

Snake Game Variations: a Reinforcement Learning Project

AIE 322: Advanced Machine Learning

Fall semester 2024

Project Team

Student ID	Student Name
221101013	Youssef Gamal
221101140	Nour Maher
221100979	Yassin Walid
221100128	Ahmed Hassany
221100049	Abdullah Hossam

Under supervision of Professor: Manar Elshazly

Index

- 1. Introduction**
- 2. Background and Related Work**
- 3. Methodology**
- 4. Implementation Details**
- 5. Game Versions and Features**
- 6. Results and Visualizations**
- 7. Challenges and Solutions**
- 8. Conclusion**
- 9. Future Work**

Abstract

This report presents an in-depth study on the application of reinforcement learning (RL) to create intelligent agents capable of navigating and excelling in multiple variations of the Snake game. Using the Deep Q-Learning (DQL) algorithm, the project introduces four progressively challenging game variants to test agent adaptability, robustness, and decision-making skills. The findings reveal significant improvements in agent performance, demonstrating the versatility of RL in dynamic and complex environments. This work underscores the potential for RL algorithms to generalize and tackle real-world challenges beyond controlled simulations.

Introduction

The Snake game, with its simplistic yet challenging mechanics, has long been utilized as a testbed for artificial intelligence research. This project leverages reinforcement learning (RL), particularly Deep Q-Learning (DQL), to train agents capable of adapting to dynamic and complex game environments. Unlike traditional supervised learning, RL enables agents to derive optimal strategies through continuous interaction with their environment, learning from rewards and penalties.

By introducing dynamic elements—including changing obstacles and adaptive rules—this project pushes the boundaries of conventional RL applications. It aims to bridge the gap between theoretical research and practical implementations, offering insights into how RL systems can adapt to real-world, unpredictable scenarios.

The research also explores the potential applications of RL in domains beyond gaming, including robotics, autonomous systems, and decision-support tools. By tackling progressively complex game variants, this project underscores the growing importance of adaptability and robustness in intelligent systems.

Background and Related Work

Overview of Reinforcement Learning

Reinforcement learning is a subset of machine learning where agents learn to maximize cumulative rewards by interacting with an environment. Unlike supervised learning, RL does not require labeled data; instead, it employs trial-and-error mechanisms to discover optimal policies.

Key advancements in RL include:

- Q-Learning: A model-free RL algorithm that learns the value of state-action pairs.
- Deep Reinforcement Learning: Combines deep neural networks with RL, enabling the handling of large and high-dimensional state spaces.

Applications to Gaming

Gaming has been a prominent domain for RL research due to its well-defined rules and reward structures. Classic works, such as DeepMind's breakthroughs with Atari games, showcase RL's ability to master complex strategies.

Previous studies on Snake games have focused on static environments. This project advances the field by introducing dynamic challenges, such as changing obstacle layouts, and analyzing the agents' adaptability to these variations. Such challenges simulate real-world scenarios where adaptability and quick decision-making are essential.

Related Work

Several studies have demonstrated the application of RL in grid-based games, emphasizing the simplicity and scalability of these environments. However, few have explored the integration of dynamic elements. By addressing this gap, the current project contributes to the evolving discourse on RL's versatility and scalability.

Methodology

The project employs the Deep Q-Learning (DQL) algorithm, an extension of Q-Learning that utilizes neural networks to approximate Q-values for state-action pairs.

Key Components

1. **Agent-Environment Interaction:** Agents interact with the Snake game by taking actions based on the current game state.
2. **Reward Structure: Agents receive:**
 - Positive rewards for collecting food.
 - Negative rewards for collisions or suboptimal movements.
3. **Experience Replay:** Past experiences are stored in a replay buffer and sampled during training to stabilize learning.
4. **Neural Network Design:**
 - Input Layer: Encodes the game state, including the snake's position, food location, and obstacle layout.
 - Hidden Layers: Three fully connected layers with ReLU activation functions.
 - Output Layer: Outputs Q-values for possible actions (up, down, left, right).

Training Pipeline

The training process integrates several steps to ensure effective learning:

1. **State Preprocessing:** Game states are preprocessed into numerical formats suitable for the neural network.
2. **Action Selection:** Actions are chosen using an ϵ -greedy strategy, balancing exploration and exploitation.
3. **Reward Assignment:** Feedback is provided based on the agent's actions.
4. **Network Updates:** Backpropagation updates the network's weights to minimize prediction errors.
5. **Periodic Evaluation:** Agent performance is evaluated at intervals to assess learning progress.

Implementation Details

1. Basic Repo Code

- **Agent (agent.py):**
 - Implements the core reinforcement learning logic with Q-Learning.
 - Uses an epsilon parameter to balance exploration and exploitation.
 - Maintains a deque for experience replay with a MAX_MEMORY of 100,000.
 - Employs a simple neural network (Linear_QNet) for action prediction.
 - Provides functionality for short-term and long-term memory training using batches of experiences.
 - **Model (model.py):**
 - Defines a two-layer feed-forward neural network (Linear_QNet) with a single hidden layer.
 - Implements basic Q-Learning with MSELoss and Adam optimizer.
 - **Game (snake_gameai.py):**
 - Simulates the Snake game environment with basic food placement logic and collision detection.
 - Tracks the score and renders the game using Pygame.
 - Simple reward structure: +10 for eating food and -10 for collision.
 - **Helper (helper.py):**
 - Includes a utility function for real-time plotting of scores and mean scores.
-

2. Refined Structure Version

- **Agent Enhancements:**
 - Modularized the codebase by adding helper functions for saving/loading records and better organization.
 - Enhanced device management for seamless switching between CPU and GPU.
 - Introduced a dynamic decay mechanism for epsilon to gradually reduce exploration.
- **Model Enhancements:**
 - Improved code structure and added more robust methods for saving and loading models.

