard, and J. Romoff, "Deep reinforcement learning for navigation in AAA video games," in *Proc. 13th Int. Joint Conf. Artif. Intell.*, Aug. 2021, pp. 2133–2139, doi: 10.24963/ijcai.2021/294.

[3] K. Shao, Z. Tang, Y. Zhu, N. Li, and D. Zhao, "A survey of deep reinforcement learning in video games," Dec. 2019, *arXiv:1912.10944*.

[4] S. Risi and J. Togelius, "Neuroevolution in games: State of the art and open challenges," 2014, *arXiv:1410.7326*.

[5] *Gymnasium Documentation*. Farama Foundation. Accessed: Jun. 19, 2024. [Online]. Available: https://gymnasium.farama.org/

[6] K. Cieślak. *How to Design Stealth Games? Try Evidence*. Try_Evidence. Accessed: Dec. 3, 2022. [Online]. Available: https://tryevidence.com/blog/how-to-design-stealth-games/

[7] K. Wong. *Stealth Game Design*. Game Developer. Accessed: Dec. 3, 2022. [Online]. Available: https://www.gamedeveloper.com/design/stealth-game-design

[8] A. Y. Majid, S. Saaybi, V. Francois-Lavet, R. V. Prasad, and C. Verhoeven, "Deep reinforcement learning versus evolution strategies: A comparative survey," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 35, no. 9, pp. 11939–11957, Sep. 2024, doi: 10.1109/TNNLS.2023.3264540.

[9] N. Milan and S. Nolfi, "Qualitative differences between evolutionary strategies and reinforcement learning methods for control of autonomous agents," *Evol. Intell.*, vol. 17, no. 2, pp. 1185–1195, Apr. 2024, doi: 10.1007/s12065-022-00801-3.

[10] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015, doi: 10.1038/nature14236.

[11] D. Ye, "Towards playing full MOBA games with deep reinforcement learning," in *Proc. 34th Int. Conf. Neural Inf. Process. Syst.*, Nov. 2020, pp. 621–632.

[12] G. N. Yannakakis and J. Togelius, *Artificial Intelligence and Games*. Cham, Switzerland: Springer, 2018, doi: 10.1007/978-3-319-63519-4.

[13] J. Togelius, A. Khalifa, S. Earle, M. C. Green, and L. Soros, "Evolutionary machine learning and games," in *Handbook of Evolutionary Machine Learning*. Singapore: Springer, 2024, pp. 715–737, doi: 10.1007/978-981-99-3814-8_25.

[14] E. Beeching, M. Peter, P. Marcotte, J. Debangoye, O. Simonin, J. Romoff, and C. Wolf, "Graph augmented deep reinforcement learning in the GameRLand3D environment," 2021, *arXiv:2112.11731*.

[15] Y. Zheng, X. Xie, T. Su, L. Ma, J. Hao, Z. Meng, Y. Liu, R. Shen, Y. Chen, and C. Fan, "Wuji: Automatic online combat game testing using evolutionary deep reinforcement learning," in *Proc. 34th IEEE/ACM Int. Conf. Automated Softw. Eng. (ASE)*, Nov. 2019, pp. 772–784, doi: 10.1109/ASE.2019.00077.

[16] J. Bergdahl, C. Gordillo, K. Tollmar, and L. Gisslen, "Augmenting automated game testing with deep reinforcement learning," in *Proc. IEEE Conf. Games (CoG)*, Aug. 2020, pp. 600–603, doi: 10.1109/COG47356.2020.9231552.

[17] A. Sestini, L. Gisslén, J. Bergdahl, K. Tollmar, and A. D. Bagdanov, "Automated gameplay testing and validation with curiosity-conditioned proximal trajectories," *IEEE Trans. Games*, vol. 16, no. 1, pp. 113–126, Mar. 2024, doi: 10.1109/TG.2022.3226910.

[18] R. Liang, Z. Ye, Y. Liang, and S. Li, "Deep learning-based player behavior modeling and game interaction system optimization research," in *Proc. 6th Int. Conf. Comput. Eng. Appl. (ICCEA)*, Apr. 2025, pp. 1–4, doi: 10.1109/ICCEA65460.2025.11103285.

