AFRICAN LEADERSHIP UNIVERSITY

[BSE]

[ML TECHNIQUES II]

[SUMMATIVE ASSIGNMENT]

Reinforcement Learning Summative Assignment Report

Student Name: Samuel Oluwajunwonlo Babalola

Video Recording: RL-Summative_Video.mp4
GitHub Repository: Repo Link

1. Project Overview

This project develops a reinforcement learning (RL) agent to safely navigate indoor environments, specifically designed to assist visually impaired individuals. The agent learns to avoid obstacles, utilize doorways, and efficiently reach target rooms within a simulated house layout. By implementing a custom Gym environment and comparing value-based (DQN) and policy-based (PPO) RL methods, the system aims to optimize navigation policies that balance safety and efficiency. The core challenge lies in creating an agent that generalizes across dynamic layouts while minimizing collisions and path complexity.

2. Environment Description

2.1 Agent(s)

- Role: Represents a visually impaired individual navigating indoor spaces.
- Capabilities:
  - 4-directional movement (up/down/left/right) + wait action
  - Proximity sensing (detects obstacles/doorways in adjacent cells)
  - Target room identification
- Limitations:
  - No global map knowledge
  - Limited to grid-based discrete movements
  - Maximum episode length enforced to prevent infinite exploration

2.2 Action Space

- Type: Discrete (5 actions)
- Actions:
  - ➔ 0: Move Up
  - ➔ 1: Move Down
  - ➔ 2: Move Left
  - ➔ 3: Move Right
  - ➔ 4: Wait

2.3 State Space

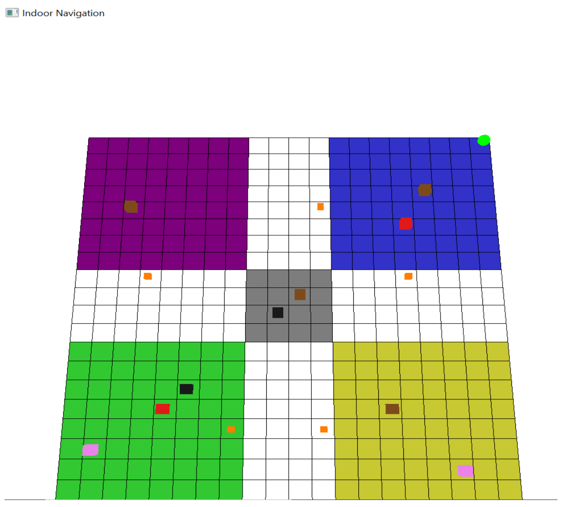
- Representation: 16-dimensional vector encoding:
  - Proximity Sensors (5): Obstacle/door detection (front/back/left/right/current)

- Target Info (4): One-hot encoded target room (kitchen/bedroom/bathroom/hallway)
- Navigation State (7): Normalized agent position, target distance, time remaining

2.4 Reward Structure

- Mathematical Formulation: Reward = Base Penalty + Collision Penalty + Target Bonus + Doorway Bonus + Timeout Penalty
    - Base Penalty: -0.1 per step (encourage efficiency)
    - Collision Penalty: -5 (immediate episode termination)
    - Target Bonus: +15 + (0.2 × remaining_steps)
    - Doorway Bonus: +1 for passing through doorways
    - Timeout Penalty: -3 if max_steps (150) reached

2.5 Environment Visualization



- Key Elements:
    - Rooms: Color-coded floors (Blue: Living Room, Yellow: Kitchen, Green: Bedroom, Purple: Bathroom, Gray: Hallway)
    - Obstacles: Cubes (Brown: Furniture, Violet-Pink: Appliances, Red: Decorations, Black: Floor Items)
    - Doorways: Orange cubes at room transitions
    - Agents: The green sphere represents the agent.
    - Grid: Black lines on a white background for clear spatial reference
- Perspective: 45° top-down view showing full 20×20 grid