- **Extended the model to support flexible input and output dimensions.**
 - **Game Enhancements:**
 - **No major functional changes but refined state tracking for better integration with the agent.**
 - **Helper Enhancements:**
 - **Improved plotting by adding annotations and legends for clarity.**
-

3. Fixed Obstacles v1

- **Obstacle Integration:**
 - **Introduced fixed obstacles in the game environment.**
 - **Obstacles are static and predefined in shape (e.g., horizontal/vertical lines, squares).**
 - **Ensures obstacles do not overlap with the snake or food.**
 - **Agent Updates:**
 - **Adapted the state representation to include collision checks against obstacles.**
 - **Incorporated obstacles into the `is_collision` function for better training feedback.**
 - **Model Updates:**
 - **Maintained the basic two-layer neural network.**
-

4. Fixed Obstacles v2 (Optimized)

- **Optimization in Neural Network:**
 - **Upgraded the model with a deeper architecture, additional layers, and dropout for regularization.**
 - **Replaced ReLU with LeakyReLU for improved gradient flow in the network.**
- **Training Enhancements:**
 - **Introduced a target network for more stable training.**
 - **Increased batch size and improved training performance by dynamically decaying epsilon.**
- **State Representation:**
 - **Added relative food position and obstacle proximity as additional inputs to the agent's state.**
- **Dynamic Adjustments:**
 - **Enhanced obstacle placement logic to avoid predictable patterns.**

5. Dynamic Every Reward

- **Dynamic Obstacles:**
 - Introduced logic to re-spawn obstacles every time the snake eats food.
 - Ensures that obstacles are placed away from the snake and food.
 - **Agent Adaptability:**
 - Increased the complexity of the training environment as the agent must adapt to frequent changes in obstacles.
-

6. Dynamic Every Game Over

- **Dynamic Obstacles:**
 - Obstacles are re-spawned only after the game resets, maintaining static obstacles during a single game.
 - Ensures consistent gameplay within each session.
- **Training Stability:**
 - Maintains a balance between environmental complexity and stability, enabling the agent to learn more predictably.

Game Versions and Features

Basic Repo Code

Key Features:

- **Device Handling:** Direct use of `.cuda()` for GPU processing without fallback mechanisms for CPU.
 - **Training Logic:**
 - Infinite training loop without termination conditions.
 - Limited exploration strategy with fixed epsilon.
 - **Persistence:** High scores and model files are not persistently saved or versioned.
 - **Feedback:** Minimal logging or user feedback about training progress.
 - **Testing:** No dedicated test script; testing required modifying the main training code.
-

Refined Structure Code

Key Features:

- **Device Handling:** Introduced `torch.device` to dynamically handle GPU/CPU, ensuring compatibility across environments.
 - **Training Logic:**
 - Training terminates after 1000 games, providing clear stopping conditions.
 - Adaptive epsilon for balancing exploration and exploitation.
 - Tracks recent performance trends with scores from the last 20 games.
 - **Persistence:**
 - High scores saved persistently in `record.txt`.
 - Models saved only when achieving a new record, with versioned filenames.
 - **Feedback:** Detailed logs with performance milestones and model-saving updates.
 - **Testing:** A dedicated `load_test.py` script for standalone testing, supporting seamless evaluation.
-

Comparison: Basic Repo vs Refined Structure

Aspect	Basic Repo Code	Refined Structure Code
Device Handling	Hardcoded <code>.cuda()</code> without fallback.	Dynamically selects GPU/CPU using <code>torch.device</code> .
Training Logic	No stopping condition; fixed epsilon.	Stops after 1000 games; adaptive epsilon for exploration.

Aspect	Basic Repo Code	Refined Structure Code
Persistence	No high score tracking; models saved with generic filenames.	Tracks high scores in <code>record.txt</code> ; models saved with versioned filenames.
Feedback	Minimal logging and feedback.	Detailed logging of milestones and training progress.
Testing	Testing requires modifying the main script.	Dedicated <code>load_test.py</code> script for testing.
Memory Management	Fixed-size memory; lacks batch sampling diversity.	Efficient <code>deque</code> memory with random batch sampling.

Fixed Obstacles v1 Code

Key Features:

- **Static Obstacles:** Introduced fixed obstacles to increase game complexity.
 - **Reward System:** Basic rewards (+10 for food, -10 for collisions) without intermediate feedback.
 - **State Representation:**
 - 11 input features: danger indicators, move direction, and food relative location.
 - Lacks contextual features like distances to obstacles.
 - **Neural Network:**
 - Simple architecture: one hidden layer with 256 neurons and ReLU activation.
 - Potential for overfitting with no dropout layers.
-

Fixed Obstacles v2 (Optimized) Code

Key Features:

- **Dynamic Enhancements:**
 - Improved reward system: penalizes moving away from food and rewards moving closer.
 - Added intermediate feedback for guiding the agent's behavior.
- **State Representation:**
 - Expanded to 12 inputs, including normalized distances to food.
 - Placeholder inputs for future obstacle proximity and tail features.
- **Neural Network:**
 - Deeper architecture with three hidden layers (256 neurons each).
 - LeakyReLU activation to avoid the "dying ReLU" problem.
 - Dropout layers for regularization to prevent overfitting.
- **Hyperparameters:**
 - Reduced learning rate for finer adjustments.
 - Increased batch size to stabilize updates.
- **Advanced Techniques:** Double DQN with a target network for improved stability.

Comparison: Fixed Obstacles v1 vs Fixed Obstacles v2 (Optimized)

Aspect	Fixed Obstacles v1	Fixed Obstacles v2 (Optimized)
Reward System	Basic rewards for food/collisions.	Rewards moving closer to food, penalizes moving away.
State Representation	11 inputs: danger, direction, food location.	12 inputs: includes normalized distances, future placeholders.
Neural Network	Single hidden layer (256 neurons).	Three hidden layers (256 neurons each) with dropout and LeakyReLU.
Hyperparameters	Higher learning rate; smaller batch size.	Lower learning rate; larger batch size.
Advanced Techniques	Standard DQN.	Double DQN with a target network for stability.

Dynamic Every Reward Code

Key Features:

- **Dynamic Environment:** Re-spawns obstacles and food after every food collection.
 - **Agent Adaptability:** Requires rapid adaptation to frequent environmental changes.
 - **Reward System:** Similar to Fixed Obstacles v2, with added penalties for inefficient movement.
-

Dynamic Every Game Over Code

Key Features:

- **Dynamic Environment:** Re-spawns obstacles only after game resets, balancing stability and variability.
 - **Agent Learning:** Allows the agent to adapt over longer, stable gameplay before facing new environments.
 - **Reward System:** Retains the enhanced reward system of Fixed Obstacles v2.
-

Comparison: Dynamic Every Reward vs Dynamic Every Game Over

Aspect	Dynamic Every Reward	Dynamic Every Game Over
Environment Dynamics	Obstacles re-spawn with every food collection.	Obstacles re-spawn only after game resets.

