Implementation of Reinforcement Learning Agent Plugins Using A2C and PPO Methods in the Godot Engine

Paulus Lucky Tirma Irawan[1*], Marvin Adinata[1], and Mochamad Subianto[1]
[1]Informatics Engineering, Faculty of Technology and Design, Ma Chung University, Malang, Indonesia

Corresponding author: Paulus Lucky Tirma Irawan (e-mail: paulus.lucky@machung.ac.id)

ABSTRACT Reinforcement learning (RL) has become one of the most dynamic areas of artificial intelligence, enabling agents to learn optimal behaviour through trial-and-error interactions with their environment. Most RL studies and implementations are conducted using Python-based libraries or within the Unity engine. However, the open-source Godot Engine, which has gained significant traction among indie developers, offers an attractive alternative platform for RL research and interactive game development. This study presents the implementation of RL agents using the Advantage Actor-Critic (A2C) and Proximal Policy Optimization (PPO) algorithms on the Godot Engine through the AgentRL plugin. Two classic RL environments, Cart Pole and Cliff Walking, were replicated in Godot to evaluate algorithm performance. The AgentRL plugin facilitates real-time communication between Godot and Python, allowing model training through the Stable-Baselines3 library. Performance was measured using average episode length and average reward, and statistical analysis was conducted using Welch's t-test. Experimental results indicate that PPO outperformed A2C in both environments: in Cart Pole, PPO achieved an average of 115.93 steps longer balance duration, while in Cliff Walking, it produced 6.19 steps shorter episodes and 27.93 points higher rewards. These findings confirm that PPO offers better training stability and efficiency than A2C. Furthermore, the results demonstrate that Godot, integrated with AgentRL, is a viable and flexible platform for reinforcement learning research and can serve as a foundation for future studies in AI-driven game development.

KEYWORDS A2C, Godot Engine, PPO, Reinforcement Learning.

I. INTRODUCTION

The development of artificial intelligence (AI) technology has brought significant changes across various sectors, including the gaming industry. AI plays an important role in controlling non-player character (NPC) behaviour, managing enemies, and creating adaptive challenges for players [1]. One rapidly growing AI approach is reinforcement learning (RL), which allows agents to learn through direct interaction with the game environment with the goal of maximizing long-term cumulative rewards [2] [3+]. Unlike rule-based or deterministic AI, RL provides adaptive flexibility since the agent can explore various strategies without relying on explicit scripts. This enables a more dynamic and non-repetitive gameplay experience [4].

Among the many RL algorithms, Advantage Actor Critic (A2C) and Proximal Policy Optimization (PPO) are two of the most popular methods. A2C combines both policy-based and value-based approaches, resulting in relatively stable learning [5], while PPO introduces a safer policy update mechanism through clipping techniques to prevent extreme parameter changes [6].

Previous studies have explored the use of RL in game simulations. Juliani et al. demonstrated that Unity MLAgents can be used as a training platform for RL agents in interactive 3D games [4]. Beeching et al. later developed the AgentRL plugin for the Godot Engine, enabling integration with popular RL libraries such as StableBaselines3 [7][8]. However, research directly comparing A2C and PPO within the Godot platform remains limited.

The main problem addressed in this study is the lack of comparative analysis of A2C and PPO performance on the Godot platform using the AgentRL plugin, especially in classical simulation games such as Cart Pole and Cliff Walking [2][9][10][11]. This study is important to evaluate the feasibility of Godot as an experimental platform for reinforcement learning, while also providing insights into the relative strengths of each algorithm in different scenarios. Based on this problem, the objective of this research is to

implement the A2C and PPO algorithms in the Godot Engine using the AgentRL plugin. In addition, this study replicates and adapts two classical RL simulation environments, Cart Pole and Cliff Walking, within Godot for testing purposes. The performance of both algorithms is evaluated and compared using two metrics: average episode length and average reward obtained during training. Furthermore, the statistical significance of performance differences between the two algorithms is analysed using Welch's t-test [12].

II. METHODOLOGY

A. Reinforcement Learning

Reinforcement Learning (RL) is a branch of machine learning in which an agent learns from direct interactions with an environment to maximize its long-term cumulative reward [2]. One of the main formulations in RL is the Markov Decision Process (MDP), which consists of states, actions, transitions, and rewards.

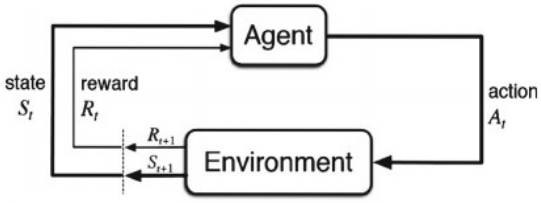


Figure 1. Reinforcement Learning Process

Figure 1 illustrates the reinforcement learning process, which involves two main components: the agent and the environment. The agent receives the current state (St) and reward (Rt) from the environment, then responds by selecting an action (At). This action affects the environment's condition, resulting in a new state (St+1) and a new reward (Rt+1). This interactive cycle continues, where the agent's main goal is to learn a policy that maximizes long-term rewards.

