AI FOR SPACE INVADERS USING REINFORCEMENT LEARNING

B Sai Abhishek, Balam Ruchith Balaji, Chillakuru Hari
Department of Computer Science and Engineering, Amrita School of Computing,
Bengaluru, Amrita Vishwa Vidyapeetham, India.
BL.EN.U4AIE21015@bl.students.amrita.edu, BL.EN.U4AIE21017@bl.students.amrita.edu,
BL.EN.U4AIE21038@bl.students.amrita.edu,

Abstract— This paper presents a comparative analysis of three reinforcement learning algorithms—Deep Q-Network (DQN), Proximal Policy Optimization (PPO), and a Random policy—applied to the classic arcade game Space Invaders. The study evaluates the performance of each algorithm in autonomously learning and playing the game using the environment created by us. DQN, which approximates Q-values through deep learning, shows rapid initial learning but exhibits performance variability due to its off-policy nature. PPO, an on-policy algorithm with stable policy updates, demonstrates more consistent and stable performance. The Random policy, lacking any learning strategy, performs poorly and serves as a baseline for comparison. Results highlight the strengths and weaknesses of value-based (DQN) versus policy-based (PPO) approaches, providing valuable insights into their application in dynamic environments.

Keywords—Reinforcement Learning, DQN, PPO, Random, Space Invaders.

I. INTRODUCTION

The resurgence of interest in artificial intelligence (AI) and machine learning has led to significant advancements in the field of reinforcement learning (RL), where agents learn to make decisions by interacting with their environment. Among the diverse applications of RL, video games serve as an excellent testbed due to their complex and dynamic nature. One such game is Space Invaders, a classic arcade game that offers a challenging environment for testing the capabilities of various RL algorithms..

This paper focuses on evaluating and comparing three distinct RL approaches: Deep Q-Network (DQN), Proximal Policy Optimization (PPO), and a Random policy. DQN, a value-based method, employs deep learning to approximate the optimal action-value function, enabling the agent to learn effective strategies through experience replay and Q-learning updates. PPO, a policy gradient method, optimizes the policy directly by employing clipped surrogate objectives, which ensure stable and efficient learning. The Random policy, which selects actions arbitrarily without any learned strategy, serves as a control to gauge the effectiveness of the RL algorithms.

The primary goal of this study is to analyse the performance of these algorithms in the Space Invaders environment, comparing their learning efficiency, stability, and overall effectiveness. By examining these methods in a controlled setting, we aim to shed light on the practical advantages and limitations of value-based versus policy-based reinforcement learning approaches. This comparison not only provides insights into the specific algorithms but also contributes to the broader understanding of RL techniques in complex, real-time decision-making scenarios.

II. LITERATURE REVIEW

Mayank Shrivastava Et al [1]. This thesis proposes a new algorithm for imitation learning for AI systems, specifically working in the Atari 2600 Space Invaders environment. The algorithm is modified to work in complex environments and is compared to GAIL, a state-of-the-art deep imitation learning algorithm. The approach divides human imitation into two subproblems: creating an agent that plays well and learning a "corrective" function to play in a human manner. The hybrid approach is fast to train and can be adjusted to match human behavior or performance levels.

Ibrahim Abu-Farha Et al [2]. This project aims to demonstrate how reinforcement learning algorithms can play a video game like Space Invader with human-like performance, comparing Deep Q-learning and Deep Deterministic Policy Gradient (DDPQ) algorithms. The OpenAI Gym Atari emulator is used to run the game, providing valuable information about the environment. The RL agent analyzes the environment's values to optimize its reward in the game. Success will be determined by the average score of the agents compared to human players and random agents.

D. Vinoth Et al [3]. Reinforcement learning (RL) studies how agents achieve rewards in environments over time. Machine RL has advanced beyond human expertise but requires extensive experience. A new approach, Theory-Based Reinforcement Learning, uses human-like intuitive theories to explore and model environments. This approach is instantiated in a video game playing agent called EMPA, which performs Bayesian inference to learn probabilistic generative models and runs internal simulations. EMPA matches human learning efficiency on 90 challenging video games and captures fine-grained structure in people's exploration trajectories and learning dynamics.