Aspect	Dynamic Every Reward	Dynamic Every Game Over
Stability	High variability, requiring rapid adaptability.	Balanced variability, allowing stable learning within each game.
Reward System	Penalizes inefficient movement; rewards efficient paths.	Same reward system as Dynamic Every Reward.

This section outlines the features and comparisons of each version, highlighting the iterative improvements made in the game's development. Each version reflects advancements in gameplay complexity, agent adaptability, and learning stability.

Challenges and Solutions

1. Basic Repo vs Refined Structure

Challenges:

1. Code Readability and Maintenance:

- The basic repo code lacked modularity, making it difficult to debug or extend.
- Device management (CPU/GPU) was inconsistently implemented, requiring manual adjustments.

2. Performance Tracking:

- Limited functionality for tracking game progress and performance metrics over multiple games.

3. Epsilon Decay:

- The exploration-exploitation tradeoff was handled statically, limiting the agent's adaptability.

Solutions:

1. Improved Code Structure:

- Modularized components (e.g., helper functions for saving/loading records).
- Enhanced parameter organization and better abstraction.

2. Dynamic Device Management:

- Implemented seamless support for GPU and CPU training using a device variable.

3. Enhanced Performance Tracking:

- Added functionality to track the best scores over recent games and visualize progress with better plots.

4. Dynamic Epsilon Decay:

- Introduced a gradual decay mechanism for epsilon, improving exploration during early training phases and exploitation later.

2. Fixed Obstacles v1 vs Fixed Obstacles v2 (Optimized)

Challenges:

1. Model Performance:

- The basic two-layer neural network in v1 struggled with the added complexity of fixed obstacles.
- 2. **Training Stability:**
 - Training in v1 lacked mechanisms to handle fluctuations caused by large batch sizes and high learning rates.
- 3. **State Representation:**
 - In v1, the state representation only considered immediate collision checks, missing broader contextual information.
- 4. **Obstacle Placement:**
 - Fixed obstacles in v1 followed predefined patterns, potentially leading to overfitting or lack of generalization.

Solutions:

- 1. **Enhanced Neural Network Architecture:**
 - v2 introduced a deeper network with additional layers, dropout for regularization, and LeakyReLU activations for better performance.
- 2. **Stable Training Mechanisms:**
 - Added a target network in v2 to stabilize Q-value updates and improve learning.
- 3. **Improved State Representation:**
 - v2 included relative food position and obstacle proximity, providing the agent with a more comprehensive understanding of the environment.
- 4. **Dynamic Obstacle Placement:**
 - v2 implemented random placement for fixed obstacles, increasing variability and generalization during training.

3. Dynamic Every Reward vs Dynamic Every Game Over

Challenges:

- 1. **Dynamic Every Reward:**
 - The frequent re-spawning of obstacles after every food collection significantly increased the environment's complexity.
 - The agent struggled to adapt to the constantly changing obstacle layout, leading to slower learning and inconsistent performance.
- 2. **Dynamic Every Game Over:**

- **While the static nature of obstacles within a game session improved stability, it reduced the environmental variability, potentially limiting the agent's ability to generalize.**

Solutions:

1. Balancing Complexity:

- **In Dynamic Every Reward, careful tuning of training parameters (e.g., epsilon decay, learning rate) mitigated the impact of frequent environment changes.**

2. Controlled Variability:

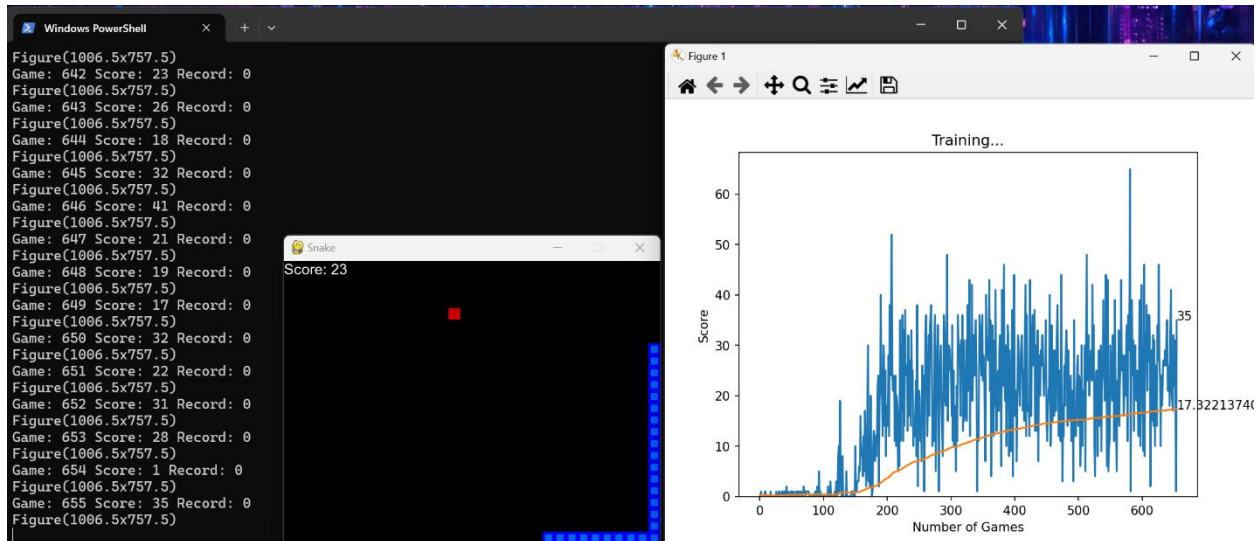
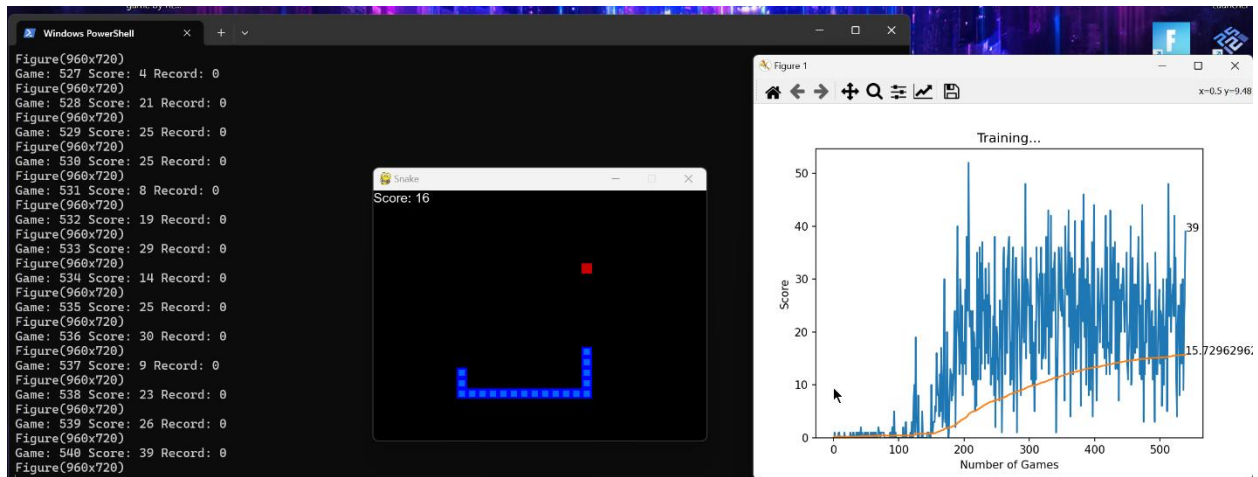
- **In Dynamic Every Game Over, maintaining obstacle consistency within a session while introducing randomness between games helped balance training stability and generalization.**