B. Advantage Actor Critic

Advantage Actor-Critic (A2C) is a synchronous version of the Asynchronous Advantage Actor-Critic (A3C) algorithm introduced by Mnih et al. [5]. A2C combines two major approaches in reinforcement learning: the actor (policy based) and the critic (value-based). The actor is responsible for generating actions based on a policy's probability distribution, while the critic evaluates the value of states or actions using a value function.

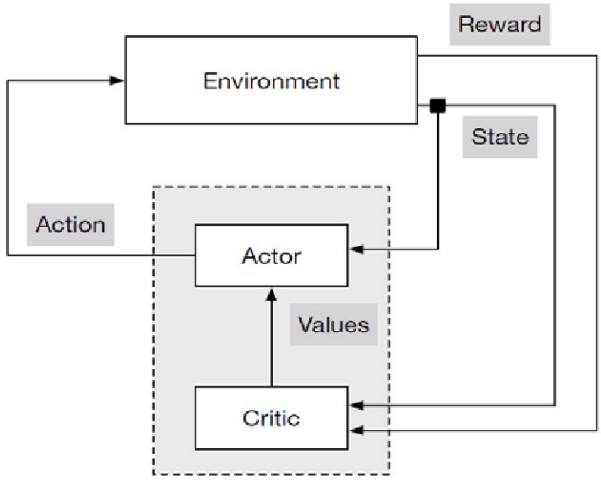


Figure 2.A2C Algorithm Architecture

Figure 2 shows the architecture of the A2C algorithm. It separates the decision-making process (actor) from the evaluation process (critic). The actor generates actions according to the policy being trained, while the critic estimates the value of each state to evaluate the chosen actions. The actor updates its policy to increase the probability of actions that lead to higher rewards, while the critic minimizes estimation errors in the value function. A2C uses the concept of "advantage" to measure how much better an action performs compared to the average, resulting in more stable and efficient policy updates.

C. Proximal Policy Optimization

Proximal Policy Optimization (PPO) is one of the most widely used policy gradient algorithms in reinforcement learning due to its stability and ease of implementation [6]. Developed by OpenAI, PPO improves upon the more computationally complex Trust Region Policy Optimization (TRPO) method.

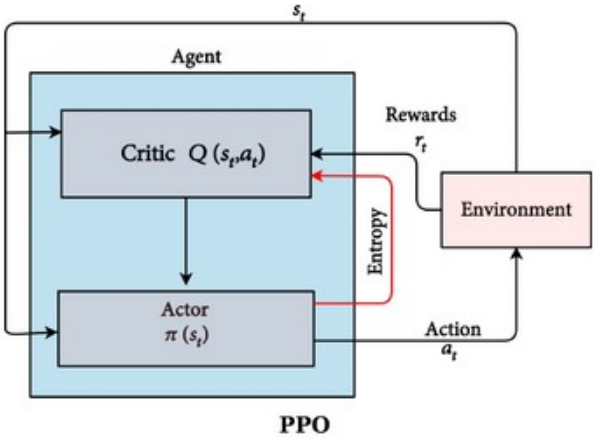


Figure 3. PPO Algorithm Architecture

Figure 3 depicts the architecture of the PPO algorithm. It maximizes a clipped objective function to ensure that policy updates between iterations are not excessively large, thus avoiding drastic performance drops. PPO collects interaction data from agents in the environment, computes the advantage values to assess the quality of actions, and updates policy parameters using stochastic gradient ascent with a clipping constraint to maintain stability.

D. StableBaseline3

Stable-Baselines3 (SB3) is a Python library built on PyTorch that provides modular and reproducible implementations of popular RL algorithms such as PPO and A2C [13]. SB3 is widely used in RL research due to its comprehensive documentation, high test coverage, and active community support.

In this study, SB3 serves as the backend for agent training, connected to the Godot Engine via the AgentRL plugin. This integration allows the training process to be conducted in Python using A2C and PPO algorithms, while the trained models are deployed in the Godot simulation environment for interactive performance evaluation.

E. Godot Engine

Godot Engine is an open-source game engine that supports both 2D and 3D game development through a modular node and scene system [14]. It supports multiple programming languages, including GDScript, C#, and Python via GDNative. Godot's popularity among indie game developers has grown rapidly due to its lightweight performance, opensource nature, and flexible customization [14].

