Implementasi *Plugin Agent Reinforcement Learning* Menggunakan Metode A2C dan PPO di *Platform Godot*

LAPORAN TUGAS AKHIR

Oleh:
Marvin Adinata
312110009

PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS TEKNOLOGI DAN DESAIN
UNIVERSITAS MA CHUNG
MALANG
2025

LEMBAR PENGESAHAN TUGAS AKHIR

Implementasi *Plugin Agent Reinforcement Learning* Menggunakan Metode A2C dan PPO di *Platform Godot*

Oleh:

Marvin Adinata

NIM. 312110009

dari:

PROGRAM STUDI TEKNIK INFORMATIKA

FAKULTAS TEKNOLOGI DAN DESAIN

UNIVERSITAS MA CHUNG

Telah dinyatakan lulus dalam melaksanakan Tugas Akhir sebagai syarat kelulusan dan berhak mendapatkan gelar Sarjana S.Kom.

Dosen Pembimbing 1,	Dosen Pembimbing 2,
Paulus Lucky Tirma Irawan, S.Kom., MT.	**Mochamad Subianto, S.Kom., M.Cs.**
NIP. 20100005	**NIP. 20100002**

Dekan Fakultas Teknologi dan Desain

Prof. Dr.Eng. Romy Budhi, ST., MT., M.Pd.

NIP. 20070035

KATA PENGANTAR

Puji syukur dipanjatkan kehadirat Tuhan Yang Maha Esa, yang atas restu-Nya sehingga Tugas Akhir inidapat diselesaikan dengan baik. Laporan ini berisikan hasil dari Tugas Akhir selama kurang lebih enam bulan. Pada kesempatan kali ini penulis mengucapkan terima kasih kepada seluruh pihak-pihak yang telah membantu selama proses pelaksanaan Tugas Akhir maupun dalam penyusunan laporannya, di antaranya:

1. Kedua orang tua terkasih, yang telah memberikan dukungan dan semangat selama menjalani kegiatan TA,
2. Paulus Lucky Tirma Irawan, S.Kom., MT. & Mochamad Subianto, S.Kom., M.Cs. sebagai pembimbing Tugas Akhir,
3. Bapak Dr. Eng. Romy Budhi Widodo selaku Dekan dari Fakultas Teknologi dan Desain Universitas,
4. Serta teman-teman yang telah memberikan dukungan selama proses penyelesaian kegiatan TA ini.

Laporan ini disusun berdasarkan hasil kegiatan Tugas Akhir yang dilaksanakan selama enam bulan. Tugas Akhir ini merupakan kewajiban mahasiswa Teknik Informatika Universitas Ma Chung sebagai prasyarat kelulusan.

Malang, 27 Oktober 2025

Marvin Adinata

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

BAB I
PENDAHULUAN

1.1 Latar Belakang

Perkembangan teknologikecerdasan buatan (*Artificial Intelligence* atau AI) telah mengalami kemajuan yang pesat dan semakin banyak diterapkan di berbagai bidang, termasuk dalam industri *game*. Dalam dunia *game*, AI memiliki peran yang sangat penting, seperti mengatur perilaku karakter non-pemain (*non-player character*/NPC), mengendalikan musuh, serta menciptakan tantangan yang seimbang dan menarik bagi pemain (Millington & Funge, 2016).

Salah satu pendekatan AI yang berkembang pesatadalah *Reinforcement Learning* (RL), yang merupakan salah satu cabang dari pembelajaran mesin (*machine learning*). RL memungkinkan agen untuk belajar dari pengalaman melalui interaksi dengan lingkungan, di mana agen tersebut menerima umpan balik berupa *reward* (penghargaan) atau *punishment* (hukuman) atas tindakan yang dilakukan (Sutton & Barto, 2018). Dengan mekanisme ini, agen dapat mempelajari strategi optimal untuk mencapai tujuan tertentu tanpa perlu diberikan instruksi eksplisit.

Berbeda dengan AI konvensionaldalam *game* yang umumnya berbasis pada aturan statis (*rule-based*) atau skrip yang telah diprogram sebelumnya, AI berbasis *Reinforcement Learning* bersifat adaptif dan dapat belajar dari pengalaman. AI konvensional cenderung bersifat deterministik dan mudah ditebak perilakunya,sedangkan agen RL mampu mengeksplorasi berbagai kemungkinan aksi dan menyesuaikan strateginya secara dinamis (Juliani et al., 2018). Hal ini membuat RL sangat potensial untuk menciptakan pengalaman bermain yang lebih alami, tidak repetitif, dan menantang.

Dalam bidang *reinforcementlearning*, pemilihan algoritma yang mampu belajar secara stabildan efisien dari interaksi dengan lingkungan menjadi sangat penting. Salah satu pendekatan yang cukup populer adalah *Advantage Actor-Critic* (*A2C*), yaitu versi sinkron dari *Asynchronous Advantage Actor-Critic* (*A3C*) yang diperkenalkan oleh Mnih et al. (2016). A2C merupakan pengembangan darimetode *Actor-Critic* yang menggabungkan dua pendekatan utama *policy-based* (*actor*)dan *value-based* (*critic*). Komponen *actor* bertugas

memilih aksi berdasarkan distribusi probabilitas dari suatu *policy*, sementara *critic* mengevaluasi aksi tersebut dengan menghitung nilai keunggulan (*advantage*) dari aksi terhadap nilai *state*. Nilai *advantage* ini kemudian digunakan untuk memperbaiki *policy* agar semakin optimal dari waktu ke waktu.

Proximal Policy Optimization (PPO)adalah salah satu algoritma dalam metode *Reinforcement Learning* yang dikembangkan untuk mengatasi kelemahan dari algoritma *policy gradient* konvensional yang cenderung tidak stabil saat pelatihan. PPO diperkenalkan oleh *OpenAI* dalam jurnal "*Proximal Policy Optimization Algorithms*" (Schulman et al., 2017) sebagai solusi yang lebih sederhana dan efisien dibandingkan metode seperti *Trust Region Policy Optimization* (TRPO), yang memerlukan komputasi kompleks. Inti dari PPO terletak pada penggunaan fungsi objektif dengan mekanisme *clipping*, yang membatasi perubahan kebijakan agar tidak terlalu ekstrem dari kebijakan sebelumnya. Dengan cara ini, PPO dapat menjaga stabilitas pelatihan sekaligus memungkinkan pembaruan kebijakan secara lebih fleksibel.

Untuk memfasilitasi eksperimen ini, *game CartPole* dan *CliffWalking* dikembangkan menggunakan *game engine Godot*. Proses pelatihan agen RL dilakukan melalui plugin *AgentRL* oleh (Beeching et al., 2021), yang memungkinkan integrasi antara *Godot* dan *Python* secara real-time. Dengan kombinasi ini, pelatihan agen dapat berjalan dalam lingkungan visual yang interaktif, sekaligus menggunakan *framework* model RL di *backend Python*.

Penelitian ini bertujuan untuk mengimplementasikan algoritma A2C dan PPO dalam *game CartPole* dan *Cliff Walking* menggunakan *Godot* dan *AgentRL*, serta mengevaluasi performa agen dalam menyelesaikan tugas. Harapannya, hasil dari penelitian ini dapat memberikan wawasan dalam pengembangan AIpada *game* serta menjadi pijakan untuk penelitian lebih lanjut dalam penerapan *Reinforcement Learning* menggunakan *game engine Godot* dan *plugin AgentRL*.

1.2 Identifikasi Masalah

Berdasarkan latar belakang yang telah dijelaskan, permasalahan yang diidentifikasi dalam penelitian ini antara lain:

1. Bagaimana implementasi *plugin AgentRL* di platform *Godot*?

2. Bagaimana performa algoritma A2C dan PPO sebagai agen RL di *Godot*?

1.3 Batasan Masalah

Agar penelitian ini tetap fokus dan terarah, terdapat beberapa batasan masalah yang ditetapkan, yaitu:

1. Penelitian dilakukan di *platform Godot* dengan bantuan *plugin AgentRL*
2. Pengujian algoritma dilakukan pada 2 jenis game yakni *CartPole* dan *CliffWalking*

1.4 Rumusan Masalah

Berdasarkan identifikasi dan batasan masalah yang telah dijelaskan, rumusan masalah dalam penelitian ini adalah:

1. Bagaimana hasil implementasinya *AgentRL* di *Godot*?
2. Algoritma mana yang lebih baik?

1.5 Tujuan Penelitian

Penelitian ini bertujuan untuk:

1. Cara implementasi RL di *Godot*
2. Mengevaluasi hasil implementasi RL di godot dengan algoritma A2C dan PPO

1.6 Manfaat Penelitian

Hasil dari penelitian ini diharapkan memberikan manfaat di antaranya:

1. Memperlihatkan bagaimana implementasi RL di *Godot*
2. Hasil implementasi RL di godot dapat dijadikan referensi jika ingin mengembangkan *game* dengan RL di *Godot*
3. Memberi kontribusi ilmiah dalam penerapan RL di *Godot*

1.7 Luaran

Penelitian ini diharapkan menghasilkan luaran berupa:

1. *Game CartPole* dan *CliffWalking* dengan RL
2. Publikasi ilmiah

1.8 Sistematika Penelitian

Sistematika penulisan Tugas Akhir ini dibagi menjadi lima bab seperti berikut ini.

Bab I Pendahuluan

Bab ini terdiri dari latar belakang, identifikasi masalah, rumusan masalah, Batasan masalah, tujuan penelitian, manfaat penelitian, luaran tugas akhir, dan sistematika penelitian.

Bab II Tinjauan Pustaka

Bab ini berisi urutan sistematis terkait literatur yang digunakan dalam proses penyusunan laporan Tugas Akhir sehingga diperoleh landasan teori terkait Implementasi *Plugin Agent Reinforcement Learning* Menggunakan Metode A2C dan PPO di *platform Godot*.

Bab III Analisis dan Perancangan

Bab ini menjelaskan mengenai tahapan pengerjaan serta analisis dari penelitian yang akan dilakukan. Tahapan pengerjaan meliputi alur penelitian studi literatur, hingga perancangan sistem dan evaluasi.

Bab IV Hasil dan Pembahasan

Bab ini menjelaskan mengenai hasil dari implementasi dan pengujian *Reinforcement Learning* menggunakan metode A2C dan PPO pada dua jenis permainan, yaitu *CartPole* dan *CliffWalking*.Pembahasan dilakukan berdasarkan data hasil pelatihan agen.

Bab V Kesimpulan dan Saran

Bab ini menjelaskan kesimpulan dari hasil penelitian yang telah dilakukan serta memberikan saran untuk pengembangan lebih lanjut di masa depan.