3. Reward Structuring:

- **Both versions benefited from refined reward systems (e.g., penalizing moves away from the food) to guide the agent's learning in complex environments.**

Results and Visualizations

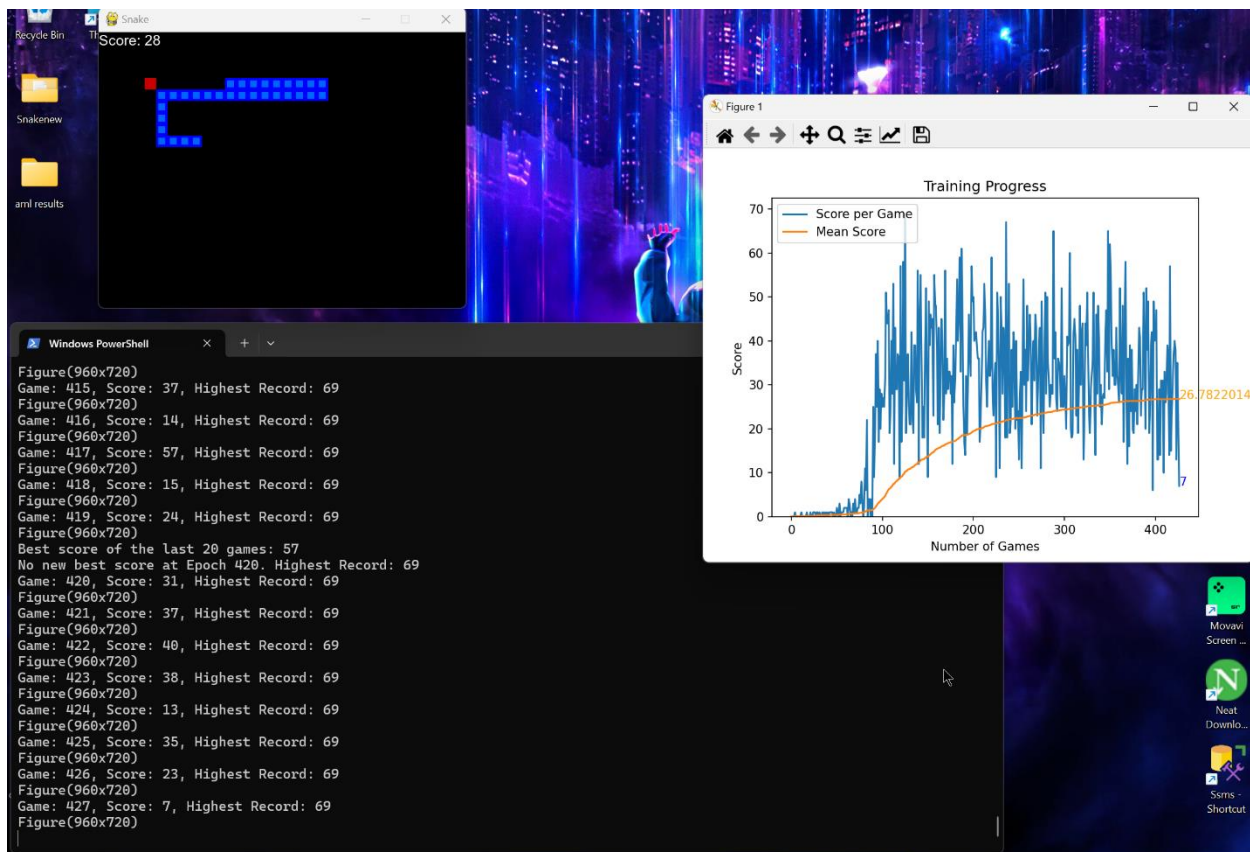
Basic Repo Snake Game:



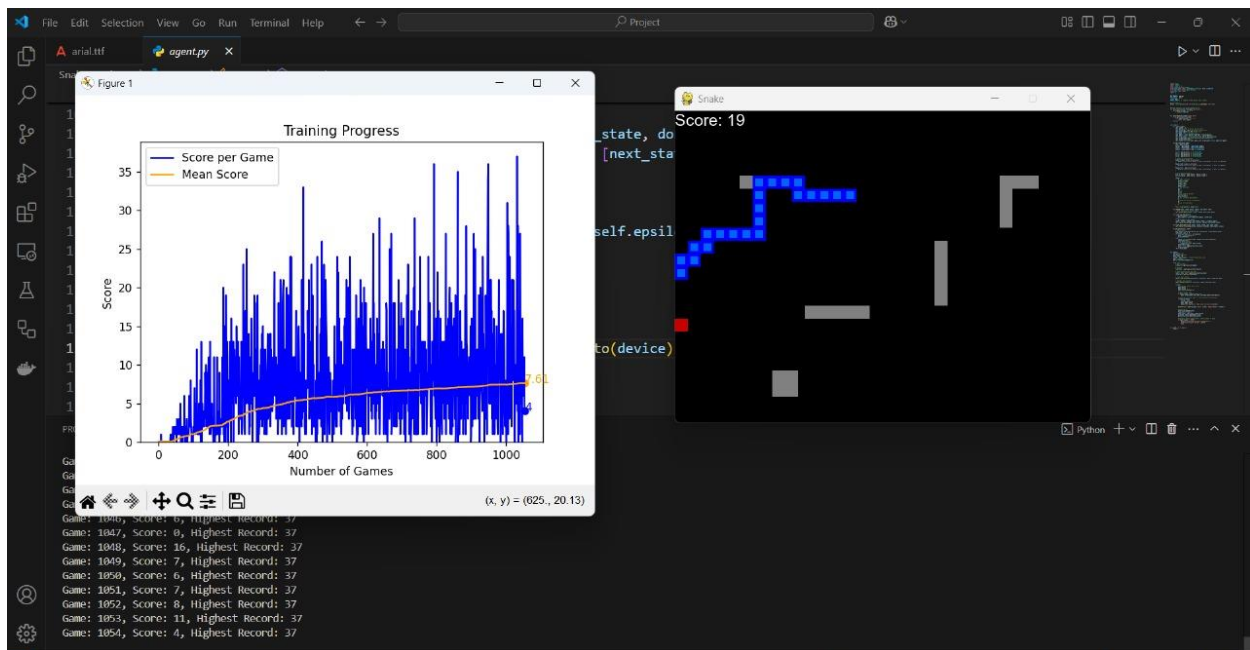
Refined Structure Snake Game

```

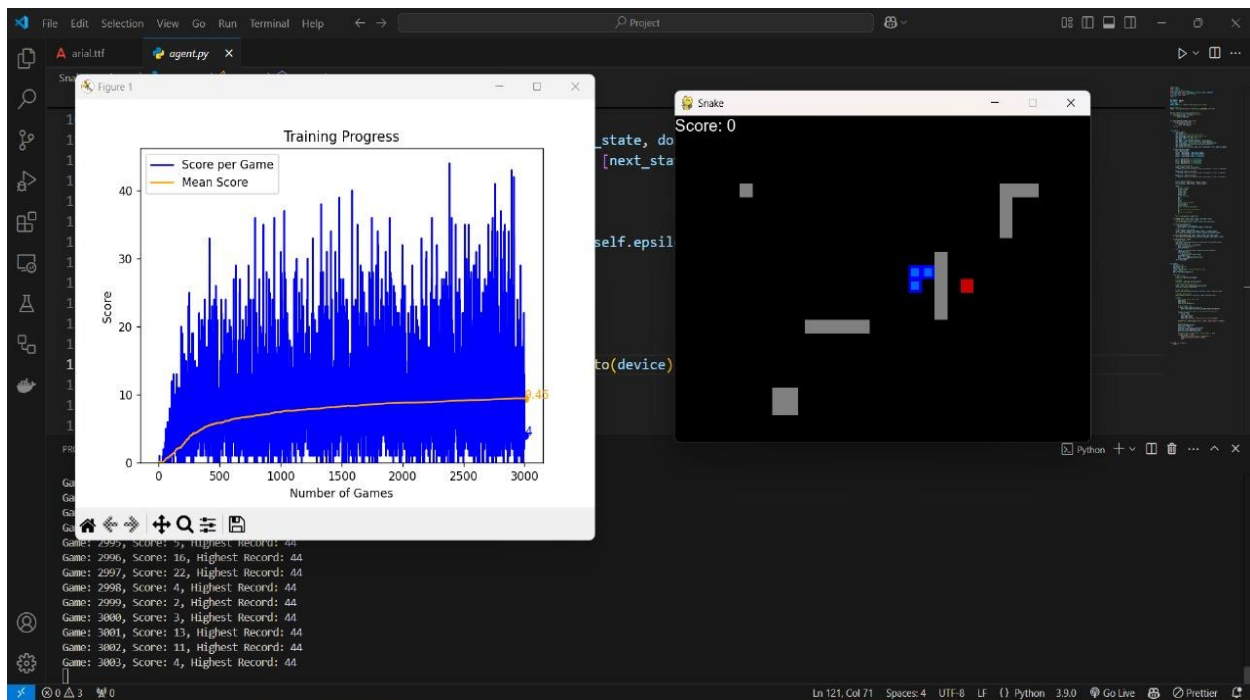
Windows PowerShell
Figure(960x720)
Game: 989, Score: 40, Highest Record: 77
Figure(960x720)
Game: 990, Score: 36, Highest Record: 77
Figure(960x720)
Game: 991, Score: 30, Highest Record: 77
Figure(960x720)
Game: 992, Score: 33, Highest Record: 77
Figure(960x720)
Game: 993, Score: 18, Highest Record: 77
Figure(960x720)
Game: 994, Score: 46, Highest Record: 77
Figure(960x720)
Game: 995, Score: 9, Highest Record: 77
Figure(960x720)
Game: 996, Score: 33, Highest Record: 77
Figure(960x720)
Game: 997, Score: 20, Highest Record: 77
Figure(960x720)
Game: 998, Score: 40, Highest Record: 77
Figure(960x720)
Game: 999, Score: 33, Highest Record: 77
Figure(960x720)
Best score of the last 20 games: 53
No new best score at Epoch 1000. Highest Record: 77
Game: 1000, Score: 17, Highest Record: 77
Figure(960x720)
Training completed after 1000 games.
Final Highest Record: 77
PS C:\Users\Nour\Desktop\Snakenew>
    
```