TABLE I
PERCENTAGE OF TOTAL GAMES IDENTIFIED ON ITCH.IO

Game Engine	2018	2022	2023
Unity	47.3%	49.75%	46.33%
Construct	12.3%	13.12%	13.82%
GameMaker	11.0%	7.32%	6.95%
Twine	6.2%	5.35%	6.03%
RPG Maker	3.9%	2.74%	2.76%
Bitsy	3.3%	3.11%	3.18%
PICO-8	2.9%	2.68%	2.60%
Unreal	2.8%	2.92%	3.01%
Godot	2.5%	5.55%	7.51%
Ren'Py	2.0%	2.07%	2.07%
Other	5.9%	5.55%	5.75%

Table I shows that Unity remains the dominant engine, contributing around 47-50% from 2018 to 2023, despite a slight decline in 2023. In contrast, Godot has experienced consistent growth from 2.5% in 2018 to 7.51% in 2023, driven by its open-source and license-free characteristics, making it increasingly relevant for both indie development and academic research.

F. AgentRL Plugin

To connect Godot with SB3, this study uses the AgentRL plugin. This plugin enables real-time, bidirectional communication between Godot and Python through TCP sockets [9] 6. In Godot, the plugin sends observation and reward data from the agent to the Python server via TCP. The Python side processes the data through the RL model to determine the next action, which is then sent back to Godot to update the environment state. This process continues iteratively until the defined timestep limit is reached.

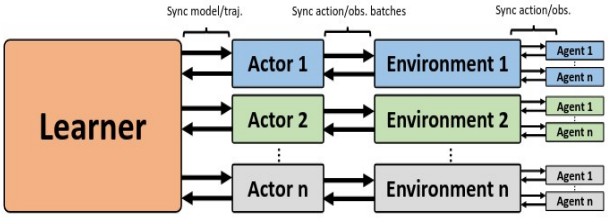


Figure 4. **AgentRL Plugin Diagram**

Figure 4 illustrates the AgentRL plugin architecture. The learner component serves as the training hub, coordinating model parameter updates. It receives trajectory batches from multiple actors, processes them to update the model, and synchronizes the new parameters across all actors. Each actor interacts with one or more environments, gathering state, action, and reward data, which are periodically sent to the learner. The synchronization between actors and agents ensures consistent training across parallel environments.

G. T-Test

The t-test is a statistical method used to compare the means of two groups and assess whether the difference is significant [15]. Variants include the independent t-test, paired t-test, and Welch's t-test, the latter being more robust against unequal variances [16]. In machine learning and reinforcement learning research, t-tests are often applied to quantitatively compare algorithm performance [17]. Welch's t-test is recommended for experimental data with unequal variances [18].

In this study, Welch's t-test is used to determine the statistical significance of performance differences between the A2C and PPO algorithms based on the metrics of average episode length (ep_len_mean) and average reward (ep_rew_mean). This approach ensures that the evaluation results are supported by strong statistical evidence.

III. RESEARCH METHOD

A. Cart Pole System Design

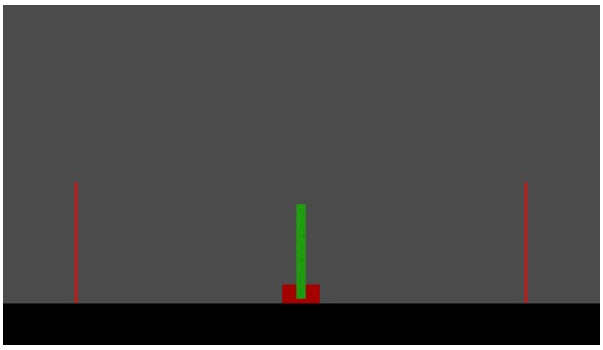


Figure 5. **Cart Pole Game**

The Cart Pole system design, as shown in Figure 5, aims to build an inverted pendulum simulation in the Godot Engine that can be used as a reinforcement learning training game. The game consists of a cart, pole, ground, and arena boundaries arranged to resemble real-world dynamics. The agent has two simple action options: pushing the cart to the left or to the right. At each step, the agent receives observation data in the form of the cart's position and velocity, the pole's angle, and the pole's angular velocity. If the agent keeps the pole balanced and does not move out of bounds, it receives a positive reward for each step. The step parameter defines the frequency of agent-environment interactions that occur prior to caching the data into the buffer for the training process. Consequently, this value dictates the buffer capacity required for calculating the advantage function and executing the policy update.

Cart (RigidBody2D): The base platform where the pole stands and the object moved horizontally by the agent.

- Width: 40 unit
- Height: 20 unit
- Mass: 1 kg
- Center Impulse Force: 5 unit

Pole (RigidBody2D): The pole that must be kept upright.