[19] D. B. Or, M. Kolomenkin, and G. Shabat, "DL-DDA–deep learning based dynamic difficulty adjustment with UX and gameplay constraints," in *Proc. IEEE Conf. Games (CoG)*, Aug. 2021, pp. 1–7, doi: 10.1109/CoG52621.2021.9619162.

[20] R. R. Torrado, P. Bontrager, J. Togelius, J. Liu, and D. Perez-Liebana, "Deep reinforcement learning for general video game AI," in *Proc. IEEE Conf. Comput. Intell. Games (CIG)*, Aug. 2018, pp. 1–8, doi: 10.1109/CIG.2018.8490422.

[21] M. Kempka, M. Wydmuch, G. Runc, J. Toczek, and W. Jaskowski, "ViZDoom: A doom-based AI research platform for visual reinforcement learning," in *Proc. IEEE Conf. Comput. Intell. Games (CIG)*, Sep. 2016, pp. 1–8, doi: 10.1109/CIG.2016.7860433.

[22] E. Beeching, J. Debangoye, O. Simonin, and C. Wolf, "Godot reinforcement learning agents," 2021, *arXiv:2112.03636*.

[23] G. Macaluso, A. Sestini, and A. D. Bagdanov, "A benchmark environment for offline reinforcement learning in racing games," in *Proc. IEEE Conf. Games (CoG)*, Aug. 2024, pp. 1–4, doi: 10.1109/cog60054.2024.10645657.

[24] A. Juliani, "Unity: A general platform for intelligent agents," May 2020, *arXiv:1809.02627*.

[25] S. Huang, "CleanRL: High-quality single-file implementations of deep reinforcement learning algorithms," *J. Mach. Learn. Res.*, vol. 23, no. 274, pp. 1–18, Jul. 2022. Accessed: Mar. 9, 2026. [Online]. Available: http://jmlr.org/papers/v23/21-1342.html

[26] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-Baselines3: Reliable reinforcement learning implementations," *J. Mach. Learn. Res.*, vol. 22, no. 268, pp. 1–8, Nov. 2021. [Online]. Available: https://github.com/DLR-RM/stable-baselines3

[27] E. Liang, "RLlib: Abstractions for distributed reinforcement learning," in *Proc. 35th Int. Conf. Mach. Learn., Proc. Mach. Learn. Res.*, 2018, pp. 3053–3062. [Online]. Available: http://rllib.io

[28] P. Gutiérrez-Sánchez, M. A. Gómez-Martín, P. A. González-Calero, and P. P. Gómez-Martín, "Reinforcement learning methods to evaluate the impact of AI changes in game design," in *Proc. AAAI Conf. Artif. Intell. Interact. Digit. Entertainment*, vol. 17, no. 1, Oct. 2021, pp. 10–17, doi: 10.1609/aiide.v17i1.18885.

[29] M. R. F. de Mendonça, H. S. Bernardino, and R. F. Neto, *Evolution of Reward Functions for Reinforcement Learning applied to Stealth Games*. Minas Gerais, Brazil: Universidade Federal de Juiz de Fora, 2016, doi: 10.13140/RG.2.2.23533.59367.

[30] P. Ladosz, L. Weng, M. Kim, and H. Oh, "Exploration in deep reinforcement learning: A survey," *Inf. Fusion*, vol. 85, pp. 1–22, Sep. 2022, doi: 10.1016/j.inffus.2022.03.003.

[31] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-learning," in *Proc. AAAI Conf. Artif. Intell.*, vol. 30, no. 1, pp. 2094–2100, Mar. 2016, doi: 10.1609/aaai.v30i1.10295.

[32] Z. Wang, T. Schaul, M. Hessel, H. van Hasselt, M. Lanctot, and N. de Freita, "Dueling network architectures for deep reinforcement learning," 2015, *arXiv:1511.06581*.

[33] R. S. Sutton, "Learning to predict by the methods of temporal differences," *Mach. Learn.*, vol. 3, no. 1, pp. 9–44, Aug. 1988, doi: 10.1007/bf00115009.

[34] M. Hessel, "Rainbow: Combining improvements in deep reinforcement learning," in *Proc. 32nd AAAI Conf. Artif. Intell.*, 2018, pp. 3215–3222. Accessed: Jan. 26, 2023. [Online]. Available: www.aaai.org

[35] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," 2015, *arXiv:1511.05952*.