. Avinash Kumar Et al [4]. The Arcade Learning Environment (ALE) is a popular platform for evaluating reinforcement learning agents, but its representation learning problem adds computational expense. MinAtar, a set of environments that capture the mechanics of specific Atari games, simplifies this complexity, focusing on behavioral challenges, improving reproducibility and allowing researchers to investigate similar challenges.

Samer Muthana Sarsam Et al [5]. Reinforcement learning is an unsupervised machine learning method in Artificial Intelligence that simulates human thinking. It's useful in game AI research for giving nonplayer-characters (NPCs) more human-like qualities. This paper aims to build a Tank-battle computer game using reinforcement learning for NPCs, enhancing interactions with intelligent NPCs to make the game more interesting.

Aparna K Ajayan Et al [6]. Deep reinforcement learning has shown impressive accuracies in games compared to traditional methods. This paper proposes two efficient neural network architectures: Light-Q-Network (LQN) and Binary-Q-Network (BQN). LQN uses depth-wise separable CNNs for memory and computation saving, while BQN uses binary weights for shortening training time and reducing memory consumption. The approach is evaluated on Atari 2600 domain and StarCraft II mini-games, showing efficiency and no impact on agent learning game strategy.

K. Sree Lakshmi Et al [7]. The development of Non-Player Characters (NPCs) is a complex task, traditionally done using techniques like NavMesh. However, reinforcement learning is more efficient and flexible. A modeling environment has been developed to integrate with game development tools, allowing direct specification of reward functions and NPC agent components with maximum code reuse and automatic code generation.

Venkatesh B Et al [8]. Machine Learning (ML) techniques can enhance game experiences by allowing adaptations to avoid easy or hard patterns. However, some game characteristics are difficult to model and require a wide variety of observations. This work proposes a novel approach using the Hidden Markov Model (HMM) for game balancing, learning patterns based on a strong co-relational between an observation and an unknown variable. The approach learns player patterns through temporal frame observation and adapts the game according to the current player, making it harder for a certain period. A Space Invaders clone validates the approach, showing 54% of participants enjoyed the game with ML.

R Prasanna Kumar Et al [9]. This study explores the role of domain-specific removal of actions and discretization of continuous actions in reinforcement learning (RL) in video game environments. The results suggest that these modifications are crucial for successful learning and aim to simplify the use of RL in new environments.

Santosh Kumar Bharti Et al [10]. This paper presents a lightweight method for generating explanations for deep Reinforcement Learning (RL) agents, overcoming the lack of explainability in games. The method involves transforming pixel-based input into interpretable, percept-like representations and training a surrogate model to replicate the behavior of the target agent. Experiments show that this method accurately approximates the underlying decision-making of a target agent on a suite of Atari games.

Research Gaps:

Traditional implementations of Space Invaders restrict the spaceship's movement to only left and right directions. This simplified action space does not fully utilize the potential complexity and strategic depth of the game.

Our study addresses this gap by extending the action space to include vertical movement, allowing the agent to move both up and down in addition to left and right. This introduces a new dimension to the gameplay, increasing the challenge for the learning algorithms.

III. Methods and Materials

METHODS:
Algorithm Implementation:

Deep Q-Network (DQN): Implement DQN using a deep convolutional neural network to approximate the Q-value function. The network will process the raw pixel inputs from the game and output Q-values for each possible action.

Proximal Policy Optimization (PPO): Implement PPO using a policy gradient approach. This will involve constructing a policy network that directly maps states to action probabilities and optimizing it using clipped surrogate objectives for stable training.

Random Policy: Implement a baseline agent that selects actions randomly, providing a control for comparing the performance of learned strategies.

Environment Setup:

Custom Environment Creation: Manually create a customized version of the Space Invaders game environment. This involves modifying the traditional game to allow the spaceship to move both vertically and horizontally (left-right-up-down).

Use appropriate game development tools or frameworks (such as Pygame or Unity) to create and integrate the new environment with the reinforcement learning algorithms.