- Width: 10 unit
- Height: 100 unit Mass: 0.1 kg

PinJoint2D: Connects the cart and the pole as a pivot that allows the pole to swing freely depending on the horizontal force on the cart.

Ground (StaticBody2D): Frictionless horizontal ground with gravitational forces affects the simulation.

Arena (Node2D): Defines the cart's movement limits on the x-axis range from -240 to 240 units. If the cart exceeds this range, the simulation episode ends or fails. In the Cart Pole game, the action space is discrete and consists of two possible actions:

- 0: push cart left
- 1: push cart right

The observation space is a vector containing the following continuous values:

- Cart position (horizontal)
- Cart velocity (horizontal speed)
- Pole angle (in radians from vertical)
- Pole angular velocity (rate of angle change)

In each step, the agent receives a reward of 1 point for keeping the pole balanced and within bounds. Thus, the total reward in one episode equals the episode's length (e.g., 500 steps = 500 reward points). Hence, episode length and reward have equivalent meaning. Each episode starts with slightly randomized initial conditions near equilibrium:

- Cart position: ±5 units
- Velocity: ±5 units
- Pole angle: ±0.05 radians
- Pole angular velocity: ±0.05 radians

The episode ends if any one of these conditions reached:

- Cart position exceeds ±240 units
- Pole angle exceeds ±12° (~0.21 radians)
- 500 steps are reached (maximum duration)

B. Cliff Walking System Design

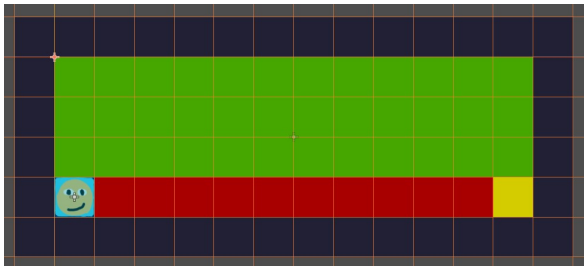


Figure 6. **Cliff Walking Game**

The Cliff Walking system design Figure 6 focuses on creating a grid-based game in the Godot Engine to test the agent's ability to make decisions under risk. The arena consists of a start tile, a finish tile, cliff tiles (large penalties), ground tiles (regular paths), and wall tiles (boundaries). The agent can move up, right, down, or left, with each move yielding different rewards or penalties. Movement and game logic use a tile map system. Ground (Tilemap): 12 columns wide and 4 rows high (coordinates x = 0-11, y = 0-3).
Key components:

- Start Tile (white): (0, 3) bottom-left corner.
- Finish Tile (yellow): (11, 3) bottom-right corner; reaching it ends the episode.
- Cliff Tiles (red): from (1, 3) to (10, 3); stepping here causes a large penalty and resets position to start.
- Ground Tiles (green): positions (0, 0) to (11, 2); normal traversable area.
- Wall Tiles (dark blue): boundaries preventing movement outside the arena.

Player (CharacterBody2D): The main agent node handling movement. The player can move in four directions but follows these rules:

- Can traverse ground, cliff, and finish tiles.
- Cannot move into wall tiles.
- Stepping on a cliff tile gives a large penalty and resets position to start.
- Reaching the finish tile resets position to start and ends the episode.

The action space has four discrete actions:

- 0: up
- 1: right
- 2: down
- 3: left

The observation space contains the player's x and y positions on the grid. The learning goal is to avoid cliffs and reach the finish with minimal penalties:

- Reward -1 for stepping on ground
- Reward -10 for attempting to move into a wall
- Reward -100 for stepping on a cliff
- Reward +100 for reaching the finish

Each episode starts at (0, 3) and ends when reaching (11, 3).

C. Evaluation

Evaluation was conducted to assess and compare the performance of A2C and PPO algorithms in the Cart Pole and Cliff Walking games. During training, two key metrics were observed:

- ep_len_mean: average episode length (how long the agent lasts)
- ep_rew_mean: average reward per episode

Training data were recorded and visualized in TensorBoard to monitor agent performance over time. A Welch's t-test with a 5% significance level ($\alpha = 0.05$) was used for statistical analysis:

If p-value < 0.05, there is a significant difference between A2C and PPO results. If p-value ≥ 0.05, there is no significant difference.

- H_0: The average results of A2C and PPO are the same (no significant difference).
- H_1: The averages differ significantly.

Five experiment runs were conducted for each algorithm per game. The results were analysed statistically, ensuring that performance differences were not only observed visually but supported by solid statistical evidence.

IV. RESULT AND DISCUSSION

A. System Implementation

The system implementation began by adding the AgentRL plugin to the Godot project through the AssetLib menu. After downloading and activating it, the plugin provides an AIController node, which connects in-game agents with the training system in Python. Within this node, several key functions were implemented:

- get_obs() to retrieve observation data from the game get_reward() to calculate reward values get_action_space() to define available actions
- set_action() to execute actions sent from the RL model.