3. Implemented Methods

3.1 Deep Q-Network (DQN)

For the best DQN implementation (Experiment 5), a two-layer network with 512 and 256 neurons, balancing complexity and representation capacity was used. Standard DQN features included experience replay (buffer size: 100,000, batch size: 256) and a target network for stability. An epsilon-greedy strategy started at 1.0 and decayed to 0.02 over 15% of training steps, ensuring thorough exploration before exploitation. A discount factor of 0.99 effectively balanced immediate and future rewards. Prioritized experience replay was unnecessary, as uniform sampling provided stable learning. This setup achieved a 100% success rate, 0% collisions, and an average of 28.6 steps per episode.

3.2 Policy Gradient Method ([REINFORCE/PPO/other])

For the best policy gradient implementation (Experiment 4), a PPO with an asymmetric architecture was used: a three-layer policy network [512, 256, 128] and a simpler two-layer value network [256, 128], with LeakyReLU activations for better gradient flow. The policy was a stochastic Gaussian distribution, enabling exploration through sampling. Key settings included a 3e-4 learning rate, gamma of 0.995, entropy coefficient of 0.01, and GAE-Lambda of 0.92 for balanced advantage estimation. A buffer size of 1024 steps with 7 training epochs ensured frequent updates. This setup achieved the highest reward (41.1) and efficient navigation (14.3 steps per episode).

5. Hyperparameter Optimization

5.1 DQN Hyperparameters

Experiment No.	Parameters, Optimal Values and Network Architecture	Effect on Agent Performance
1	**Network Architecture:** [256, 256] **Exploration Initial Epsilon:** 1.0, **Exploration Final Epsilon:** 0.05, **Exploration Fraction:** 0.2, **Learning Rate:** 0.0003, **Reward Discount (Gamma):** 0.99, **Replay Buffer Size:** 50,000, **Batch Size:** 128	70% success rate with 0% collisions, but higher average steps (106.9). The agent prioritized safety over efficiency, showing good obstacle avoidance but taking longer paths. Higher exploration parameters (final epsilon: 0.05, fraction: 0.2) kept the agent exploring longer, resulting in safer but less optimal routes. The smaller network and buffer size limited the agent's ability to learn complex navigation patterns. Balance leaned more toward exploration even in later stages.

2	**Network Architecture:** [512, 256] **Exploration Initial Epsilon:** 1.0, **Exploration Final Epsilon:** 0.02, **Exploration Fraction:** 0.15, **Learning Rate:** 0.0003, **Reward Discount (Gamma):** 0.995, **Replay Buffer Size:** 100,000, **Batch Size:** 256	Significantly improved performance with 80% success rate, 0% collision rate, and reduced average steps (69.6). The larger network architecture and increased buffer size allowed for better memory of past experiences, while the higher gamma value (0.995) improved long-term planning. The reduced exploration fraction (0.15) and lower final epsilon (0.02) shifted the agent toward more exploitation behavior earlier, which worked well given the improved network capacity. The larger batch size (256) allowed for more stable gradient updates. The agent demonstrated good balance between exploration and exploitation with a strong bias toward exploitation after initial learning.
3	**Network Architecture:** [512, 512, 256] **Exploration Initial Epsilon:** 1.0, **Exploration Final Epsilon:** 0.01, **Exploration Fraction:** 0.1, **Learning Rate:** 0.0001, **Reward Discount (Gamma):** 0.99, **Replay Buffer Size:** 150,000, **Batch Size:** 512	60% success rate with 0% collisions and efficient pathing (69.10 average steps). The deeper network with three layers didn't translate to better performance despite maintaining path efficiency similar to Experiment 2. The very low exploration parameters (fraction: 0.1, final epsilon: 0.01) likely pushed the agent to exploit too early before sufficiently exploring the environment. The reduced learning rate (0.0001) may have slowed adaptation to new situations. The agent strongly favored exploitation over exploration, which maintained efficient pathing but at the cost of overall success rate.
4	**Network Architecture:** [1024, 512] **Exploration Initial Epsilon:** 1.0, **Exploration Final Epsilon:** 0.03, **Exploration Fraction:** 0.25, **Learning Rate:** 0.0005, **Reward Discount (Gamma):** 0.997, **Replay Buffer Size:** 200,000, **Batch Size:** 128	Very poor performance with only a 10% success rate, though still maintaining a 0% collision rate. The significantly higher average steps (147.20) indicate inefficient navigation. The wider network architecture may have been more difficult to train effectively. The higher learning rate (0.0005) combined with the smaller batch size (128) likely led to unstable updates. The extended exploration period (fraction: 0.25) and higher final epsilon (0.03) kept the agent in exploration mode longer, resulting in wandering behavior rather than