Ensure the custom environment is compatible with standard RL libraries and interfaces for seamless integration with DQN and PPO implementations.

Training and Evaluation:

Train each algorithm (DQN, PPO, Random) over a significant number of episodes to ensure adequate learning. Evaluate the performance based on key metrics such as average score, stability of learning (variance in performance), and convergence time. Perform statistical analysis to compare the performance across the three algorithms.

MATERIALS:
Software and Libraries:

TensorFlow or PyTorch: For building and training neural networks.

Stable Baselines: For readily available implementations of PPO and other RL algorithms.

Game Development Frameworks: Such as Pygame or Unity for creating and modifying the Space Invaders environment.
Hardware:

Computational Resources: Access to high-performance computing resources such as GPUs for efficient training of deep learning models.

Development Tools:

Integrated Development Environment (IDE): Such as PyCharm or VSCode for coding and debugging.

Data Collection and Analysis:

Logging Tools: For recording performance metrics during training and evaluation.
Visualization Tools: Such as Matplotlib or TensorBoard for visualizing learning curves and other relevant statistics.

IV. METHODOLOGY

Reward Function:

Action	Reward
Every Frame Change	-0.001
Shooting	-0.001
Killing Enemy	1
Losing Game	-2

Reason to give reward that number only:

In this game scenario, the main objective is for the agent to survive in the environment by avoiding being stuck in corners. To encourage the agent to actively engage with the environment and not just stay put, the frame change reward is decreased by -0.001. This means that the agent receives a slight penalty for staying idle, nudging it to either shoot bullets or move around.

When the agent shoots a bullet, it receives a reward of -0.001. This negative reward discourages the agent from indiscriminately shooting, as it would otherwise tend to shoot excessively to maximize rewards. Instead, it incentivizes the agent to be strategic and efficient in its use of ammunition.

However, when the agent successfully kills an enemy, it receives a substantial reward of 1. This high positive reward motivates the agent to actively seek out and eliminate enemies whenever possible, as it contributes significantly to its overall score and success in the game.

On the other hand, if the agent loses the game, it incurs a penalty of -2. This substantial negative reward encourages the agent to prioritize survival above all else, as losing the game results in a significant decrease in its overall performance. By carefully balancing these rewards and penalties, the game incentivizes the agent to navigate the environment strategically, engaging with enemies while also prioritizing its own survival.

DQN – Deep Q-Network:

The Deep Q-Network (DQN) is a reinforcement learning algorithm that learns to make decisions by directly processing raw pixel inputs from the game environment. During training, the agent's experiences, comprising state, action, reward, and next state, are stored in a replay buffer, facilitating experience replay. This replay buffer allows the agent to learn from past experiences by randomly sampling mini batches during training.

The Q-network is updated through Q-learning, minimizing the mean squared error between the predicted Q-values and target Q-values derived from the Bellman equation. This update mechanism reinforces the network's ability to approximate optimal action values. To balance exploration and exploitation, an epsilon-greedy policy is implemented, where epsilon gradually decreases over time. This policy enables the agent to explore new actions with a decreasing probability while exploiting its current knowledge.