To enable real-time data exchange between Godot and Python, a Sync node was configured in training mode. This node ensures that every frame of observation data is sent to Python and the resulting actions are received back for the next simulation step. On the Python side, agent training was managed using the Stable-Baselines3 library with A2C and PPO algorithms. All experiments were conducted within a Python virtual environment, and training data were recorded for analysis using TensorBoard.

After training was complete, the models were saved in ONNX format for use directly within Godot. The Sync node was then switched to ONNX Inference mode, allowing agents to act autonomously based on the learned policy.

TABLE II
HYPERPARAMETERS OF A2C AND PPO ALGORITHMS

Hyperparameter	Cart Pole A2C	Cart Pole PPO	Cliff Walking A2C	Cliff Walking PPO
total_timesteps	1000000	1000000	50000	50000
n_steps	5	32	32	32
learning_rate	0.0007	0.0003	0.0007	0.0003
gamma	0.99	0.99	0.99	0.99
gae_lambda	1.0	0.95	1.0	0.95
ent_coef	0.0	0.0	0.0	0.0
vf_coef	0.5	0.5	0.5	0.5
max_grad_norm	0.5	0.5	0.5	0.5
batch_size	-	64	-	64
n_epochs	-	10	-	10
clip_range	-	0.2	-	0.2

The hyperparameters used for both algorithms are listed in Table II. Most values followed the default configurations of Stable-Baselines3 and were adjusted slightly according to standard practice in Cart Pole and Cliff Walking experiments. Equal training steps were maintained to ensure fair comparison.

B. Cart Pole Experiment Results

Figure 7 shows the results of the Cart Pole experiment comparing A2C and PPO based on the average episode length metric (ep_len_mean). The results indicate that PPO consistently achieved stable increases in episode duration, approaching the maximum step limit, while A2C exhibited greater fluctuations and less consistent improvement. This trend demonstrates that PPO is more effective and stable in maintaining the pole's balance on the cart. In contrast, A2C required a longer training duration to reach optimal performance and was more prone to mid-training performance drops.