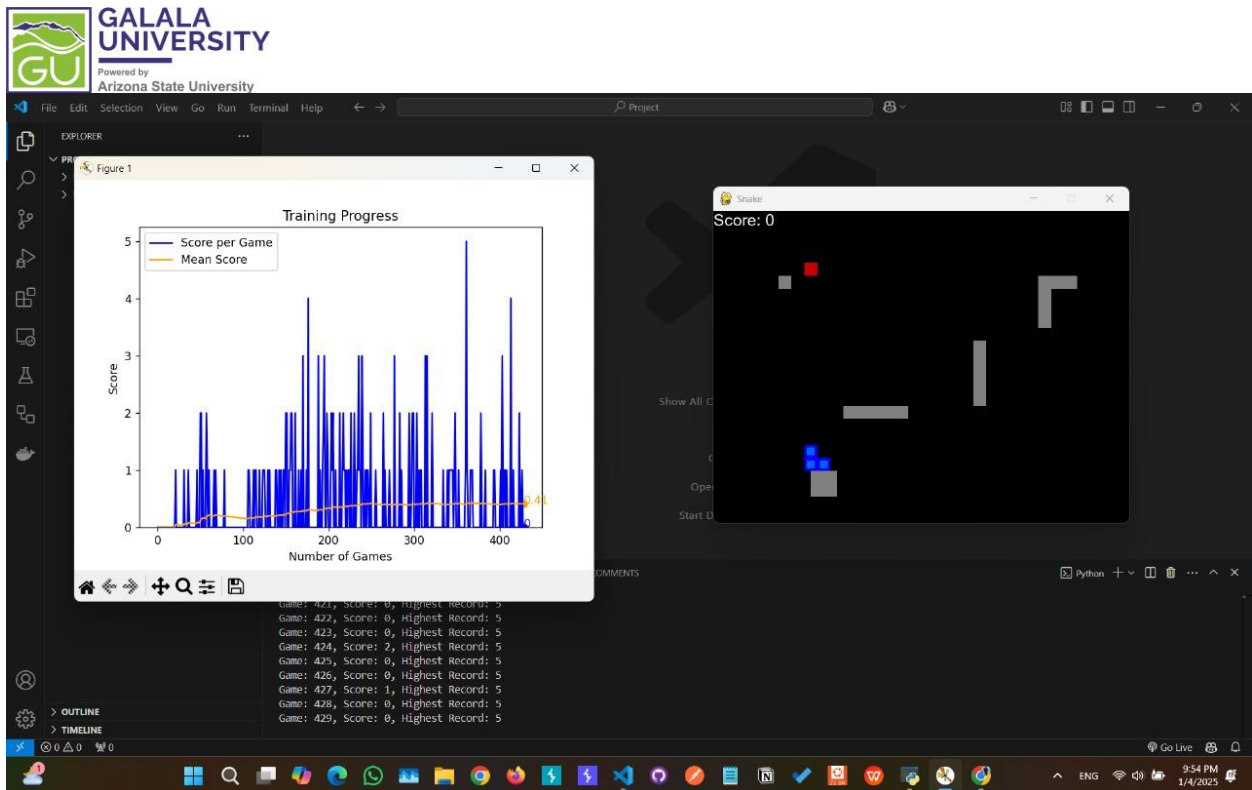


Fixed obstacles v1

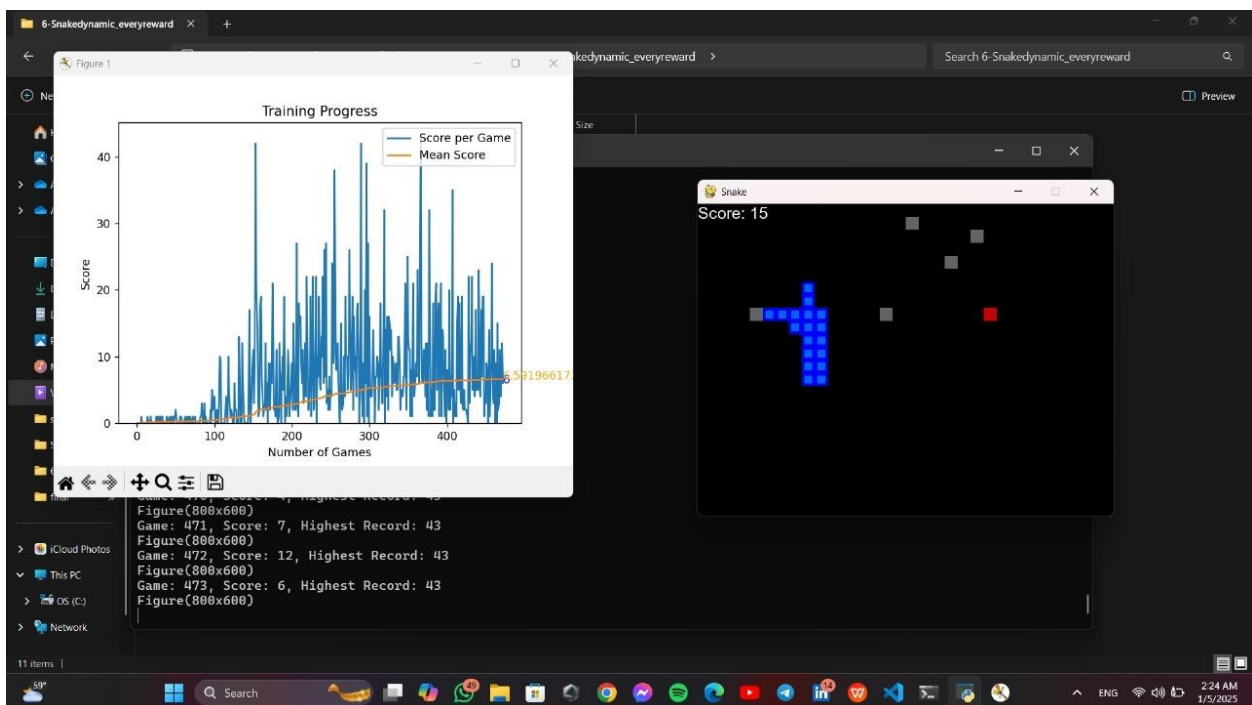