BAB II
TINJAUAN PUSTAKA

2.1 Godot Engine

Godot Engine merupakan sebuah platform pengembangan *game* sumber terbuka (open-source) yang mendukung pengembangan *game* 2D maupun 3D. *Godot* dikembangkan oleh komunitas global dengan lisensi MIT, sehingga bebas digunakan untuk keperluan komersial maupun non-komersial tanpa biaya lisensi. *Godot* menyediakan lingkungan pengembangan terpadu (IDE) yang dilengkapi dengan fitur *drag and drop*, sistem *scene* dan *node*, serta bahasa pemrograman internal yang disebut *GDScript*.

Salah satu kekuatan *Godot* adalah sistem hierarki *scene* dan *node* yang modular dan fleksibel. Pengembang dapat membuat unit-unit fungsional kecil (*node*) yang dapat dikombinasikan menjadi *scene*, dan kemudian *scene* tersebut dapat digunakan kembali dalam *scene* lainnya. Sistem ini memudahkan pengembangan aplikasi interaktif dengan struktur yang terorganisir.

Selain *GDScript*, *Godot* juga mendukung bahasa pemrograman lain seperti *C#*, dan *C++* melalui *GDNative*. Fitur *cross-platform export* memungkinkan *game* yang dikembangkan di *Godot* dapat dijalankan di berbagai platform seperti *Windows, Linux, Android, iOS*, dan *Web HTML5*.

Kemampuan *Godot* dalam berintegrasi dengan sistem eksternal seperti *Python, ONNX*, maupun komunikasi *TCP/UDP* menjadikannya *platform* yang fleksibel untuk implementasi *Reinforcement Learning*. Dalam penelitian berbasis pembelajaran agen, *Godot* dapat digunakan sebagai lingkungan simulasi tempat agen melakukan interaksi dan belajar dari hasil aksinya. Ketersediaan dokumentasi yang terus berkembang dan dukungan komunitas *open-source* juga mempermudah proses pengembangan dan eksperimen akademik. Oleh karena itu, *Godot Engine* dinilai relevan dan potensial untuk digunakan sebagai basis pengembangan simulasi interaktif dalam penelitian *reinforcement learning* berbasis *game*.

Tabel 2.1 Persentase Total Game yang Teridentifikasi di 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%

Tabel 2.1 diambil dari "*On the relevance of the Godot Engine in the indie game development industry*" (Holfeld, 2023) menunjukkan persentase penggunaan berbagai *game engine* berdasarkan total jumlah *game* yang diunggah di platform *itch.io* pada tahun 2018, 2022, dan 2023. Data ini dikumpulkan pada periode 30 September 2022 hingga 21 Oktober 2023 dan memberikan gambaran tren penggunaan *engine* dalam komunitas pengembang *game* independen.

Dari data tersebut, *Unity* masih menjadiengine paling dominan,dengan kontribusi sebesar 47.3% pada 2018 dan meningkat signifikan sebesar 49.75% pada 2022, meskipun kemudian mengalami penurunan menjadi 46.33% pada 2023. *Construct* juga menunjukkan tren yang menarik, yaitu peningkatan sebesar 13.12% pada 2022, namun turun menjadi 13.82% pada 2023. *GameMaker* dan *Twine* mengalamipenurunan dan kenaikan bergantian,sedangkan *engine* seperti *RPG Maker, Unreal*, dan *Ren'Py* menunjukkan perubahan yang relatif kecil dari tahun ke tahun.

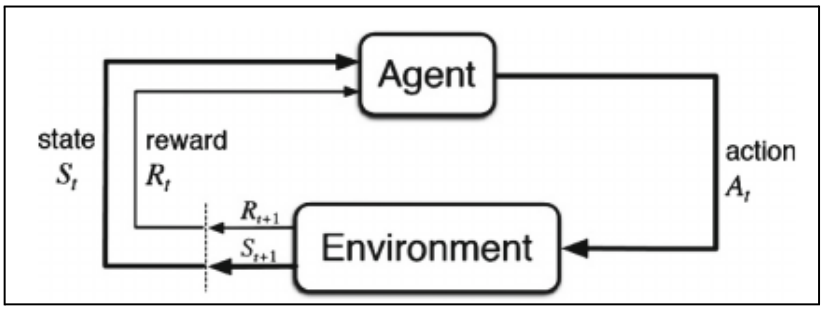
Penggunaan *Godot* mengalami pertumbuhan yang positif dan konsisten: dari 2.5% di tahun 2018 meningkat 5.55% di 2022 dan naik lagi menjadi 7.51% di 2023. Hal ini menunjukkan bahwa *Godot* semakin populer, kemungkinan karena sifatnya yang open-source, ringan, dan tanpa biaya lisensi, sehingga menarik bagi pengembang *game indie*. Peningkatan ini juga menunjukkan relevansi dan potensi *Godot* sebagai alternatif yang layak bagi engine komersial dalam pengembangan *game*, termasuk untuk kebutuhan eksperimen akademik atau penelitian.

Dengan kata lain, tren ini mencerminkan pergeseran preferensi komunitas terhadap *game engine* yang lebih terbuka dan dapat diakses secara gratis seperti *Godot*, terutama di tengah meningkatnya kekhawatiran terhadap biaya dan kebijakan lisensi dari *engine* komersial.

2.2 Reinforcement Learning

Reinforcement Learning (RL) merupakan salah satu pendekatan dalam pembelajaran mesin di mana agen belajar melalui interaksi langsung dengan lingkungan untuk memaksimalkan *reward* kumulatif jangka panjang (Sutton & Barto, 2018). Dalam *RL*, agen mengambil tindakan berdasarkan kondisi lingkungan, kemudian menerima umpan balik dalam bentuk *reward* dan mengubah kebijakannya berdasarkan pengalaman tersebut. Pendekatan RL telah diaplikasikan secara luas dalam berbagai permasalahan, termasuk pengendalian sistem dinamis seperti robotika dan *game*. Salah satu formulasi yang umum digunakan dalam *RL* adalah *Markov Decision Process* (*MDP*), yang terdiri atas *state*, *action*, *transition*, dan *reward*.

Dalam eksperimen yang kompleks, penggunaan *RL* sering dikombinasikan dengan pembelajaran mendalam (*deep learning*), menghasilkan pendekatan *Deep Reinforcement Learning* (*DRL*). Mnih et al. (2016) memperkenalkan metode *asynchronous* yang mampu mempercepat dan menstabilkan pembelajaran agen melalui penggunaan banyak *thread* pelatihan secara paralel.



Gambar 2.1 Diagram Reinforcement Learning

Gambar 2.1 merupakan arsitektur dasar dari proses *Reinforcement Learning* (RL) yang diambildari "*Artificial Intuitions of Generative Design: An Approach Based on Reinforcement Learning"* (Wang et al., 2021) yang merupakan kerangka kerja pembelajaran di mana agen belajar mengambil tindakan dalam suatu lingkungan untuk memaksimalkan total *reward* kumulatif (Sutton & Barto, 2018).

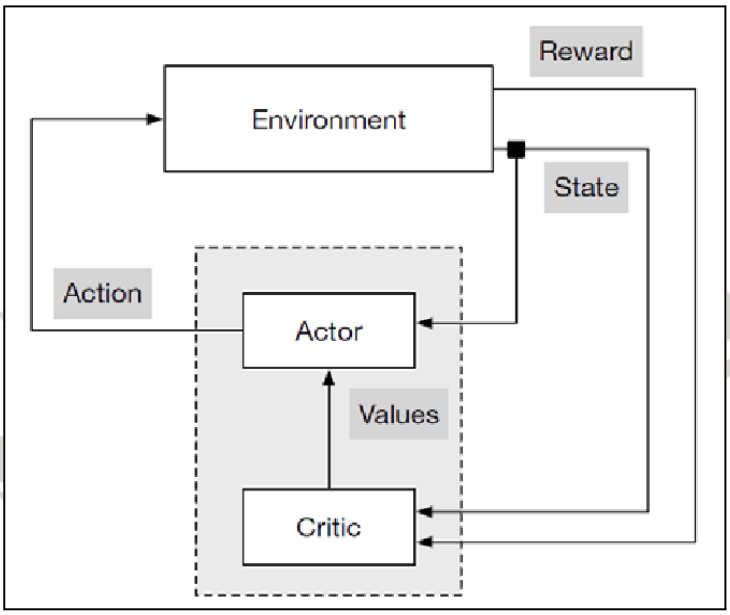
Dalam skema ini, terdapat dua komponen utama, yaitu *agent* dan *environment*. *Agent* menerima *state* saat ini (St) dan *reward* (Rt) dari environment, kemudian merespons dengan memilih suatu *action* (At). Aksi tersebut memengaruhi kondisi *environment* yang kemudian menghasilkan *state* berikutnya (St+1) dan *reward* baru (Rt+1). Siklus ini berlangsung berulang dalam proses interaktif antara *agent* dan *environment*, di mana tujuan utama *agent* adalah mempelajari kebijakan yang memaksimalkan *reward* jangka panjang.

Gambar 2.1 menekankan bahwa proses RL adalah bentuk pembelajaran berbasis pengalaman langsung, berbeda dengan pembelajaran *supervised learning* yang mengandalkan data berlabel. Dengan pendekatan ini, *agent* dapat mengembangkan perilaku optimal secara bertahap melalui eksplorasi dan evaluasi hasil dari setiap aksi yang diambil.

2.3 Advantage Actor Critic

Advantage Actor Critic (A2C) merupakan pengembangan dari metode *Actor-Critic* yang menggabungkan keunggulan dari dua pendekatan utama dalam *RL*, yaitu *policy-based* dan *value-based*. Kelebihan utama dari A2C adalah kemampuannya dalam mengurangi varians dan mempercepat konvergensi melalui

penggunaan *advantage function*, yaitu selisih antara estimasi nilai sebenarnya dan estimasi nilai rata-rata. Metode ini juga menggunakan pendekatan sinkron dalam pelatihan, yang membedakannya dari *Asynchronous Advantage Actor-Critic* (A3C) yang bersifat *asynchronous*. Kinerja A2C dievaluasi berdasarkan akumulasi reward rata-rata per episode.



Gambar 2.2 Arsitektur Algoritma A2C

Gambar 2.2 merupakan representasi arsitektur algoritma *Advantage Actor-Critic* (A2C) yang diambil dari "*Reinforcement Learning: A Technical Introduction – Part I*" (Ramstedt, 2019). Dalam algoritma A2C, terdapat dua komponen utama, yaitu *Actor* dan *Critic*. *Actor* bertugas untuk memilih aksi berdasarkan kondisi lingkungan saat ini (*state*), sedangkan *Critic* bertugas untuk mengevaluasi aksi yang diambil *Actor* dengan memberikan nilai (*value*) dari suatu kondisi.