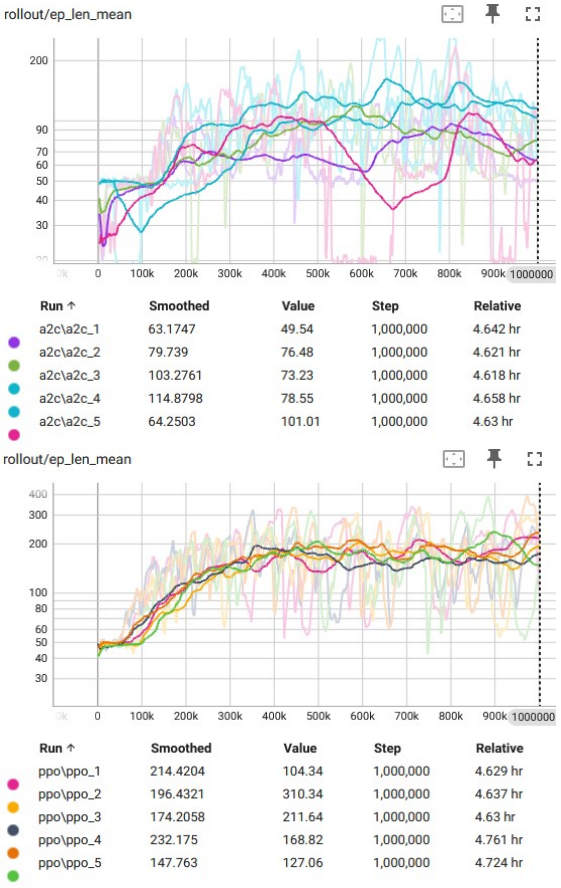


Figure 7. Cart Pole Test Results Graph

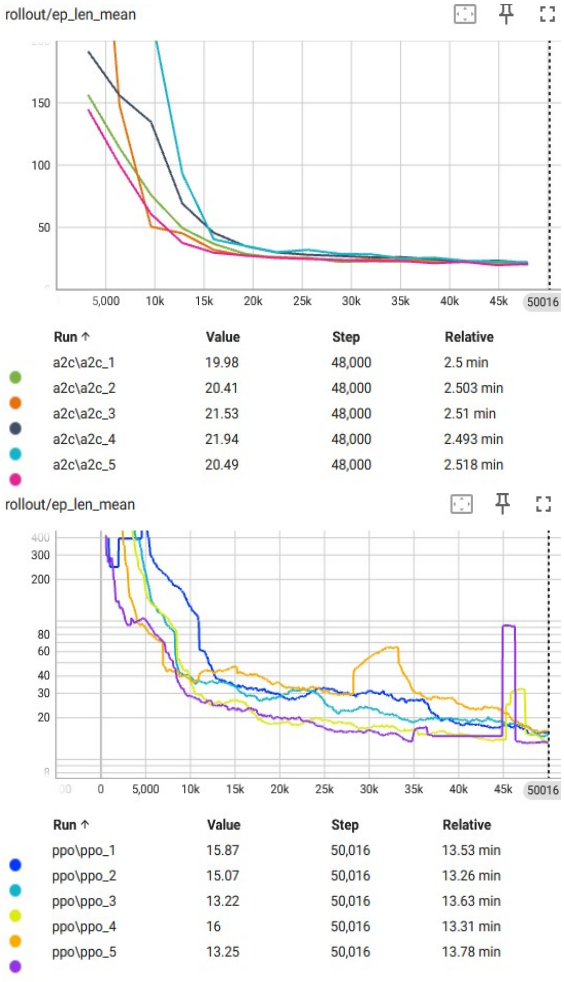
The results of the Cart Pole experiment (Figure 7) show that the PPO algorithm consistently achieved a longer average episode duration (ep_len_mean) compared to A2C. PPO displayed steady improvement and quickly approached the maximum step limit, while A2C exhibited more fluctuations and slower convergence.

TABLE III
CART POLE AVERAGE EPISODE LENGTH TEST

Run	1	2	3	4	5	Mean
A2C	50.32	88.06	69.46	85.86	103.90	79.52
PPO	126.82	305.03	220.83	199	125.59	195.45
Deviation	76.50	216.97	151.37	113.14	21.69	115.93

Table III summarizes the average episode lengths. The test result ($t = -3.35$, $p = 0.023 < 0.05$) indicates a statistically significant difference between A2C and PPO. On average, PPO balanced the pole 115.93 steps longer than A2C. This confirms that PPO is more effective and stable in learning to balance the pole, whereas A2C required a longer training period and showed occasional performance drops. These findings align with PPO's design principle of performing safer and more efficient policy updates.

C. Cliff Walking Experiment Results



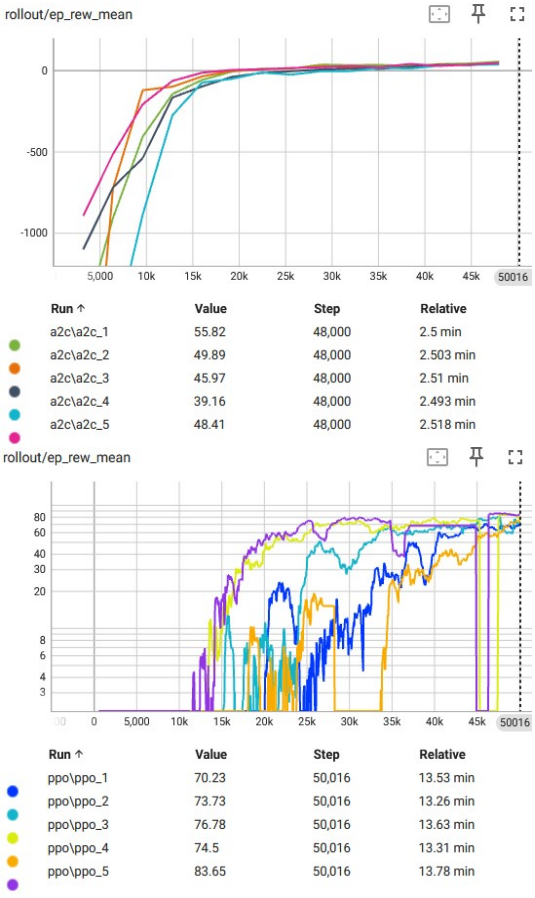


Figure 8. **Cliff Walking Test Results Graph**

```
import scipy.stats as stats

a2c = [19.98, 20.41, 21.53, 21.94, 20.49]
ppo = [15.87, 15.07, 13.22, 16, 13.25]

t_stat, p_value = stats.ttest_ind(a2c, ppo, equal_var=False)
print("CliffWalking ep_len_mean")
print("t-statistic:", t_stat)
print("p-value:", p_value)
```

```
CliffWalking ep_len_mean
t-statistic: 8.658280723363953
p-value: 7.872547278649721e-05
```

```
import scipy.stats as stats

a2c = [55.82, 49.89, 45.97, 39.16, 48.41]
ppo = [70.23, 76.78, 74.5, 83.65, 73.73]

t_stat, p_value = stats.ttest_ind(a2c, ppo, equal_var=False)
print("CliffWalking ep_rew_mean")
print("t-statistic:", t_stat)
print("p-value:", p_value)
```

```
CliffWalking ep_rew_mean
t-statistic: -7.953066781567552
p-value: 5.579634215640081e-05
```

Figure 9. **Cliff Walking T-Test Results**

The Cliff Walking experiment (Figure 8 and 9) compares A2C, and PPO based on average episode length (ep_len_mean) and average reward (ep_rew_mean). Both algorithms showed a decrease in episode duration as training progressed, but PPO demonstrated slightly better stability. In terms of rewards, PPO achieved consistently higher and more stable average rewards than A2C.