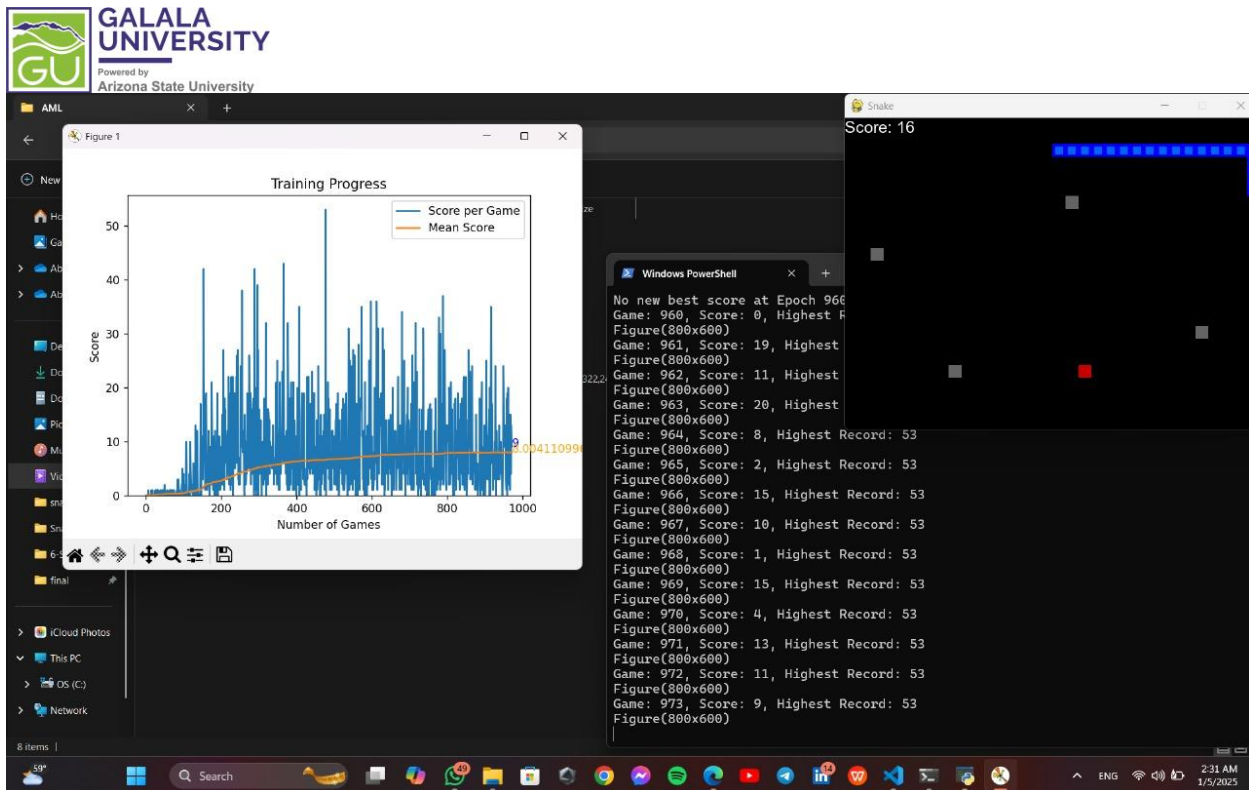
Fixed obstacles v2 (optimized)