[36] M. Fortunato, "Noisy networks for exploration," 2017, *arXiv:1706.10295*.

[37] M. G. Bellemare, W. Dabney, and R. Munos, "A distributional perspective on reinforcement learning," in *Proc. 34th Int. Conf. Mach. Learn.*, 2017, pp. 449–458. Accessed: Oct. 5, 2023. [Online]. Available: https://proceedings.mlr.press/v70/bellemare17a.html
[38] T. Salimans, J. Ho, X. Chen, S. Sidor, and I. Sutskever, "Evolution strategies as a scalable alternative to reinforcement learning," Sep. 2017, *arXiv:1703.03864*.
[39] F. P. Such, V. Madhavan, E. Conti, J. Lehman, K. O. Stanley, and J. Clune, "Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning," Apr. 2018, *arXiv:1712.06567*.
[40] S. M. Elsayed, R. A. Sarker, and D. L. Essam, "A comparative study of different variants of genetic algorithms for constrained optimization," in *Proc. Asia–Pacific Conf. Simulated Evol. Learn.*, 2010, pp. 177–186.
[41] A. Jenkins, V. Gupta, A. Myrick, and M. Lenoir, "Variations of genetic algorithms," 2019, *arXiv:1911.00490*.
[42] S. Katoch, S. S. Chauhan, and V. Kumar, "A review on genetic algorithm: Past, present, and future," *Multimedia Tools Appl.*, vol. 80, no. 5, pp. 8091–8126, Feb. 2021, doi: 10.1007/s11042-020-10139-6.
[43] S. M. Libelli and P. Alba, "Adaptive mutation in genetic algorithms," *Soft Comput.*, vol. 4, no. 2, pp. 76–80, Jul. 2000, doi: 10.1007/s005000000042.
[44] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evol. Comput.*, vol. 10, no. 2, pp. 99–127, Jun. 2002, doi: 10.1162/106365602320169811.
[45] K. O. Stanley, J. Clune, J. Lehman, and R. Miikkulainen, "Designing neural networks through neuroevolution," *Nature Mach. Intell.*, vol. 1, no. 1, pp. 24–35, Jan. 2019, doi: 10.1038/s42256-018-0006-z.
[46] J. Lehman and K. O. Stanley, "Abandoning objectives: Evolution through the search for novelty alone," *Evol. Comput.*, vol. 19, no. 2, pp. 189–223, Jun. 2011.
[47] G. Cuccu and F. Gomez, "When novelty is not enough," in *Proc. Eur. Conf. Appl. Evol. Comput.*, 2011, pp. 234–243.
[48] J. Lehman and K. O. Stanley, "Revising the evolutionary computation abstraction: Minimal criteria novelty search," in *Proc. 12th Annu. Conf. Genetic Evol. Comput.*, Jul. 2010, pp. 103–110.
[49] E. Segal and M. Sipper, "Adaptive combination of a genetic algorithm and novelty search for deep neuroevolution," in *Proc. 14th Int. Joint Conf. Comput. Intell.*, Sep. 2022, pp. 143–150.
[50] E. Conti, V. Madhavan, F. P. Such, J. Lehman, K. O. Stanley, and J. Clune, "Improving exploration in evolution strategies for deep reinforcement learning via a population of novelty-seeking agents," in *Proc. 32nd Int. Conf. Neural Inf. Process. Syst.*, 2018, pp. 5032–5043.
[51] O. Halaš, F. Zúbek, and I. Sekaj, "Comparison of deep reinforcement methods and neuroevolution for bipedal Walker learning," in *Proc. Cybern. Informat. (KI)*, Feb. 2025, pp. 1–6, doi: 10.1109/ki64036.2025.10916475.
[52] N. Justesen, P. Bontrager, J. Togelius, and S. Risi, "Deep learning for video game playing," *IEEE Trans. Games*, vol. 12, no. 1, pp. 1–20, Mar. 2020.
[53] M. Hausknecht, J. Lehman, R. Miikkulainen, and P. Stone, "A neuroevolution approach to general Atari game playing," *IEEE Trans. Comput. Intell. AI Games*, vol. 6, no. 4, pp. 355–366, Dec. 2014, doi: 10.1109/TCIAIG.2013.2294713.
[54] P. Li, J. Hao, H. Tang, X. Fu, Y. Zheng, and K. Tang, "Bridging evolutionary algorithms and reinforcement learning: A comprehensive survey on hybrid algorithms," *IEEE Trans. Evol. Comput.*, vol. 29, no. 5, pp. 1707–1728, Oct. 2025, doi: 10.1109/TEVC.2024.3443913.