TABLE IV
CLIFF WALKING AVERAGE Episode Length TEST RESULTS

Run	1	2	3	4	5	Mean
A2C	19.98	20.41	21.53	21.94	20.49	20.87
PPO	15.87	15.07	13.22	16	13.25	14.68
Deviation	4.11	5.34	8.31	5.94	7.24	6.19

TABLE V
CLIFF WALKING AVERAGE REWARD PER EPISODE TEST RESULTS

Run	1	2	3	4	5	Mean
A2C	55.82	49.89	45.97	39.16	48.41	47.85
PPO	70.23	76.78	74.5	83.65	73.73	75.78
Deviation	14.41	26.89	28.53	44.49	25.32	27.93

From Tables IV and V, the t-test results were:

- Episode length: $t = 8.66$, $p = 0.00008$ (< 0.05)
- Average reward: $t = -7.95$, $p = 0.00005$ (< 0.05)

These values indicate significant performance differences, with PPO completing episodes about 6.19 steps faster and earning 27.93 higher average rewards than A2C. PPO effectively minimized penalties from falling into cliffs and found more optimal paths to the goal.

D. Comparative Evaluation of A2C and PPO

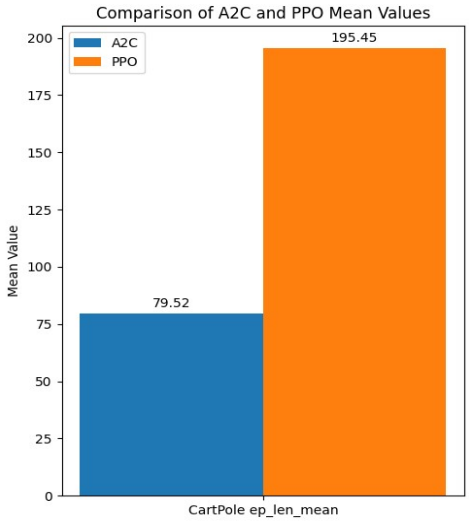


Figure 10. Cart Pole Evaluation Results

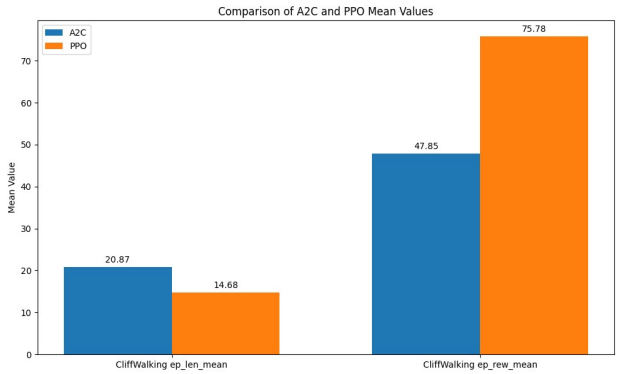


Figure 11. Cliff Walking Evaluation Results

The overall performance comparison is illustrated in Figures 10 and 11.

- In Cart Pole, PPO achieved an average episode length of 195.45, far exceeding A2C's 79.52, indicating much greater training efficiency and balance control.
- In Cliff Walking, PPO completed episodes in 14.68 steps on average, compared to 20.87 for A2C, while obtaining a mean reward of 75.78, much higher than A2C's 47.85.

These results show that PPO outperforms A2C in both environments being more stable, faster to converge, and capable of higher reward acquisition. The outcomes are consistent with prior studies [6], which found PPO superior for complex control tasks.

V. CONCLUSION

This study successfully implemented the Advantage Actor-Critic (A2C) and Proximal Policy Optimization (PPO) algorithms in two classic games, Cart Pole and Cliff Walking, using the Godot Engine integrated with Python through the AgentRL plugin. The experimental results show that in the Cart Pole game, PPO was able to maintain balance longer than A2C, with an average deviation of 115.93 additional steps. In the Cliff Walking game, PPO achieved an average episode duration that was 6.19 steps shorter and a reward deviation that was 27.93 points higher compared to A2C. Overall, PPO demonstrated better performance than A2C, especially in terms of training stability, efficiency, and reward acquisition in both games.