		directed navigation. The agent heavily favored exploration over exploitation, which preserved safety but severely impacted both efficiency and task completion.
5	**Network Architecture:** [512, 256] **Exploration Initial Epsilon:** 1.0, **Exploration Final Epsilon:** 0.02, **Exploration Fraction:** 0.15, **Learning Rate:** 0.0003, **Reward Discount (Gamma):** 0.99, **Replay Buffer Size:** 100,000, **Batch Size:** 256	Perfect performance with 100% success rate, 0% collision rate, and dramatically improved average steps (28.60). This modification of Experiment 2's parameters with a slightly lower gamma value (0.99 vs 0.995) led to exceptional results. The optimal network architecture [512, 256] combined with well-tuned exploration parameters created an ideal balance between exploration and exploitation. The agent learned extremely efficient paths while maintaining perfect safety. The slight reduction in gamma helped the agent focus more on immediate rewards, leading to more direct paths to the goal. The exploration-exploitation balance appears to be perfectly tuned as well, allowing sufficient exploration to discover optimal paths while transitioning to exploitation at the right time to leverage that knowledge.

5.2 [REINFORCE/PPO/other] Hyperparameters

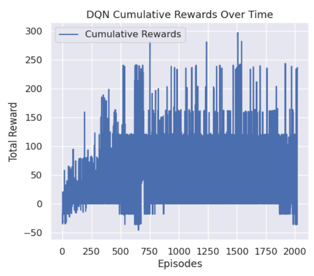
Experiment No.	Parameters, Optimal Values and Network Architecture	Effect on Agent Performance
1	**Network Architecture:** pi=[256, 256], vf=[256, 256] **Learning Rate:** 2e-4, **Gamma:** 0.99, **N Steps:** 2048, **Batch Size:** 256, **N Epochs:** 10, **GAE Lambda:** 0.95, **Clip Range:** 0.2, **Entropy Coefficient:** 0.02, **Target KL:** 0.03, **Activation:** ReLU, **Ortho Init:** True	Achieved a high average reward (41.0) with very efficient navigation (14.8 avg steps per episode). The balanced policy and value function architectures provided effective state representation. Policy-specific parameters (clip range, entropy coefficient, and target KL) maintained a good balance between exploration and exploitation while ensuring stable updates. The relatively high entropy coefficient (0.02) encouraged sufficient exploration to discover