Proses dimulai ketika lingkungan (*environment*) memberikan kondisi (*state*) kepada agen. Berdasarkan kondisi ini, *Actor* memilih aksi yang akan diambil. Aksi tersebut kemudian dikirimkan kembali ke lingkungan yang akan merespons dengan kondisi baru dan *reward*. *Reward* ini digunakan oleh *Critic* untuk memperbarui estimasi nilai kondisi, yang kemudian dikirim kembali ke *Actor* sebagai dasar pembelajaran dalam memilih aksi berikutnya.

Gambar 2.2 menggambarkan bagaimana interaksi antara *Actor* dan *Critic* terjadi secara simultan dan saling memperkuat satu sama lain dalam proses pembelajaran.Pendekatan ini memberikan kestabilan dalam pembaruan kebijakan dan efisiensi dalam pembelajaran karena pemisahan peran eksplorasi (*Actor*) dan evaluasi (*Critic*), serta penggunaan keunggulan (*advantage*) sebagai sinyal pelatihan untuk mengurangi variasi.

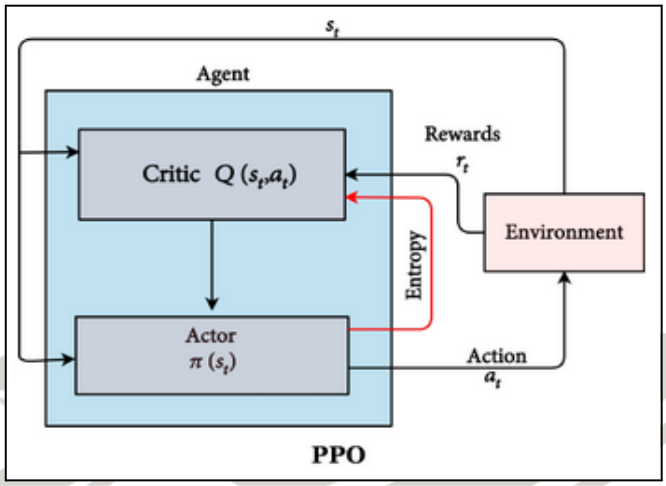
2.4 Proximal Policy Optimization

Proximal Policy Optimization (PPO) adalah algoritma *RL* yang dikembangkan oleh Schulman et al. (2017) untuk mengatasi instabilitas yang sering muncul pada algoritma *policy gradient*. PPO menggunakan teknik *clipped surrogate objective*, yang membatasi perubahan drastis pada kebijakan selama proses pelatihan, sehingga menghasilkan pembelajaran yang lebih stabil dan andal. PPO saat ini menjadi salah satu algoritma paling populer di kalangan praktisi karena keseimbangan antara performa, stabilitas, dan kemudahan implementasi.Selain itu, PPO banyak digunakan dalam berbagai *platform* seperti *OpenAI Gym*, *Unity ML-Agents*, dan juga diadaptasi dalam *plugin Godot AgentRL*. Kalkulasi performa PPO mencakup reward rata-rata, stabilitas pembelajaran (dilihat dari grafik reward terhadap episode), dan waktu konvergensi. Berdasarkan hasil dari Schulman et al. (2017), PPO menunjukkan hasil superior dalam berbagai lingkungan, termasuk Atari dan Mujoco, karena stabil dan mudah diimplementasikan.

Gambar 2.3 memperlihatkan arsitektur dari algoritma Proximal Policy Optimization (PPO) yang diambil dari "*On Improving Cross-dataset Generalization of Deepfake Detectors"* (Nadimpalli et al., 2022), yang merupakan salah satu algoritma *policy gradient* paling stabil dan efisien dalam *reinforcement learning*. PPO merupakan pengembangan dari metode *Actor-Critic* yang dirancang untuk memperbaiki kebijakan secara stabil tanpa memerlukan algoritma kompleks seperti *Trust Region Policy Optimization* (TRPO) (Schulman et al., 2017).

Dalam gambat 2.3, agen terdiri dari dua komponen utama, yaitu *Actor* dan *Critic*. *Actor* bertugas menghasilkan kebijakan $\pi(st)$ yang digunakan untuk

memilih aksi at berdasarkan keadaan saatini st. Sementara itu, *Critic* berfungsi untuk mengevaluasi seberapa baik aksi tersebut melalui estimasi fungsi nilai Q(st, at). Lingkungan memberikan umpan balik berupa *reward* (rt) dan keadaan baru (st+1) setelah aksi dilakukan.



Gambar 2.3 Arsitektur Algoritma PPO

PPO menggunakan mekanisme pembaruan kebijakan yang menghindari perubahan drastis (*over-updating*) melalui kliping (*clipping*) fungsi objektif. Gambar 2.3 juga menampilkan entropi (*entropy*) sebagai bagian dari sinyal pembelajaran tambahan untuk menjaga eksplorasi kebijakan agar tidak terlalu deterministik. Pendekatan ini menghasilkan pembelajaran yang lebih stabil dan efisien dibanding metode pendahulunya.

2.5 StableBaseline3

Stable-Baselines3 (SB3) merupakan kerangka kerja (*framework*) *open-source* yang menyediakan implementasi algoritma *reinforcementlearning* (RL) yang stabil dan dapat direproduksi. *Framework* ini ditulis dalam bahasa *Python* dan dibangun di atas *PyTorch* untuk memudahkan proses pelatihan dan evaluasi agen RL secara konsisten. Raffin et al. (2021) menyebutkan bahwa SB3 hadir sebagai perbaikan dari proyek sebelumnya, *Stable-Baselines*, dengan mengedepankan modularitas, dokumentasi yang lebih lengkap, dan integrasi yang

baik dengan ekosistem RL modern. *Framework* ini mendukung berbagai algoritma populer seperti *Proximal Policy Optimization* (PPO)dan *Advantage Actor-Critic* (A2C). Keandalan SB3 juga diperkuat dengan serangkaian pengujian unit dan eksperimen yang sistematis, menjadikannya sebagaisalah satu acuan utama dalam penelitian maupun pengembangan aplikasi berbasis RL. Kemudahan penggunaan, fleksibilitas dalam konfigurasi *hyperparameter*,serta komunitas aktif menjadikan SB3 sangat relevan untuk digunakan baik oleh peneliti maupun praktisi dalam mengembangkan agen RL yang efisien dan dapat dikontrol.

Tabel *2.2* diambil dari "*Stable-Baselines3: Reliable Reinforcement Learning Implementations*" (Raffin et al., 2021). *Stable-Baselines3* (SB3) menunjukkan performa yang unggul dibandingkan pustaka *reinforcement learning* lainnya seperti *OpenAI Baselines, PFRL, RLlib, Tianshou, Acme*, dan *Tensorforce*. Berdasarkan data perbandingan dari Raffin et al. (2021), SB3 dibangun menggunakan *PyTorch*, yang saat ini menjadi backend populer dalam pengembangan model machine learning. SB3 memiliki dokumentasi API dan panduan pengguna yang lengkap, yang menjadi aspek penting dalam kemudahan pemakaian oleh peneliti maupun praktisi. Fitur *benchmarking* dan *pretrained models* juga tersedia, memperkuatposisinya sebagaipustaka RL yang aplikatif dan siap pakai.

Dalam hal kualitas kode, SB3 memiliki tingkat cakupan pengujian (*test coverage*) hingga 95%, jauh di atas beberapa pustaka lain seperti *OpenAI Baselines* (49%) atau *Acme* (74%). SB3 juga menyediakan sistem pemeriksaan tipe (*type checking*) dan *template* untuk pelaporan isu atau pengajuan perubahan kode (*pull request*), yang menandakan pengelolaan proyek yang profesional. Selain itu, frekuensi pembaruan proyek SB3 cukup tinggi, dengan komit terbaru selalu dalam rentang kurang dari satu minggu, menunjukkan komunitas yang aktif. Dalam enam bulan terakhir, SB3 juga mencatatkan 75 *pull request* yang disetujui, jauh melampaui pustaka seperti *PFRL* (13) atau *Acme* (5).

Tabel 2.2 Perbedaan SB3 Dengan Framework Lainnya

	SB3	OAI Baselines	PFRL	RLlib	Tianshou	Acme	Tensorforce

Backend	PyTorch	TF	PyTorch	PyTorch /TF	PyTorch	Jax/TF	TF
User Guide / Tutorials	✓✓	–	✓✓	✓✓	✓✓	✓✓	✓
API Documentation	✓	✗	✓	✓	–	✓	✓
Benchmark	✓	✗	✓	✓	✓	✓	✗
Pretrained models	✓	✗	✗	✓	✗	✓	✗
Test Coverage	95%	49%	?	?	94%	74%	81%
Type Checking	✓	✗	✓	✓	✓	✓	✓
Issue / PR Template	✓	✗	✗	✓	✓	✓	✓
Last Commit (age)	< 1 week	> 6 months	< 1 month	< 1 week	< 1 week	< 1 week	< 1 month
Approved PRs (6 mo.)	75	0	13	222	85	5	7

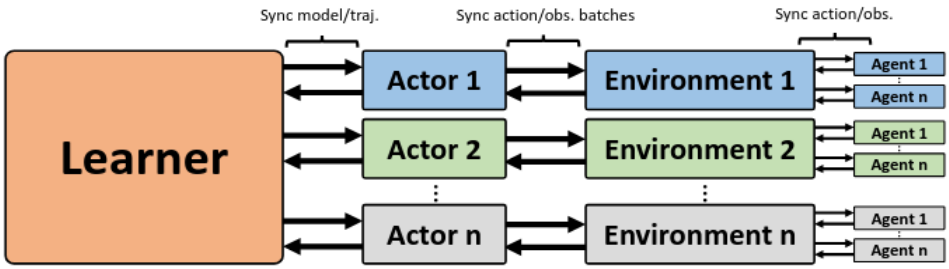
Secara keseluruhan, SB3 menonjol sebagai pustaka RL yang matang, terdokumentasidengan baik, serta secara aktif dikembangkan dan didukung oleh komunitas. Kelebihan-kelebihan ini menjadikannya pilihan utama dalam penelitian dan pengembangan sistem *reinforcement learning*.

2.6 AgentRL Plugin

AgentRL merupakan *plugin open-source* yang memungkinkan pengguna mengintegrasikan algoritma *reinforcement learning* ke dalam *game engine Godot* (Torrado et al., 2021). *Plugin* ini mendukung berbagai jenis lingkungan simulasi dan kompatibel dengan algoritma seperti *PPO* dan *A2C* melalui antarmuka *Python TCP* yang terhubung dengan *backend* seperti *Stable-Baselines3*.

Plugin AgentRL di *Godot* bekerja dengan cara mengirimkan data observasi dan *reward* dari agen ke *server Python* melalui koneksi TCP. Di sisi *Python*, data

tersebut diterima, diekstrak, lalu diproses oleh model untuk menentukan aksi selanjutnya. Hasil aksi ini kemudian dikirim kembali ke *Godot* melalui TCP, diekstrak, dan digunakan untuk memperbarui kondisi *environment* di Godot sehingga episode dapat berlanjut, proses ini berulang hingga batas *timesteps* tercapai.