PEDRO F. SILVA received the B.S. degree in game development and the M.S. degree in artificial intelligence from IADE University, Lisbon, Portugal, in 2021 and 2024, respectively. He is currently pursuing the Ph.D. degree in informatics engineering with the University of Coimbra, Coimbra, Portugal.

He is currently a Junior Researcher with the Centre for Informatics and Systems of the University of Coimbra (CISUC), Coimbra. He has experience in full-stack web, mobile, the Internet of Things applications, deep learning libraries, and video game development. He also possesses research and development skills in various artificial intelligence techniques, including classical algorithms, game AI, and deep learning methods, such as large language models, deep reinforcement learning, and evolutionary algorithms.

JACINTO ESTIMA received the Ph.D. degree (cum laude) in information management from the NOVA University of Lisbon, Lisbon, Portugal, in 2015. He is currently an Assistant Professor with the Department of Informatics Engineering, University of Coimbra, Coimbra, Portugal, and a Senior Researcher with the Information Systems Group, CISUC. Previously, he was the Program Officer of the International Renewable Energy Agency (IRENA), where he led the development and management of several information systems initiatives. He has been involved in multiple research projects addressing public participation in environmental monitoring, conservation, optimization, and automation supporting green transitions. His work integrates expertise in volunteered geographic information and machine learning to tackle interdisciplinary challenges at the intersection of technology, data, and environmental sustainability. His research interests include information systems, intelligent systems, geographic information science, and citizen science.

INÊS NOBRE received the B.S. degree in sports sciences and the M.S. degree in exercise and health from the Faculty of Human Kinetics, University of Lisbon, Lisbon, Portugal, in 2013 and 2016, respectively, where she is currently pursuing the Ph.D. degree in physical activity and health. She is currently an Exercise Physiologist and a Junior Researcher with the Faculty of Human Kinetics, University of Lisbon. Her work experience is centered on developing, monitoring, and implementing tailored exercise programs for individuals with chronic conditions, with a specific focus on breast cancer survivors. She has collaborated on various scientific research projects with national and European partners, focusing on the development of effective and safe physical activity interventions. Her research interests include physical activity interventions for chronic conditions, patient well-being, and oncology exercise physiology.

EDIRLEI SOARES DE LIMA received the B.Sc. degree in computer science from Contestado University (UnC), Caçador, Brazil, in 2008, the M.Sc. degree in computer science from the Federal University of Santa Maria (UFSM), Santa Maria, Brazil, in 2010, and the Ph.D. degree in computer science from the Pontifical Catholic University of Rio de Janeiro (PUC-Rio), Rio de Janeiro, Brazil, in 2014. He is currently a Lecturer and a Researcher in artificial intelligence and games at Breda University of Applied Sciences, Breda, The Netherlands. Previously, he was an Assistant Professor, a Program Coordinator, and a Department Coordinator with IADE, Universidade Europeia, Lisbon, Portugal. He also served as an Adjunct Professor with Rio de Janeiro State University (UERJ/IPRJ) and held lecturer positions with the Federal University of the State of Rio de Janeiro (UNIRIO) and the Pontifical Catholic University of Rio de Janeiro (PUC-Rio). His work explores how artificial intelligence can generate, adapt, and structure interactive narratives, combining techniques from automated planning, machine learning, and evolutionary algorithms. His research interests include interactive narratives, procedural content creation, and adaptive storytelling systems. He was a recipient of the best paper awards at the IFIP International Conference on Entertainment Computing (ICEC) in 2015 and 2024, and the multiple best paper awards at Brazilian Symposium on Games and Digital Entertainment (SBGames) from 2016 to 2023. He was also awarded the Indie Game Development Festival Prize at SBGames 2010 and received the honorable mentions for innovation and interactivity from the International Telecommunication Union (ITU) in 2011 and 2012.