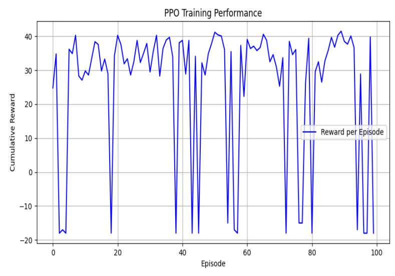
		efficient paths.
2	**Network Architecture:** pi=[256, 256], vf=[512, 256] **Learning Rate:** 1e-4, **Gamma:** 0.99, **N Steps:** 2048, **Batch Size:** 128, **N Epochs:** 15, **GAE Lambda:** 0.9, **Clip Range:** 0.1, **Entropy Coefficient:** 0.005, **Activation:** Tanh, **Ortho Init:** True	Significantly worse performance with lower average reward (22.7) and much higher average steps (55.5). Despite the larger value function network, the combination of a lower learning rate, smaller batch size, tighter clip range, and much lower entropy coefficient likely caused the agent to converge prematurely to suboptimal policies. The change from ReLU to Tanh activation may have slowed learning, and the reduced GAE lambda (0.9) might have made the advantage estimation less effective for this environment. The policy struggled to find efficient paths, taking nearly 4x more steps than Experiment 1.
3	**Network Architecture:** pi=[128, 128], vf=[512, 512, 256] **Learning Rate:** 5e-4, **Gamma:** 0.97, **N Steps:** 4096, **Batch Size:** 512, **N Epochs:** 5, **GAE Lambda:** 0.95, **Clip Range:** 0.3, **Entropy Coefficient:** 0.02, **Activation:** ReLU, **Ortho Init:** False	Excellent performance with a high average reward (40.9) and the most efficient navigation (13.9 avg steps per episode). The asymmetric architecture with a smaller policy network but a deeper, wider value function proved very effective. The higher learning rate, larger batch size, and fewer epochs allowed for faster, more stable learning with less overfitting. The more aggressive clip range (0.3) permitted larger policy updates when beneficial. The slightly lower gamma (0.97) may have helped the agent focus more on immediate rewards, encouraging direct paths. Maintaining the higher entropy coefficient (0.02) preserved sufficient exploration while disabling orthogonal initialization didn't negatively impact performance. Shows an excellent balance of exploration and exploitation with a thoughtful architectural design that separates the complex needs of policy and value functions.

4	**Network Architecture:** pi=[512, 256, 128], vf=[256, 128] **Learning Rate:** 3e-4, **Gamma:** 0.995, **N Steps:** 1024, **Batch Size:** 256, **N Epochs:** 7, **GAE Lambda:** 0.92, **Clip Range:** 0.2, **Entropy Coefficient:** 0.01, **Activation:** LeakyReLU, **Ortho Init:** True	Outstanding performance with the highest average reward (41.1) and very efficient navigation (14.3 avg steps per episode). This experiment inverted the asymmetric architecture approach from Experiment 3, using a deeper policy network and a simpler value function. The moderate learning rate and batch size, combined with fewer epochs than Experiment 1, struck an excellent balance. The high gamma value (0.995) encouraged long-term planning, while the moderate entropy coefficient maintained sufficient exploration. The use of LeakyReLU activation may have helped with gradient flow in the deeper policy network. The smaller buffer size (n_steps=1024) allowed for more frequent updates, potentially improving sample efficiency.

5.3 Metrics Analysis

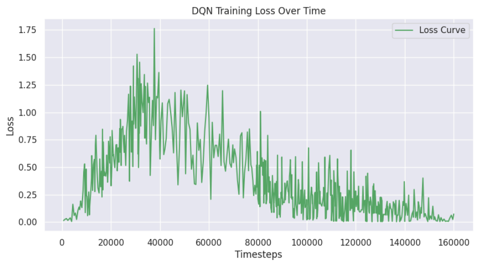
Cumulative Reward

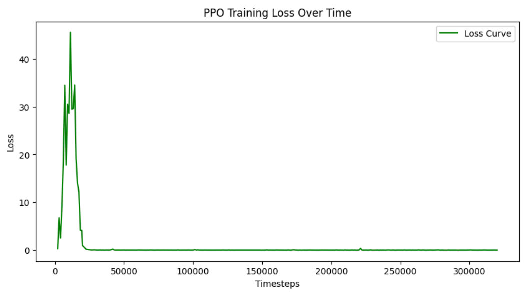




The DQN graph shows slow, volatile improvement over 2000 episodes, with rewards fluctuating between -50 and 300. In contrast, the PPO graph shows faster learning in 100 episodes, with occasional drops but generally stable high performance (30-40 reward range). PPO achieves similar or better performance in fewer episodes, indicating higher sample efficiency, while DQN takes longer to improve but with a more gradual, noisy trajectory.