Gambar 2.4 Diagram Plugin AgentRL

Gambar 2.4 menggambarkan arsitektur paralel dalam *AgentRL* yang diambil dari (Beeching et al., 2021). Arsitektur ini mengadopsi pendekatan terdistribusi untuk meningkatkan efisiensi pelatihan dalam Reinforcement Learning dengan memisahkan komponen utama ke dalam tiga bagian utama: *Learner*, *Actor*, dan *Environment*.

1. **Learner**

   Learner merupakan komponen pusat yang bertugas mengelola proses pelatihan. Ia menerima *batch* trajektori (*trajectory batches*) dari para *actor*, melakukan pembaruan model, dan menyinkronkan parameter terbaru ke semua *actor*. Ini memungkinkan proses pelatihan yang efisien dan terpusat.

2. **Actors**

   Setiap *actor* menerima salinan model dari *learner* dan menjalankan proses interaksi dengan lingkungan secara paralel. Mereka mengumpulkan data interaksi (*state, action, reward*) dari lingkungan dan mengirimkannya kembali ke *learner* dalam bentuk *batch*. Proses ini mempercepat eksplorasi ruang aksi karena dilaksanakan secara simultan oleh banyak *actor*.

3. **Environments dan Agents**

   Setiap *actor* terhubung dengan satu atau lebih *environment*, yang di

dalamnya terdapat satu atau lebih *agent*. Para agen inilah yang melakukan aksi dalam lingkungan berdasarkan kebijakan yang ditentukan oleh model. Sinkronisasi antara *actor* dan *agent* mencakup pertukaran observasi dan aksi.

2.7 Uji T

Uji t adalah metode statistik yang digunakan untuk membandingkan rata-rata dua kelompok, guna menentukan apakah perbedaan tersebut signifikan secara statistik atau hanya terjadi karena variasi acak. Uji t pertama kali diperkenalkan oleh William Sealy Gosset pada tahun 1908 dengan nama pena "Student", sehingga dikenal juga sebagai Student's t-test (Gosset, 1908). Menurut Montgomery dan Runger (2014), terdapat beberapa varian uji t:

- Independent t-test: Membandingkan rata-rata dua kelompok independen.
- Paired t-test: Digunakan jika data berpasangan (misalnya sebelum dan sesudah perlakuan).
- Welch's t-test: Versi modifikasi independent t-test yang tidak mengasumsikan varians kedua kelompok sama.

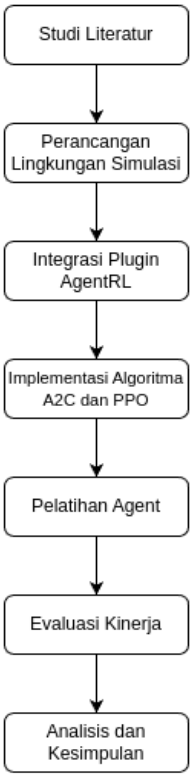
Welch (1947) memperkenalkan metode ini untuk mengatasi perbedaan varians antar sampel sehingga lebih fleksibel digunakan dalam data eksperimen yang heterogen. Seperti dijelaskan oleh Ruxton (2006), uji t memiliki beberapa asumsi penting:

- Data berskala interval atau rasio.
- Distribusi mendekati normal (untuk ukuran sampel kecil).
- Observasi antar sampel independen.

Namun, Welch's t-test dianggap lebih robust terhadap pelanggaran asumsi kesamaan varians, sehingga banyak digunakan dalam eksperimen komputer, termasuk pembelajaran mesin dan RL. Demšar (2006) dalam Statistical Comparisons of Classifiers over Multiple Data Sets menyebutkan bahwa uji t banyak digunakan untuk membandingkan performa algoritma pembelajaran mesin, meskipun harus hati-hati terhadap kesalahan tipe I jika dilakukan berulang kali. Dalam RL, Colas et al. (2019) juga merekomendasikan uji t atau alternatif non-parametrik untuk membandingkan metode.

BAB III
ANALISIS DAN PERANCANGAN SISTEM

3.1 Alur Penelitian



Gambar 3.1 Flowchart Alur Penelitian

Penelitian ini dilakukan dengan beberapa tahapan sistematis yang bertujuan untuk mengimplementasikan dan menguji algoritma *Advantage Actor-Critic* (A2C) dan *Proximal Policy Optimization* (PPO) menggunakan plugin *AgentRL* di platform Godot Engine. Adapun tahapan alur penelitian secara garis besar dijelaskan sebagai berikut:

1. **Studi Literatur**

   Tahap awal dimulai dengan mengkaji berbagai referensi dan literatur yang relevan mengenai reinforcement learning, algoritma A2C dan PPO, game CartPole dan CliffWalking, serta plugin AgentRL di Godot Engine.

Sumber yang digunakan antara lain dari Sutton & Barto (2018), Schulman et al. (2017), dan Torrado et al. (2021).

2. **Perancangan Lingkungan Simulasi**

   Lingkungan permainan (game environment) disiapkan menggunakan Godot Engine. Dua lingkungan yang digunakan adalah CartPole dan CliffWalking, masing-masing dirancang menggunakan node seperti CharacterBody2D, StaticBody2D, Camera2D, dan skrip AI yang mengontrol agen.

3. **Integrasi Plugin AgentRL**

   Plugin AgentRL yang dikembangkan oleh Torrado et al. (2021) diintegrasikan ke dalam Godot. Plugin ini memungkinkan komunikasi dua arah antara Godot dan Python melalui protokol TCP, sehingga model reinforcement learning dapat dilatih secara eksternal.

4. **Implementasi Algoritma A2C dan PPO**

   Algoritma A2C dan PPO diimplementasikan menggunakan Stable-Baselines3 (Raffin et al., 2021), yang menyediakan pustaka RL berbasis PyTorch. Model dilatih dengan data observasi dan reward dari lingkungan simulasi Godot, yang dikirim melalui plugin AgentRL.

5. **Pelatihan Agen**

   Proses pelatihan dilakukan dengan menjalankan simulasi secara berulang hingga agen berhasil belajar menyelesaikan tugasnya. Pelatihan dilakukan untuk kedua game CartPole dan CliffWalking dengan menggunakan kedua algoritma A2C dan PPO. Parameter pelatihan seperti learning rate, total timesteps, dan n_steps disesuaikan untuk setiap eksperimen.

6. **Evaluasi Kinerja**

   Setelah model RL dilatih, dilakukan evaluasi berdasarkan metrik seperti total reward per episode. Hasil dari algoritma A2C dan PPO dibandingkan untuk masing-masing game untuk mengetahui efektivitas masing-masing metode.

7. **Analisis dan Kesimpulan**

   Dari hasil evaluasi, dilakukan analisis untuk menentukan keunggulan masing-masing algoritma dalam menyelesaikan dua jenis lingkungan.

Penelitian ini ditutup dengan kesimpulan mengenai efektivitas penggunaan A2C dan PPO di platform Godot menggunakan plugin AgentRL serta saran untuk penelitian selanjutnya.

3.2 Studi Literatur

3.2.1 Reinforcement Learning (RL)

ReinforcementLearning (RL) merupakan metode pembelajaran mesin di mana agen belajar mengambil keputusan melalui interaksi dengan lingkungan. Agen bertindak berdasarkan *state* yang diamatinya,memilih aksi tertentu, dan menerima *reward* sebagai umpan balik (Sutton & Barto, 2018). Tujuan agen adalah memaksimalkan akumulasireward jangka panjang dengan membentuk kebijakan (*policy*) yang optimal.

Dalam konteks pengembangan *game* dan simulasi, RL banyak diterapkan pada lingkungan interaktif seperti *CartPole* dan *CliffWalking* karena kedua kasus tersebut merepresentasikan tantangan kontrol dan eksplorasi (Juliani et al., 2018; Millington & Funge, 2009).

3.2.2 Algoritma Advantage Actor-Critic (A2C)

A2C adalah algoritma pengembangan dari metode *Actor-Critic* yang bekerja secara sinkron. Algoritma ini memisahkan proses pengambilan keputusan (*policy network*) dan evaluasi nilai (*value network*), namun keduanya dilatih bersamaan. Keunggulan A2C dibanding *Actor-Critic* biasa adalah stabilitas dan efisiensi pembelajaran yang lebih baik karena pendekatan *advantage*, yaitu selisih antara estimasinilai tindakan aktualdan rata-rata nilai semua tindakan (Mnih et al., 2016).

Dalam lingkungan *CartPole*, A2C mampu belajar menjaga keseimbangan tiang dengan cepat melalui pembaruan berbasis *reward* yang diberikan setiap langkah. Sementara itu, pada *CliffWalking*, A2C menghadapi tantangan besar karena tingginya penalti dari jurang, namun dengan pendekatan *advantage* dan pembaruan sinkron, agen tetap dapat membentuk strategi optimal secara bertahap.

3.2.3 Algoritma Proximal Policy Optimization (PPO)

PPO merupakan salah satu algoritma *policy gradient* paling populer karena stabilitas dan efisiensinya dalam pelatihan. PPO memperbarui kebijakan dengan membatasi perubahan dari kebijakan sebelumnya, sehingga mencegah perubahan drastis yang dapat menyebabkan kerugian performa. Hal ini dilakukan melalui fungsi klip (*clipping function*) dalam optimasi kebijakan (Schulman et al., 2017).

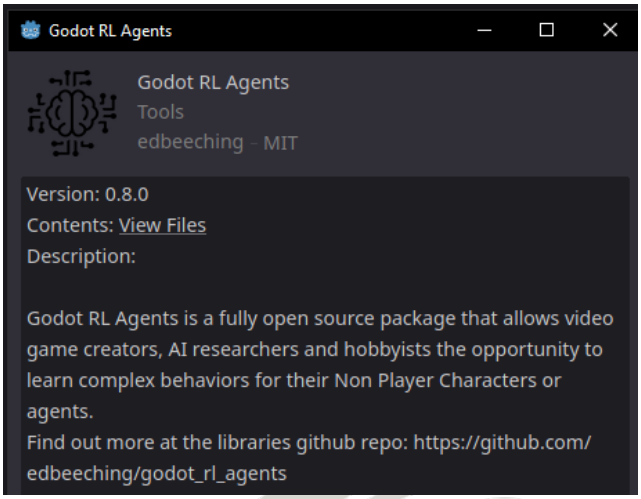
Dalam percobaan pada *CartPole*, PPO dikenal mampu mencapai performa optimal dalam waktu pelatihan yang relatif singkat. Untuk *CliffWalking*, PPO lebih tahan terhadap jebakan reward negatif karena kemampuannya mengontrol perubahan kebijakan, memungkinkan agen menghindari *cliff* sambil tetap mencapai tujuan (Torrado et al., 2021).