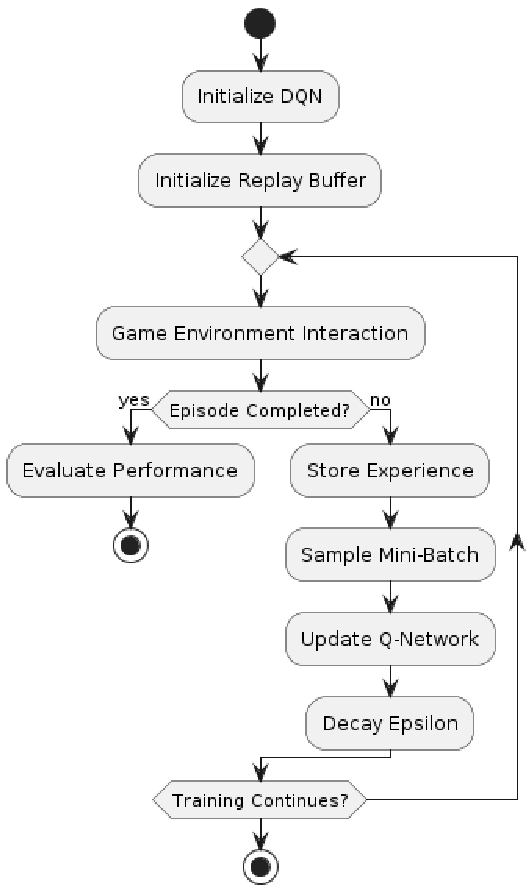

Fig-1 Flowchart: Training Process of a Deep Q-Network (DQN) for Reinforcement Learning.

Hyperparameters such as learning rate, discount factor (γ), batch size, and epsilon decay rate are crucial tuning components of the DQN. These hyperparameters significantly influence the network's learning dynamics and convergence speed. Techniques such as grid search or Bayesian optimization can be employed to systematically tune these hyperparameters for optimal performance. Evaluation of the agent's performance is conducted periodically by measuring its average score over a fixed number of episodes. This evaluation provides insights into the agent's progress and helps assess its effectiveness in the game environment. By continuously refining the network architecture and fine-tuning hyperparameters, the DQN strives to achieve superior performance through iterative learning and optimization.

PPO – Proximal Policy Optimization:

Proximal Policy Optimization (PPO) employs a network architecture comprising a policy network, responsible for outputting action probabilities, and a value network, tasked with estimating state values. The choice between sharing or separating network architectures for these components is determined through experimentation, allowing for optimization based on performance metrics. During training, the policy network is updated using the policy gradient method, aiming to maximize the expected return. To ensure stable updates, a clipped surrogate objective function is employed, which constrains policy updates based on a clipping parameter and advantage estimate, derived from Generalized Advantage Estimation (GAE) for improved variance reduction.

Hyperparameters such as learning rate, clipping parameter (ϵ), discount factor (γ), and GAE parameter (λ) play pivotal roles in PPO's performance. These parameters are fine-tuned using techniques like grid search or Bayesian optimization to optimize the algorithm's learning dynamics and convergence properties.

Evaluation of the agent's performance occurs periodically by measuring its average score over a fixed number of episodes. This evaluation provides insights into the agent's progress and effectiveness in navigating the game environment, enabling iterative refinement and optimization of the PPO algorithm.

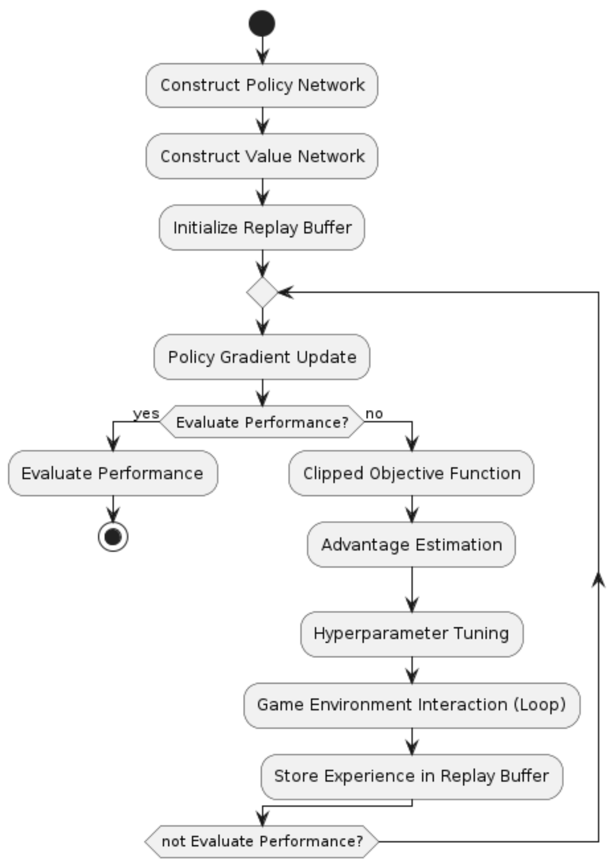

Fig-2 Flowchart: Training Process of a Proximal Policy Optimization (PPO) for Reinforcement Learning

Random Policy:

Random policy is defined as one that selects actions uniformly at random from the available action space, which typically includes actions such as moving left, right, up, down, or shooting, with each action having an equal probability of being chosen. This policy lacks any strategic decision-making and merely serves as a baseline for comparison against more sophisticated learned policies.