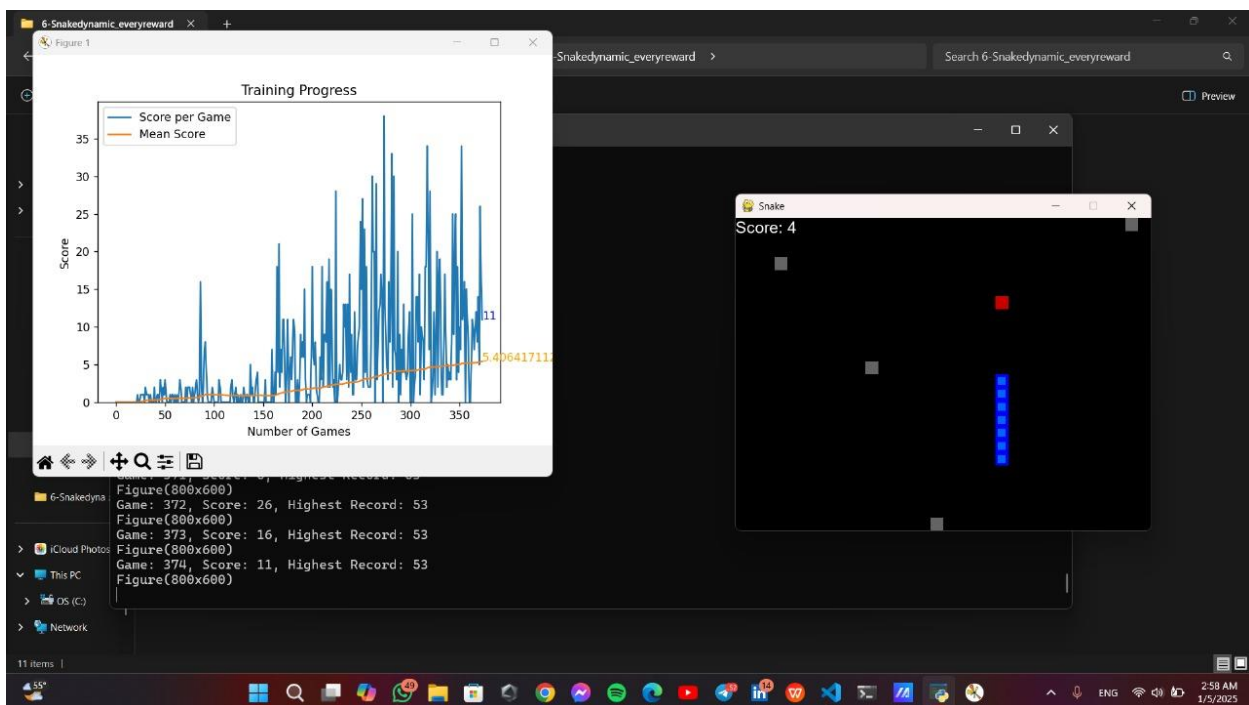


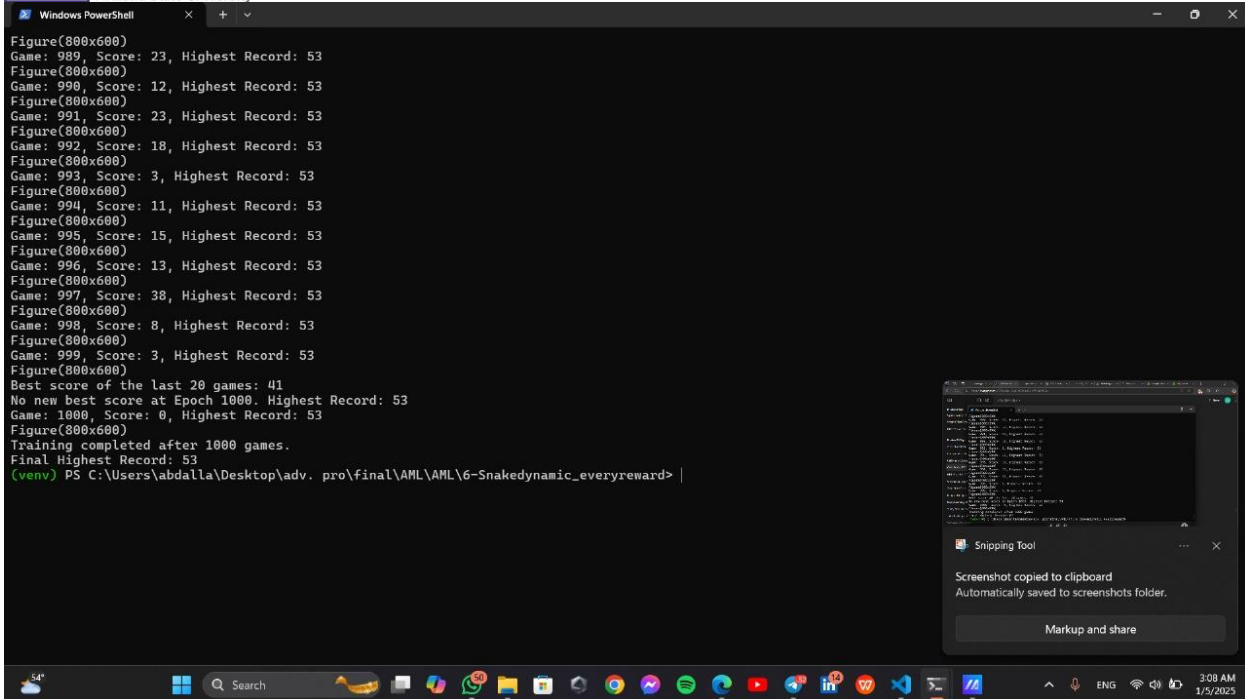
Dynamic Obstacles changing every reward





Dynamic Obstacles changing every game over





```

Windows PowerShell
Figure(800x600)
Game: 989, Score: 23, Highest Record: 53
Figure(800x600)
Game: 990, Score: 12, Highest Record: 53
Figure(800x600)
Game: 991, Score: 23, Highest Record: 53
Figure(800x600)
Game: 992, Score: 18, Highest Record: 53
Figure(800x600)
Game: 993, Score: 3, Highest Record: 53
Figure(800x600)
Game: 994, Score: 11, Highest Record: 53
Figure(800x600)
Game: 995, Score: 15, Highest Record: 53
Figure(800x600)
Game: 996, Score: 13, Highest Record: 53
Figure(800x600)
Game: 997, Score: 38, Highest Record: 53
Figure(800x600)
Game: 998, Score: 8, Highest Record: 53
Figure(800x600)
Game: 999, Score: 3, Highest Record: 53
Figure(800x600)
Best score of the last 20 games: 41
No new best score at Epoch 1000. Highest Record: 53
Game: 1000, Score: 0, Highest Record: 53
Figure(800x600)
Training completed after 1000 games.
Final Highest Record: 53
(venv) PS C:\Users\abdalla\Desktop\adv. pro\final\AML\AML\6-Snakedynamic_everyreward>
  
```

Conclusion & Future Enhancements

Conclusion

The development journey of the Snake game using reinforcement learning demonstrates a structured and iterative approach to enhancing both gameplay complexity and the agent's learning capabilities. Starting from the Basic Repo Code, the project evolved through multiple stages, each addressing specific challenges and introducing significant improvements.

1. Basic Repo to Refined Structure:

The foundational version of the game provided a simple setup for training a Q-learning agent. Transitioning to the refined structure version improved code modularity, readability, and performance tracking. These changes laid the groundwork for further advancements by making the code more adaptable and efficient.

2. Fixed Obstacles v1 to Fixed Obstacles v2 (Optimized):

The introduction of static obstacles in v1 increased the game's complexity, challenging the agent's ability to navigate constrained environments. Optimizations in v2, such as an enhanced neural network, dynamic obstacle placement, and a target network for stability, significantly improved the agent's learning efficiency and generalization.

3. Dynamic Every Reward to Dynamic Every Game Over:

Moving to dynamic obstacles added variability to the environment, pushing the agent to adapt to changing scenarios. The Dynamic Every Reward version introduced frequent changes, requiring

high adaptability, while the Dynamic Every Game Over version balanced stability and variability by introducing changes only after game resets.

Key Takeaways:

- **Iterative Development:**
Each stage of development addressed specific limitations of the previous version, showcasing the importance of an iterative approach in reinforcement learning projects.
 - **Balancing Complexity and Stability:**
As the game's complexity increased, careful adjustments to training parameters, model architecture, and state representations were necessary to maintain stability and ensure effective learning.
 - **Enhanced Adaptability:**
From static to dynamic obstacles, the agent's ability to generalize and adapt improved significantly, highlighting the value of incremental environmental challenges.
-

Future Prospects:

- **Environment Expansion:**
Introducing more diverse obstacles, larger game grids, or multiple food items could further enhance the agent's learning capabilities.
- **Advanced Learning Techniques:**
Incorporating more sophisticated RL algorithms, such as Deep Deterministic Policy Gradient (DDPG) or Proximal Policy Optimization (PPO), could unlock higher efficiency in learning.
- **Transfer Learning:**
Leveraging the trained model to perform well in entirely new and unfamiliar game scenarios would be an exciting next step.

This project demonstrates how iterative refinement, paired with innovative solutions, can build increasingly intelligent agents capable of tackling progressively complex tasks. The outcomes of this journey provide a strong foundation for future advancements in reinforcement learning applications.