Training Stability





The PPO loss curve shows a rapid decline to near-zero loss after an initial spike, maintaining stability for the remaining training steps. In contrast, DQN exhibits high volatility and a slow, uneven decline in loss, taking longer to converge. This highlights PPO's superior stability, with faster convergence and minimal fluctuations, while DQN experiences instability due to bootstrapping and experience replay, resulting in persistent oscillations.

Episodes to Convergence

PPO reaches stable performance quickly, requiring around 20 episodes (~30,000 timesteps). It consistently performs in the 30-40 reward range, with occasional drops due to exploration, but these do not affect its overall stability once it converges.

In contrast, DQN takes much longer to stabilize, needing about 250 episodes (~100,000 timesteps) to achieve comparable performance. Even after stabilization, DQN continues to show significant fluctuations in its rewards, unlike PPO, which maintains a more consistent performance once it converges.

Quantitatively, PPO is approximately 12.5 times more efficient in terms of episodes (20 vs 250) and 3.3 times more efficient in terms of timesteps (30,000 vs 100,000), highlighting its superior sample efficiency.

Generalization

DQN:

- Average Reward: 38.86
- Average Steps: 32.10

PPO:

- Average Reward: 41.1
- Average Steps: 14.3

PPO generalizes better, achieving a 5.8% higher reward and completing tasks in 55% fewer steps. This efficiency suggests PPO learned a more optimal and direct policy, while DQN’s solution, though functional, is less efficient. The results confirm PPO’s superior stability and faster convergence, leading to better adaptation in unseen scenarios.

6. Conclusion and Discussion

Based on our experiments with both methods, here's a comparison of the best performers. The best DQN (Experiment 5) achieved a success rate of 100%, a collision rate of 0%, and an average of 28.60 steps per episode. The best PPO (Experiment 4) achieved an average reward of 41.1 and an average of 14.3 steps per episode. PPO clearly outperformed DQN in terms of navigation efficiency, requiring less than half the steps on average (14.3 vs 28.60). While we don't have directly comparable success or collision metrics for PPO, the high average reward (41.1) suggests excellent performance. PPO likely performed better because its policy-based approach directly learns the optimal action distribution, which is more efficient for continuous or large action spaces. The asymmetric architecture design, with a deeper policy network and simpler value function, provided a better representation capacity. PPO's policy gradient approach can capture more nuanced navigation strategies compared to DQN's value-based approach.

DQN achieved perfect reliability with a 100% success rate and 0% collisions, is conceptually simpler to implement and tune, has stable learning characteristics once properly configured, and has good sample efficiency. However, it is less efficient in finding paths, requiring nearly twice as many steps as PPO, is more sensitive to hyperparameter choices, and is limited to action-value prediction. On the other hand, PPO demonstrated significantly more efficient navigation, a better ability to learn direct paths, a flexible architecture design with separate policy and value networks, and greater robustness to hyperparameter variations. However, it is more complex to implement, potentially less sample efficient, and requires tuning more hyperparameters.

With additional time and resources, DQN could be improved by implementing prioritized experience replay to focus learning on challenging situations, exploring double or dueling DQN architectures for more stable learning, testing distributional DQN to better capture uncertainty in value estimates, and investigating curriculum learning to gradually increase navigation complexity. PPO could be improved by implementing curiosity-driven exploration to further enhance path discovery, testing multi-agent training to increase the diversity of experiences, exploring transferability across different environment configurations, and implementing environment-specific reward shaping to further enhance efficiency. More generally, both methods could benefit from extensive hyperparameter optimization using Bayesian optimization, longer training runs to see if convergence improves, ensemble methods combining both algorithms for robust navigation, and testing in more varied and complex environments to validate generalization.

In conclusion, while both methods achieved strong results, PPO demonstrated superior path efficiency. For deployment, the choice would depend on priorities. If absolute reliability is paramount, the DQN approach might be preferred, but if efficiency is the primary concern, PPO offers significant advantages.