To establish a baseline for evaluation, the random policy is implemented to provide a point of reference against which the performance of more advanced reinforcement learning algorithms, such as Deep Q-Networks (DQN) and Proximal Policy Optimization (PPO), can be measured. By measuring the average score achieved over a fixed number of episodes, the effectiveness of the random policy in navigating the game environment can be assessed.

The evaluation of the random policy serves to highlight the improvements achieved by DQN and PPO algorithms. By comparing the performance of these algorithms against the random policy baseline, the efficacy of their learned policies can be quantified, demonstrating the extent to which they surpass random decision-making and effectively learn optimal strategies for achieving higher scores in the given Environment.

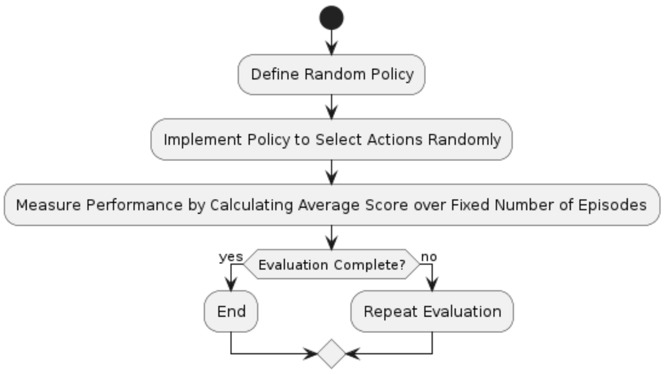
Define Random Policy
Implement Policy to Select Actions Randomly
Measure Performance by Calculating Average Score over Fixed Number of Episodes
yes
Evaluation Complete?
no
End
Repeat Evaluation

Fig-3 Flowchart: Random Policy

RL Formulation:

Agent: The agent is the player-controlled character whose goal is to maximize its score by shooting down invading aliens while avoiding their attacks.

Environment: The environment represents the game of Space Invaders itself, including the game screen, the aliens, the player's spaceship, and any other elements present in the game

State Space: The state space consists of all the information that the agent can observe at a given time step. In Space Invaders, this includes:

- The current frame of the game screen, representing the visual input to the agent.
- The position and speed of the player's spaceship.
- The positions, speed of the aliens on the screen.
- The positions and speeds of any bullets fired by the player.

Action Space: The action space defines the set of actions that the agent can take at each time step. In Space Invaders, the agent can typically perform the following actions:

- Move left.
- Move right.
- Move Up.
- Move Down.
- Shoot (fire a bullet).
- Do nothing (stay still).

Reward function is discussed at the start of the Methodology.

V. Results

The 2 algorithms DQN and PPO are trained 10 million episodes. Literally it took 4 days to complete all the training of the models. Here is the table which shows in detail about the training.

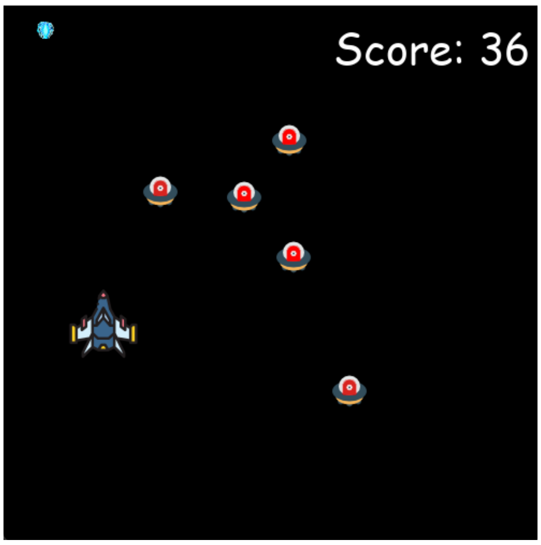

Fig-4 Training the agent.