3.2.4 Uji T

Uji t adalah salah satu metode statistik yang paling umum digunakan untuk menentukan apakah perbedaan rata-rata dua kelompok signifikan secara statistik. Metode ini diperkenalkan oleh Gosset (1908) dan memiliki beberapa varian seperti independent t-test, paired t-test, dan Welch's t-test (Welch, 1947) yang tidak memerlukan asumsi kesamaan varians. Uji ini banyak diterapkan dalam penelitian berbasis eksperimen, termasuk pembandingan algoritma pembelajaran mesin dan reinforcement learning (Demšar, 2006; Colas et al., 2019). Untuk memastikan validitas, uji t memerlukan asumsi independensi data dan distribusi mendekati normal, meskipun Welch's t-test lebih toleran terhadap perbedaan varians (Ruxton, 2006).

3.2.5 AgentRL Plugin di Godot

Plugin AgentRL untuk Godot dapat di menyediakan kerangka kerja modular berbasis *Python* dan TCP yang memungkinkan integrasi algoritma RL eksternal seperti A2C dan PPO dengan lingkungan simulasi *Godot* (Torrado et al., 2021). *Plugin* ini memisahkan sisi pelatihan di *Python* dan simulasi di *Godot Engine*, serta memungkinkan eksperimen menggunakan visualisasi *real-time* dan desain level yang fleksibel.



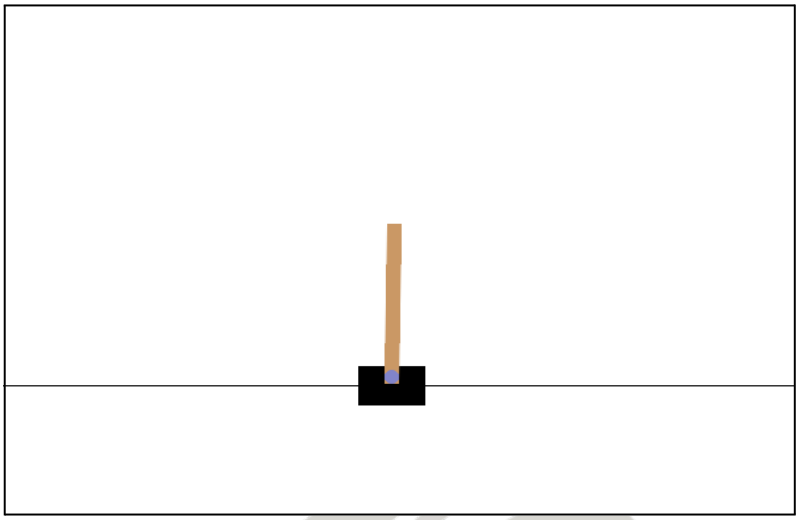
Gambar 3.2 Plugin AgentRL di Godot

Lingkungan *CartPole* dapat direplikasi di *Godot* menggunakan objek fisika 2D, dengan *feedback* berupa posisi tiang dan kecepatan kereta dikirim ke *Python* melalui TCP. Demikian pula, *CliffWalking* dapat dikonstruksi menggunakan *tile map* sederhana, di mana posisi agen, aksi, dan *reward* dikontrol secara sinkron antara *Godot* dan *Python*.

Dengan menggunakan *plugin AgentRL*, pelatihan A2C dan PPO dapat dimonitor secara interaktif, sekaligus menjaga fleksibilitas pengembangan visual di dalam *Godot Engine*. Hal ini menjadikan AgentRL sebagai solusi ideal untuk pengujian dan visualisasi algoritma RL dalam proyek berbasis *game engine*.

3.2.6 CartPole

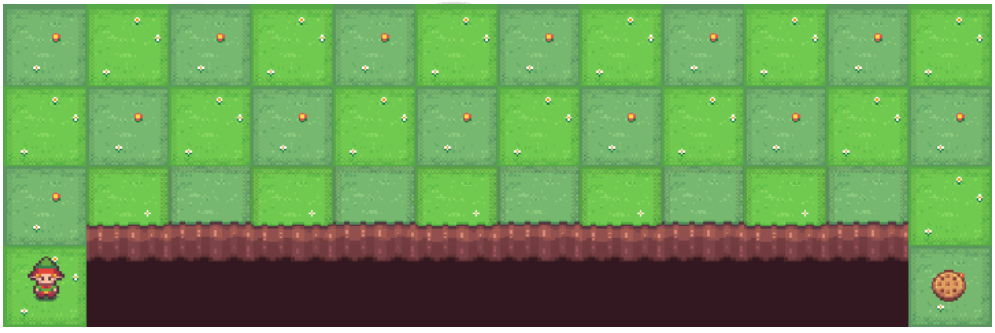
CartPole merupakan salah satu lingkungan klasik dalam *reinforcement learning*. Lingkungan ini sering digunakan sebagai *benchmark* karena memiliki dinamika fisika sederhana, tetapi cukup menantang untuk menguji algoritma penguatan. Dalam *CartPole*, tugas agen adalah menjaga tiang (*pole*) tetap seimbang di atas kereta (*cart*) dengan cara memberikan gaya ke kiri atau ke kanan. Lingkungan ini termasuk dalam kategori *continuous control problem* yang di diskretisasi ke dalam dua aksi.



Gambar 3.3 Environment Game CartPole

Menurut Sutton dan Barto (2018), lingkungan seperti *CartPole* sangat cocok digunakan untuk menguji algoritma *policy gradient* seperti A2C, dan PPO, karena lingkungan ini menyediakan umpan balik langsung terhadap perubahan kebijakan. Beberapa studi seperti Schulman et al. (2017) juga menggunakan *CartPole* sebagai bagian dari eksperimen awal saat memperkenalkan PPO, untuk menunjukkan efisiensi konvergensi dan stabilitas algoritma.

3.2.7 CliffWalking



Gambar 3.4 Environment Game CliffWalking

CliffWalking merupakan lingkungan diskrit berbasis *grid* yang diperkenalkan dalam buku "*Reinforcement Learning: An Introduction*" oleh Sutton dan Barto (2018). Lingkungan ini menyerupai papan *grid* di mana agen harus mencapai titik tujuan sambil menghindari "jurang" di sisi bawah lintasan.

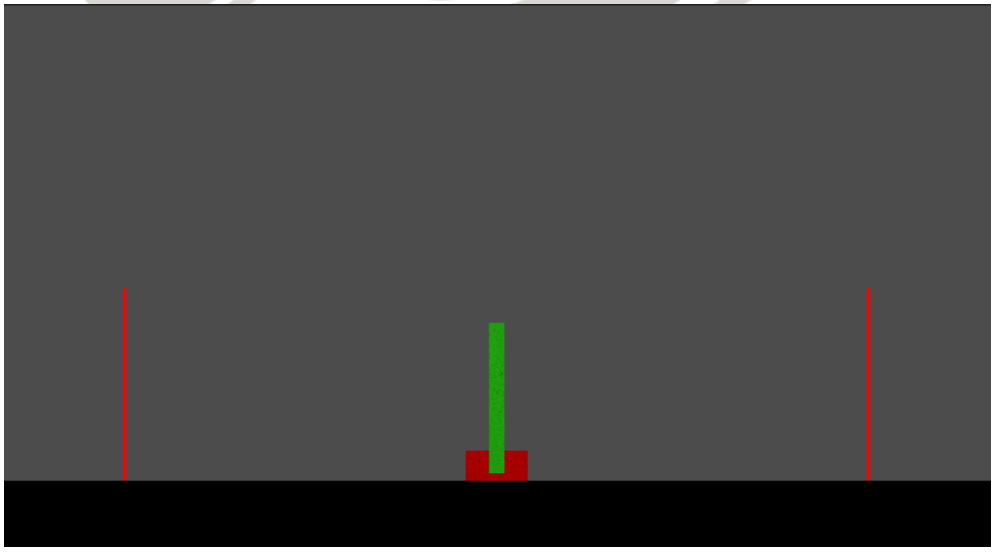
Jika agen jatuh ke jurang, maka akan mendapatkan penalti besar dan episode akan diulang dari titik awal.

Lingkungan ini sering digunakan untuk menekankan pentingnya exploration dengan exploitation serta untuk menguji efektivitas algoritma berbasis nilai seperti *Q-learning* dan *SARSA*.Namun, *CliffWalking* juga dapat digunakan untuk algoritma *policy-based* seperti A2C dan PPO, khususnya untuk melihat bagaimana agen belajar menghindari jebakan melalui optimasi kebijakan langsung.

Lingkungan *CliffWalking* memberikan tantangan unik karena adanya *sparse reward* dan penalti besar, yang membuat proses pelatihan lebih sulit dibanding *CartPole*. Ini menjadikannya pilihan yang baik untuk menguji ketahanan dan efisiensi dari algoritma yang diuji.

3.3 Perancangan Sistem

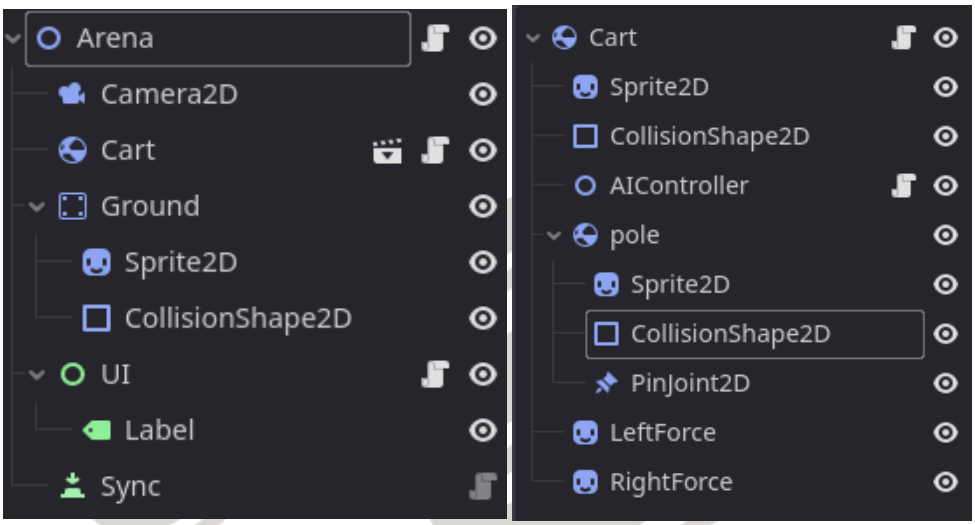
3.3.1 *Game CartPole*



Gambar 3.5 Game CartPole di Godot

CartPole adalah simulasi klasik dalam *reinforcement learning* yang merepresentasikan masalah pengendalian sistem dinamis.Dalam permainan ini, sebuah tiang vertikal (*pole*) dipasang pada sebuah kereta (*cart*) yang dapat bergerak ke kiri atau kanan pada lintasan datar.Agen yang mengendalikan *cart*

harus memberikan aksiyang tepat, yaitu mendorong *cart*ke kiri atau ke kanan, untuk menjaga keseimbangan tiang. Jika tiang jatuh terlalu miring hingga melebihi batas sudut tertentu, atau jika *cart* bergerak terlalu jauh dari pusat lintasan, maka episode akan berakhir.