This research contributes to the development of the Godot Engine as an alternative platform for reinforcement learning experiments using Python via the AgentRL plugin. The comparative results between the two algorithms provide insight into their characteristics and advantages when facing different game challenges, which can serve as a reference for researchers and game developers in selecting the most suitable algorithm for RL-based agent control.

Suggestions for future research include expanding the variety of tested games, particularly those with higher complexity, to further evaluate the generalization capability of the algorithms. Additionally, hyperparameter tuning can be explored in greater depth to optimize each algorithm's performance. Integrating more efficient model formats or developing custom reward functions could also enhance training outcomes in future studies.

ACKNOWLEDGMENTS

The authors gratefully acknowledge the dedication and collaborative efforts of all research colleagues who have significantly contributed to this research. Furthermore, we would like to extend our deepest appreciation to the Informatics Engineering Study Program at Ma Chung University for their unwavering support and assistance throughout the duration of this study.

AUTHORS CONTRIBUTION

Paulus Lucky Tirma Irawan: Conceptualization, Methodology, Supervision, Validation, Original Draft Writing Preparation, Review Writing and Editing.
Marvin Adinata: Formal Analysis, Resources, Software, Visualization.
Mochamad Subianto: Supervision, Review Writing & Editing

COPYRIGHT

REFERENCES

[1] S. Millington and J. Funge, Artificial Intelligence for Games, 2nd ed. Burlington, MA, USA: CRC Press, 2009.
[2] R. S. Sutton and A. G. Barto, Reinforcement Learning: An Introduction, 2nd ed. Cambridge, MA, USA: MIT Press, 2018.
[3] D. Wang and R. Snooks, "Artificial intuitions of generative design: An approach based on reinforcement learning," in Proceedings of the 2020 DigitalFUTURES, Singapore: Springer, 2021, pp. 189–198. [Online]. Available: https://doi.org/10.1007/978-981-33-4400-6_18
[4] A. Juliani et al., "Unity: A General Platform for Intelligent Agents," arXiv preprint arXiv:1809.02627, 2018.
[5] V. Mnih et al., "Asynchronous Methods for Deep Reinforcement Learning," in Proc. 33rd Int. Conf. Machine Learning (ICML), New York, NY, USA, 2016, pp. 1928-1937.
[6] J. Schulman et al., "Proximal Policy Optimization Algorithms," arXiv preprint arXiv:1707.06347, 2017. [Online]. Available: https://arxiv.org/abs/1707.06347.
[7] E. Beeching, et al., "AgentRL: Reinforcement Learning Plugin for the Godot Engine," GitHub, 2021. [Online]. Available: https://github.com/edbeeching/godot_rl_agents
[8] E. Beeching, et al., "Godot Reinforcement Learning Agents," arXiv preprint arXiv:2112.03636, 2021. [Online]. Available: https://arxiv.org/pdf/2112.03636
[9] G. Brockman et al., "OpenAI Gym," arXiv preprint arXiv:1606.01540, 2016.

[10] M. Towers et al., "Gymnasium: A standard interface for reinforcement learning environments," arXiv preprint arXiv:2407.17032, 2024. [Online]. Available: https://doi.org/10.48550/arXiv.2407.17032
[11] L. Zhong, "Comparison of Q-learning and SARSA reinforcement learning models on Cliff Walking problem," in Proceedings of DAI-23, Atlantis Press, 2023. [Online]. Available: https://www.atlantis-press.com/proceedings/dai-23/125998063
[12] C. Colas, O. Sigaud, and P. Y. Oudeyer, "A hitchhiker's guide to statistical comparisons of reinforcement learning algorithms," arXiv preprint arXiv:1904.06979, 2022. [Online]. Available: https://arxiv.org/abs/1904.06979
[13] A. Raffin et al., "Stable-Baselines3: Reliable Reinforcement Learning Implementations," Journal of Open Source Software, vol. 6, no. 54, p. 1237, 2021.
[14] Godot Engine, "Godot Documentation," Godot Docs, 2021. [Online]. Available: https://docs.godotengine.org.
[15] W. S. Gosset, "The probable error of a mean," Biometrika, vol. 6, no. 1, pp. 1-25, 1908.
[16] B. L. Welch, "The generalization of 'Student's' problem when several different population variances are involved," Biometrika, vol. 34, no. 1-2, pp. 28-35, 1947.
[17] J. Demšar, "Statistical comparisons of classifiers over multiple data sets," Journal of Machine Learning Research, vol. 7, pp. 1-30, 2006.
[18] C. Colas, O. Sigaud, and P.-Y. Oudeyer, "A hitchhiker's guide to statistical comparisons of reinforcement learning algorithms," arXiv preprint, arXiv:1904.06979, 2019. [Online]. Available: https://arxiv.org/abs/1904.06979.