Algorithm	Learning Rate	Mean reward	Exploration Fraction
DQN	1e-4	15.85	0.1
PPO	1e-4	20.831	0.025

The testing of the trained model, Since the random policy has no training, it can be directly tested.

Algorithm	Episode Number	Average Score	Highest Score
DQN	200	19.85	154
PPO	200	24.831	175
Random	200	7.454	34

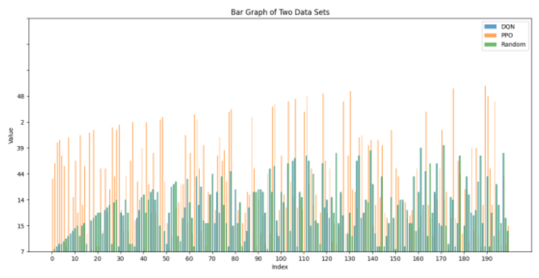

Fig-5 Score Distribution up to 200 Episodes.

VI. Conclusion

Based on the results of testing the Space Invaders game agent across 200 episodes using three different approaches—PPO (Proximal Policy Optimization), DQN (Deep Q-Network), and a random policy—the conclusions indicate a clear performance hierarchy. PPO achieved the highest average reward of 24.831 and the highest single-episode reward of 175, demonstrating its superior effectiveness in learning optimal strategies for the game. In comparison, DQN obtained an average reward of 19.85 and a highest single-episode reward of 154, indicating it is capable of learning and improving game strategies but not to the same extent as PPO. The random policy had the lowest performance, with an average reward of 7.454 and a highest single-episode reward of 34, serving as a baseline for comparison. These results highlight PPO's ability to consistently achieve better outcomes and peak performance in the Space Invaders game compared to DQN and a random policy.

References

[1] McKenzie, M., Loxley, P., Billingsley, W. and Wong, S., 2017. Competitive reinforcement learning in atari games. In AI 2017: Advances in Artificial Intelligence: 30th Australasian Joint Conference, Melbourne, VIC, Australia, August 19–20, 2017, Proceedings 30 (pp. 14-26). Springer International Publishing.

[2] Martinez-Nieves, C.L., Defeating the Invaders with Deep Reinforcement Learning.

[3] Tsividis, P.A., Loula, J., Burga, J., Foss, N., Campero, A., Pouncy, T., Gershman, S.J. and Tenenbaum, J.B., 2021. Human-level reinforcement learning through theory-based modeling, exploration, and planning. arXiv preprint arXiv:2107.12544.

[4] Young, K. and Tian, T., 2019. Minatar: An atari-inspired testbed for thorough and reproducible reinforcement learning experiments. arXiv preprint arXiv:1903.03176.

[5] "Applying Reinforcement Learning for the AI in a Tank-Battle Game." (2010).

[6] Li, Y., Fang, Y. and Akhtar, Z., 2020. Accelerating deep reinforcement learning model for game strategy. Neurocomputing, 408, pp.157-168.

[7] Gomes, G., Vidal, C.A., Cavalcante-Neto, J.B. and Nogueira, Y.L., 2022. A modeling environment for reinforcement learning in games. Entertainment Computing, 43, p.100516.

[8] Zamith, M., da Silva Junior, J.R., Clua, E.W. and Joselli, M., 2020, November. Applying hidden Markov model for dynamic game balancing. In 2020 19th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames) (pp. 38-46). IEEE.

[9] Kanervisto, A., Scheller, C. and Hautamäki, V., 2020, August. Action space shaping in deep reinforcement learning. In 2020 IEEE conference on games (CoG) (pp. 479-486). IEEE.

[10] Sieusahai, A. and Guzdial, M., 2021, October. Explaining deep reinforcement learning agents in the atari domain through a surrogate model. In Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (Vol. 17, No. 1, pp. 82-90).