Gambar 3.6 SceneTree Game CartPole

Dalam perancangan sistem *game CartPole* menggunakan *Godot Engine*, digunakan sebuah *node* utama yang dinamakan arena. Arena ini berfungsi sebagai wadah utama yang menampung seluruh elemen yang terlibat dalam simulasi. Di dalam arena terdapat beberapa elemen penting, antara lain kamera yang digunakan untuk mengikuti pergerakan *cart*, serta sebuah *scene* bernama *Cart* yang berisi objek-objek utama dalam simulasi.

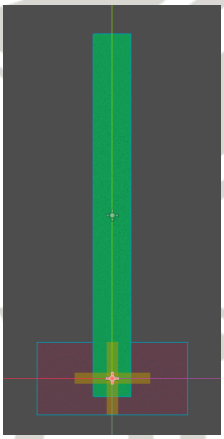
Scene *Cart* ini terdiri dari tiga komponen utama, yaitu *cart* (*RigidBody2D*), pole (*RigidBody2D*), dan sebuah sambungan engsel (*pinjoint2D*) yang menghubungkan keduanya. *Cart* dan *pole* adalah objek yang bergerak secara fisik, sementara *pinjoint* memungkinkan *pole* untuk berayun secara bebas di atas *cart*, menyerupai sistem pendulum terbalik. Selain itu, terdapat juga permukaan *Ground* sebagai landasan tempat *cart* bergerak. Permukaan ini bersifat statis dan tidak memiliki gesekan, sesuai dengan asumsi dasar dari simulasi *CartPole* klasik.

Untuk menampilkan informasi selama simulasi, sistem dilengkapi dengan antarmuka pengguna (UI) yang menampilkan nilai-nilai penting seperti posisi

cart, sudut *pole*, kecepatan, dan *steps*. Informasi ini berguna dalam proses pengamatan dan evaluasi performa agen selama pelatihan.

Sebagai komponen pengendali, terdapat sebuah *node* agen yang bertugas memberikan aksi pada sistem berdasarkan hasil perhitungan dari algoritma *reinforcement learning*. Agen ini berinteraksi langsung dengan lingkungan dan menerima masukan berupa kondisi sistem yang sedang berlangsung.

Agar komunikasi antara *Godot* dan sistem eksternal berbasis *Python* dapat berjalan, ditambahkan sebuah node sinkronisasi yaitu *Sync*. Node ini bertugas untuk mengirim dan menerima data observasi serta aksi melalui koneksi jaringan *TCP*. Dengan adanya sinkronisasi ini, model yang dilatih di *Python* dapat digunakan untuk mengontrol pergerakan *cart* secara langsung di dalam simulasi *Godot*.



Gambar 3.7 Objek Cart dan Pole

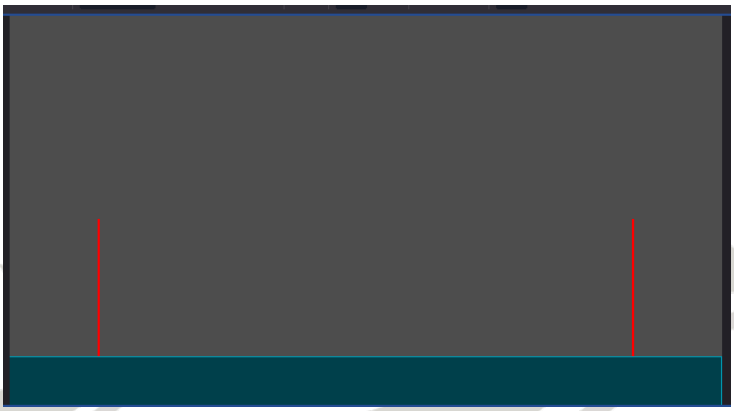
Cart (*RigidBody2D*) *platform* dasar tempat *pole* berdiri dan objek yang digerakkan oleh agen secara horizontal.

- Lebar: 40 unit
- Tinggi: 20 unit
- Massa: 1 kg
- *Center Impulse Force*: 5 unit

Pole (*RigidBody2D*) sebagai *pole* yang harus dijaga agar tetap tegak oleh sistem.

- Lebar: 10 unit
- Tinggi: 100 unit
- Massa: 0.1 kg

Pin Joint (*PinJoint2D*) menghubungkan *cart* dan *pole* sebagai sumbu rotasi yang memungkinkan *pole* berayun bebas tergantung gaya horizontal pada *cart*.



Gambar 3.8 Objek Ground dan Batas Arena

Ground (StaticBody2D) memastikan hanya gaya horizontal dan gaya gravitasi yang berpengaruh dalam simulasi. Tidak memiliki gesekan (*frictionless*). Arena (Node2D) Menentukan batas ruang gerak *cart* dari rentang posisi sumbu x: dari -240 unit hingga 240 unit. Jika *cart* melewati batas ini, maka episode simulasi dinyatakan gagal atau berakhir.

```
func get_action_space() -> Dictionary:
    return {
        "move_action" : {"size": 2, "action_type": "discrete"},
    }
```

Gambar 3.9 Code Action Space CartPole

Pada permainan *CartPole*, *action space* bersifat diskrit dan terdiri dari dua kemungkinan aksi. Dengan kata lain, agen hanya bisa memilih antara dua tindakan setiap step.

- 0 → mendorong kereta ke kiri
- 1 → mendorong kereta ke kanan

```
func get_obs() -> Dictionary:
    var obs: Array = [
        _player.global_position.x,
        _player.linear_velocity.x,
        _player.pole.global_rotation,
        _player.pole.angular_velocity,
    ]

    return {"obs": obs}
```

Gambar 3.10 Code Action Space CliffWalking

Observation space adalah vektor berisi informasi dari lingkungan yang diamati oleh agen. Dalam *CartPole*, terdapat empat nilai kontinu yang membentuk observasi:

1. Posisi kereta (*cart position*): posisi horizontal dari *cart* (unit)
2. Kecepatan kereta (*cart velocity*): laju horizontal *cart* (unit)
3. Sudut tiang (*pole angle*): kemiringan tiang terhadap vertikal (dalam radian)
4. Kecepatan sudut tiang (*pole angular velocity*): laju perubahan sudut tiang

Pada *game CartPole*,agen menerima *reward* sebesar 1 poin untuk setiap langkah yang berhasil dilalui tanpa menjatuhkan tiang atau keluar dari batas lintasan. Artinya, *reward* diberikan secara konstan setiap langkah hingga episode berakhir. Karena reward selalu bernilai 1 per langkah, maka total reward yang diperoleh agen dalam satu episode akan sama dengan panjang episode tersebut. Misalnya, jika agen bertahan selama 500 langkah, maka total reward-nya juga adalah 500 poin. Dengan demikian, nilai *ep_rew_mean* akan sama dengan *ep_len_mean*.

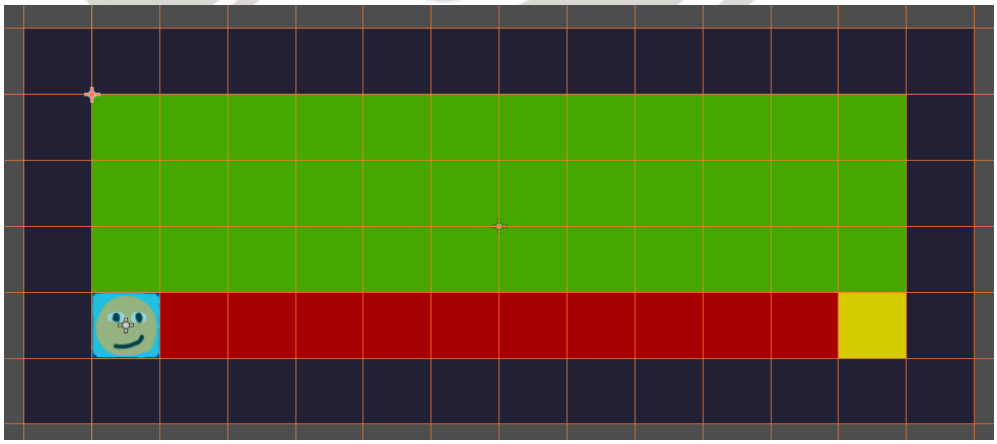
Setiap episode dimulai dengan kondisi awal yang diacak dalam rentang kecil di sekitaran keadaan seimbang. Artinya sistem memulai dalam posisi hampir seimbang, namun dengan sedikit gangguan acak yang harus dikendalikan agen.

- Posisi kereta (*cart position*) ± 5 unit
- Kecepatan (*cart velocity*) ± 5 unit
- Sudut tiang (*pole angle*) ± 0.05 radians
- Kecepatan sudut (*pole angular velocity*) ± 0.05 radians

Episode dalam CartPole akan berakhir apabila terjadi salah satu dari kondisi berikut:

1. Posisi kereta keluar dari batas: ± 240 unit dari pusat.
2. Sudut tiang melebihi ± 12° (sekitar ± 0.21 radians).
3. Jumlah langkah telah mencapai 500 step yang berarti agen berhasil menjaga keseimbangan dalam waktu maksimal.

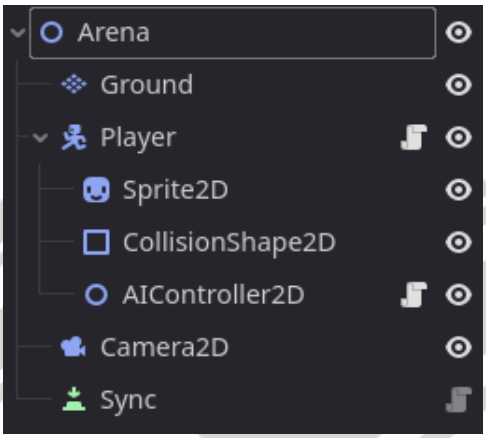
3.3.2 *Game CliffWalking*



Gambar 3.11 Game CliffWalking di Godot

CliffWalking adalah lingkungan *gridworld* 2D berukuran 4x12 yang digunakan untuk menguji kemampuan agen dalam eksplorasi dan pengambilan keputusan di bawah risiko. Agen memulai dari titik (3, 0) dan bertujuan mencapai titik (3, 11). Di antara titik start dan goal terdapat area berbahaya yang disebut jurang (cliff), yaitu sel sepanjang tepi bawah *grid* dari (3, 1) hingga (3, 10). Aksi yang tersedia adalah bergerak ke atas, bawah, kiri, atau kanan.

CliffWalking merupakan permainan simulasi *reinforcement learning* berbasis *grid* yang terdiri dari sebuah arena berukuran 12x4. Arena ini terbagi menjadi beberapa jenis tile yang memiliki peran dan aturan tertentu dalam permainan. Tujuan utama dari game ini adalah agar pemain (agen) dapat bergerak dari titik start menuju titik finish sambil menghindari *tile* berbahaya yang disebut cliff.



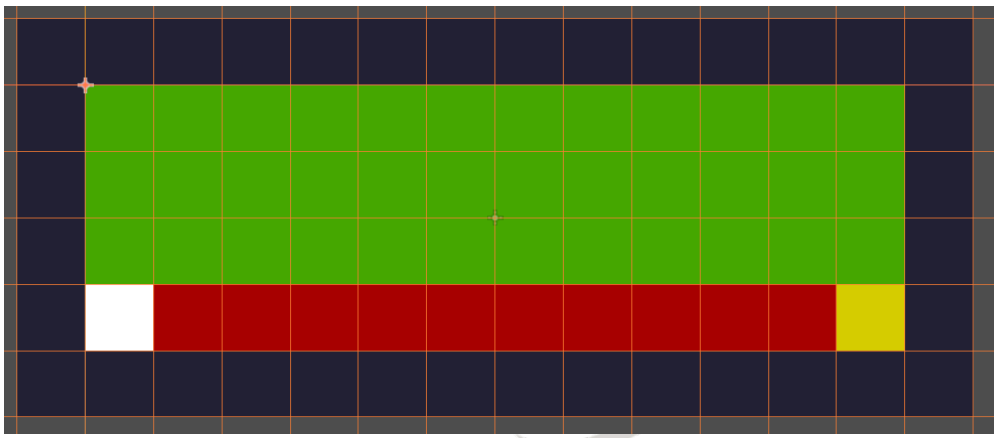
Gambar 3.12 Scene Tree CliffWalking

Gambar 3.4.2.2 menunjukkan struktur *Scene Tree* dari *game CliffWalking* yang dibuat menggunakan *Godot Engine* dalam mode 2D. Struktur ini menggambarkan susunan node yang merepresentasikan elemen-elemen dalam satu skena permainan. *Node* utama dari *scene* ini adalah *Node2D Arena*, yang berfungsi sebagai wadah atau *root* dari seluruh elemen dalam skena. Di dalam *Arena*, terdapat beberapa node anak:

Ground (Tilemap) terdiri atas 12 kolom (lebar) dan 4 baris (tinggi), dengan sistem koordinat posisi berbentuk (x, y), di mana x adalah kolom (0–11) dan y adalah baris (0–3). Komponen-komponen penting pada arena ini meliputi:

- *Tile Start* (putih): Titik awal pemain berada di posisi (0, 3), yaitu pojok kiri bawah arena.
- *Tile Finish* (kuning): Titik tujuan berada di posisi (11, 3), yaitu pojok kanan bawah arena. Jika pemain memasuki *tile* ini, agen dianggap berhasil mencapai *finish* dan episode berakhir.

- *Tile Cliff* (merah): Rentetan jurang berada disepanjang baris bawah dari (1, 3) hingga (10, 3). Jika pemain memasuki *tile* ini, agen akan dianggap jatuh ke jurang dan posisinya akan di-reset ke tile start.
- *Tile Ground* (hijau): Semua posisi dari (0, 0) hingga (11, 2) merupakan area *ground* biasa yang dapat dilalui.
- *Tile Wall* (biru tua): Dinding pembatas mengelilingiseluruh arena agar pemain tidak dapat bergerak keluar dari batas arena.



Gambar 3.13 Tilemap Arena CliffWalking

Player (CharacterBody2D)adalah node utama untuk karakter pemain merupakan CharacterBody2D yang menangani pergerakan dan fisika pemain. Pemain atau agen memiliki kemampuan untuk bergerak ke empat arah utama, yaitu atas, bawah, kiri, dan kanan. Namun, gerakannya dibatasi oleh beberapa aturan:

- Pemain dapat melintasi *tile ground*, *cliff,* dan finish.
- Pemain tidak dapat bergerak ke *tile wall* (pembatas arena).
- Ketika pemain melangkah ke *tile cliff*, permainan akan memberikan penalti besar, dan posisi pemain akan di-reset ke tile start.
- Ketika pemain mencapai *tile finish*, permainan juga akan di-*reset* ke *start*, menandai akhir dari satu episode.

Gambar 3.14 Player dan Agen CliffWalking

Game ini dirancang untuk menguji kemampuan agen dalam belajar kebijakan optimal untuk mencapai tujuan dengan risiko minimum, terutama menghindari *tile cliff* yang memberikan konsekuensi buruk.

```
func get_action_space() -> Dictionary:
    return {
        "movement": {"size": 4, "action_type": "discrete"},
    }
```

Gambar 3.15 Code Action Space CliffWalking

CliffWalking memiliki 4 aksi diskrit, aksi-aksi ini memungkinkan agen untuk bergerak ke empat arah utama dalam *grid*.

- 0→atas (*up*)
- 1→kanan (*right*)
- 2→bawah (*down*)
- 3→kiri (*left*)

Observation space adalah vektor berisi informasi dari lingkungan yang diamati oleh agen. Dalam *CliffWalking*, terdapat empat nilai kontinu yang membentuk observasi:

1. Posisi player di pada *grid* di sumbu x
2. Posisi player di pada *grid* di sumbu y

Tujuan pembelajaran adalah menghindari *cliff* sambil mencapai *finish* dengan jumlah penalti seminimal mungkin.

- Agen mendapatkan *reward* -1 setiap berada di *tile ground*.
- Agen mendapatkan *reward* -10 setiap mencoba melangkah ke *tile wall*.
- Agen mendapatkan *reward* -100 jika jatuh ke *tile cliff*.

- Agen mendapatkan *reward* 100 jika berada di *tile finish*.

Agen selalu memulai dari pojok kiri bawah *grid*, yaitu posisi (0, 3). Episode berakhir jika agen mencapai *tile finish*, yaitu pojok kanan bawah grid (11, 3).

```
func get_obs() -> Dictionary:
    var observations: Array = [
        player.current_position_in_tile.x,
        player.current_position_in_tile.y,
    ]
    return {"obs": observations}
```

Gambar 3.16 Kode Observation Space CliffWalking

3.4 Evaluasi

Evaluasi dalam penelitian ini menggunakan pendekatan kuantitatif, dengan tujuan untuk membandingkan performa dua algoritma *reinforcement learning*, yaitu *Advantage Actor-Critic (A2C)* dan *Proximal Policy Optimization (PPO)*. Pengukuran dilakukan berdasarkan hasil pelatihan agen pada dua jenis lingkungan permainan, yaitu *CartPole* dan *CliffWalking*.

Selama proses pelatihan, metrik performa akan dicatat dan divisualisasikan menggunakan *TensorBoard*. Visualisasi ini memudahkan analisis terhadap stabilitas, konvergensi, dan efisiensi pelatihan algoritma. Adapun dua metrik utama yang digunakan dalam evaluasi adalah sebagai berikut:

- *ep_len_mean*

  Menunjukkan rata-rata panjang episode (dalam jumlah langkah) selama pelatihan. Nilai ini memberikan gambaran seberapa lama agen mampu bertahan dalam lingkungan sebelum episode berakhir.
- *ep_rew_mean*

  Menunjukkan rata-rata reward yang diterima agen per episode. Metrik ini menjadi indikator utama dalam menilai efektivitas kebijakan yang dipelajari oleh agen. Nilai reward yang lebih tinggi mengindikasikan kebijakan yang lebih optimal.

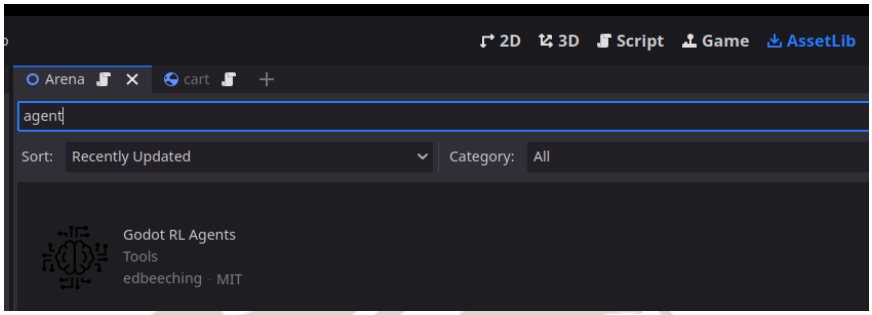
Evaluasi dilakukan dengan menggunakan ujit dua sampel (*independent t-test*) untuk mengetahui apakah terdapat perbedaan yang signifikan secara statistik antara performa algoritma A2C dan PPO. Uji t dipilih karena kedua algoritma menghasilkan data numerik yang bersifat independen dan berasal dari dua kelompok berbeda. Dalam proses ini, ditentukan dua hipotesis:

- Hipotesis nol (H_0): Tidak terdapat perbedaan signifikan antara performa A2C dan PPO.
- Hipotesis alternatif (H_1): Terdapat perbedaan signifikan antara performa A2C dan PPO.

Lima hasil pelatihan (*run*) diambil dari masing-masing algoritma sebagai data uji. Pengujian dilakukan menggunakan *Python* dengan pendekatan *Welch's t-test* untuk mengakomodasi kemungkinan perbedaan varians antar kelompok data. Setelah pengujian dilakukan, hasil dianalisis untuk menentukan apakah perbedaan performa yang diamati bersifat signifikan secara statistik atau tidak.

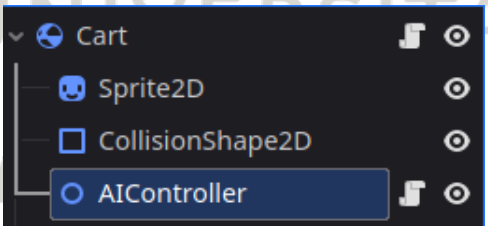
BAB IV
HASIL DAN PEMBAHASAN

4.1 Implementasi Sistem



Gambar 4.1 Menu AssetLib di Godot

Plugin AgentRL bisa diinstal langsung melalui*AssetLib* bawaan *Godot*, sehingga pengguna tidak perlu mengunduh file secara manual. Cukup buka menu *AssetLib* di dalam *editor*, cari *AgentRL*, lalu klik unduh dan pasang. Setelah terpasang, *plugin* ini akan muncul di daftar *plugin* proyek dan bisa diaktifkan melalui menu *Project Settings* → *Plugins*.



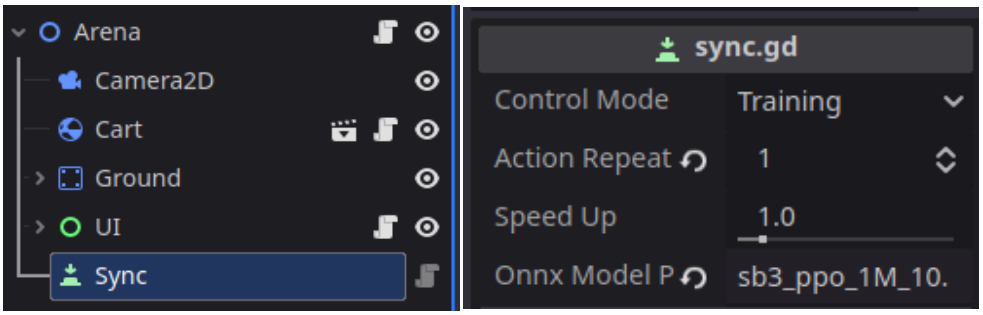
Gambar 4.2 Implementasi Node AIController

Setelah plugin diinstal, *node AIController* dapat ditambahkan ke agen yang ingin digerakkan. *Node* ini berfungsi sebagai penghubung antara agen dengan node *Sync* agar agen dapat menerima aksi dari *Python* dan menggunakannya untuk bergerak sesuai *action space*.

```gdscript
class_name AIController
extends AIController2D

var move_action: int = 0
@onready var cart: Cart = get_parent()
@onready var pole: RigidBody2D = $"../Pole"

func get_obs() -> Dictionary:
    var obs: Array = [
        _player.global_position.x,
        _player.linear_velocity.x,
        _player.pole.global_rotation,
        _player.pole.angular_velocity,
    ]

    return {"obs": obs}

func get_reward() -> float:
    return reward

func get_action_space() -> Dictionary:
    return {
        "move_action" : {"size": 2, "action_type": "discrete"},
    }

func set_action(action) -> void:
    #print(_player.episode, " : ", action)
    move_action = action["move_action"]
```

Gambar 4.3 Potongan Kode AIController

Setelah itu, *script* pada *node AIController* perlu di-*extend* untuk mengimplementasikan beberapa fungsi antarmuka yang dibutuhkan, yaitu *get_obs()*, *get_reward()*, *get_action_space()*, dan *set_action()*. Fungsi-fungsi ini bertanggung jawab untuk menyediakan data observasi dari environment, mendapatkan *reward*, mendefinisikan ruang aksi yang tersedia, serta menerapkan aksi yang dikirimkan oleh model pembelajaran ke agen di dalam *game*.



Gambar 4.4 Konfigurasi Node Sync

Node Sync ditambahkan ke dalam *scene* yang digunakan sebagai arena pelatihan agen. Pada *node* ini, pengaturan *Control*Mode *harus* diubah menjadi *Training* agar proses pelatihan dapat berjalan. Selain itu, nilai *Action Repeat* diset ke 1, yang berarti Godot akan mengirim dan menerima data dengan *Python* melalui koneksi TCP setiap satu *frame*.

Langkah berikutnya adalah melakukan konfigurasi *Python* di *VSCode* dengan membuat *virtual environment* terlebih dahulu. Setelah *environment* aktif, instal paket *AgentRL* beserta *framework Stable Baselines3* (SB3) dan *TensorFlow* untuk kebutuhan visualisasi data menggunakan perintah "*pip install godot-rl[sb3*]". Selanjutnya, instal pustaka *Microsoft.ML.OnnxRuntime* untuk mendukung proses inferensi model ONNX dengan perintah "*dotnet add package Microsoft.ML.OnnxRuntime*"

Setelah semua paket terpasang, buat kode *Python* untuk menjalankan proses pelatihan model. Hasil pelatihan dapat divisualisasikan menggunakan *TensorBoard* dengan menjalankan "*tensorboard --logdir ./logs/sb3*". Ketika pelatihan selesai, model yang dihasilkan dapat langsung digunakan di *Godot*. Dengan mengubah *Control Mode* pada *Node Sync* menjadi ONNX *Inference* dan menetapkan path ke file model ONNX yang dihasilkan.

Tabel 4.1 Hyperparameter Algoritma A2C dan PPO

Hyperparameter	CartPole A2C	CartPole PPO	CliffWalking A2C	CliffWalking 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

Tabel 4.1.1 adalah *hyperparameter* yang digunakan dalam pelatihan agen dengan algoritma A2C dan PPO pada game *CartPole* dan *CliffWalking*.

1. **total_timesteps**

   Menentukan jumlah total langkah interaksi agen dengan lingkungan selama pelatihan. Nilai ini menunjukkan seberapa lama agen belajar. Semakin besar nilainya,semakin banyak pengalaman yang dikumpulkan agen, yang biasanya menghasilkan performa yang lebih baik, namun juga membutuhkan waktu komputasi lebih lama. Dalam grafik pelatihan, sumbu-x biasanya menunjukkan progres hingga mencapai *total timesteps* tersebut.

2. **n_steps**

   Menentukan jumlah langkah interaksiagen dengan lingkungan sebelum data disimpan ke dalam buffer untuk proses pelatihan. Nilai ini berpengaruh terhadap ukuran buffer yang akan digunakan untuk menghitung nilai *advantage* dan melakukan pembaruan kebijakan.

3. **learning_rate**

   Menentukan seberapa cepat model memperbarui bobotnya berdasarkan *error* yang didapat. Nilai ini cukup umum digunakan karena memberikan keseimbangan antara stabilitas dan kecepatan belajar.

4. **batch_size**

   Ukuran data yang digunakan dalam satu kali pembaruan gradien.*Batch* yang lebih besar bisa membuat pembaruan lebih stabil,sementara *batch* yang lebih kecil mempercepat iterasi tapi meningkatkan variansi.

5. **n_epochs**

   Jumlah pengulangan proses optimisasi untuk setiap *batch* data. Nilai yang lebih besar memungkinkan model lebih maksimal memanfaatkan data yang dikumpulkan, namun terlalu tinggi bisa menyebabkan *overfitting* terhadap data yang sama.

6. **gamma**

   Faktor diskon untuk menghitung *reward* masa depan. Nilai ini menandakan bahwa agen tetap mempertimbangkan imbalan jangka panjang dalam pengambilan keputusan.

7. **gae_lambda**

   Parameter dalam perhitungan *Generalized Advantage Estimation* (GAE). Nilai ini mengatur keseimbangan antara bias dan variansi. Angka 0.95 umumnya menghasilkan pelatihan yang lebih stabil.

8. **clip_range**

   Digunakan untuk membatasi perubahan besar pada kebijakan selama proses pelatihan.Dengan adanya batas ini, pembaruan kebijakan menjadi lebih aman dan tidak terlalu drastis, sehingga mengurangi risiko pelatihan yang tidak stabil.

9. **ent_coef**

   Koefisien entropi yang mengatur seberapa besar dorongan agen untuk mengeksplorasi aksi baru. Jika diatur ke 0, agen hanya fokus pada eksploitasi dari kebijakan yang telah dipelajari.

10. **vf_coef**

    Koefisien yang mengatur kontribusi fungsi nilai terhadap total kerugian. Digunakan untuk menyeimbangkan pembelajaran antara bagian *actor* dan *critic*.

11. **max_grad_norm**

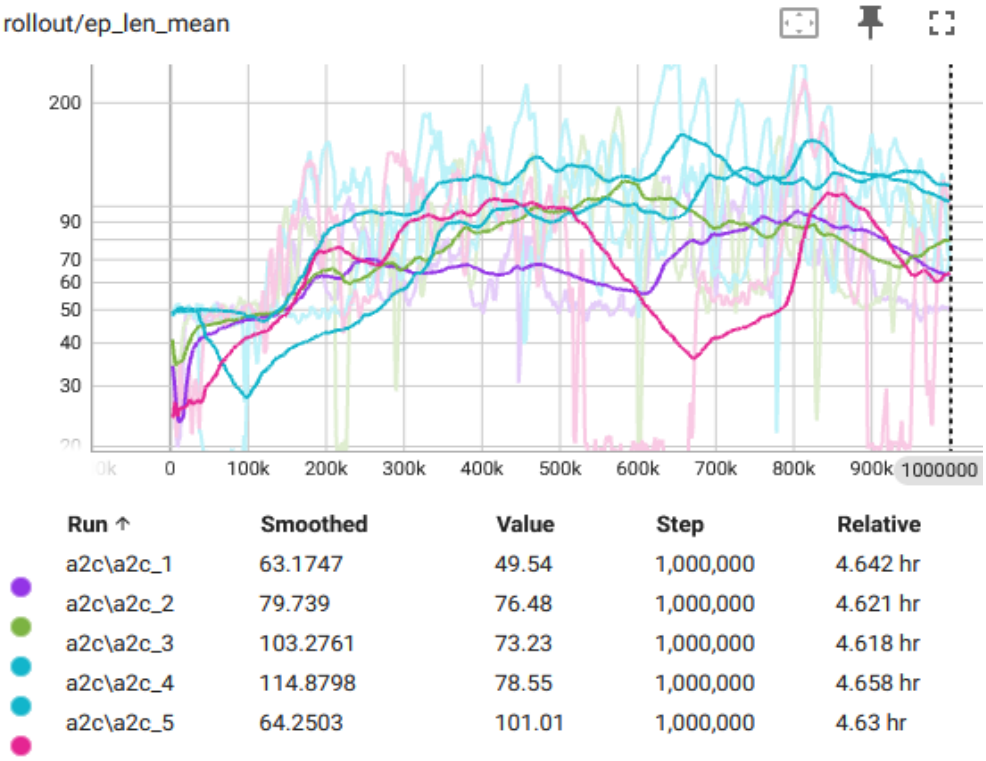
    Digunakan untuk melakukan *gradient clipping*, yaitu membatasi nilai maksimum gradien agar tidak terlalu besar. Hal ini penting untuk menjaga stabilitas proses pelatihan, terutama saat terjadi perubahan besar dalam jaringan.

Parameter-parameter tersebut sebagian besar dipilih berdasarkan nilai default dari SB3 dan disesuaikan dengan praktik umum dalam pelatihan agen pada lingkungan yang relatif sederhana seperti *CartPole* dan *CliffWalking*. Nilai *total timesteps* yang digunakan sama untuk kedua algoritma agar perbandingan performa menjadi adil.

4.2 Hasil Pengujian

Setelah model selesai dilatih, hasilnya dianalisis menggunakan *TensorBoard*. *Tool* ini membantu melihat perkembangan performa agen secara visual, seperti seberapa panjang agen bisa bertahan (*ep_len_mean*) dan seberapa besar *reward* yang didapat (*ep_rew_mean*). Dengan tampilan grafik yang interaktif, *TensorBoard* memudahkan dalam membandingkan kinerja algoritma A2C dan PPO, sekaligus memberikan gambaran yang jelas tentang bagaimana agen belajar dan berkembang seiring waktu.

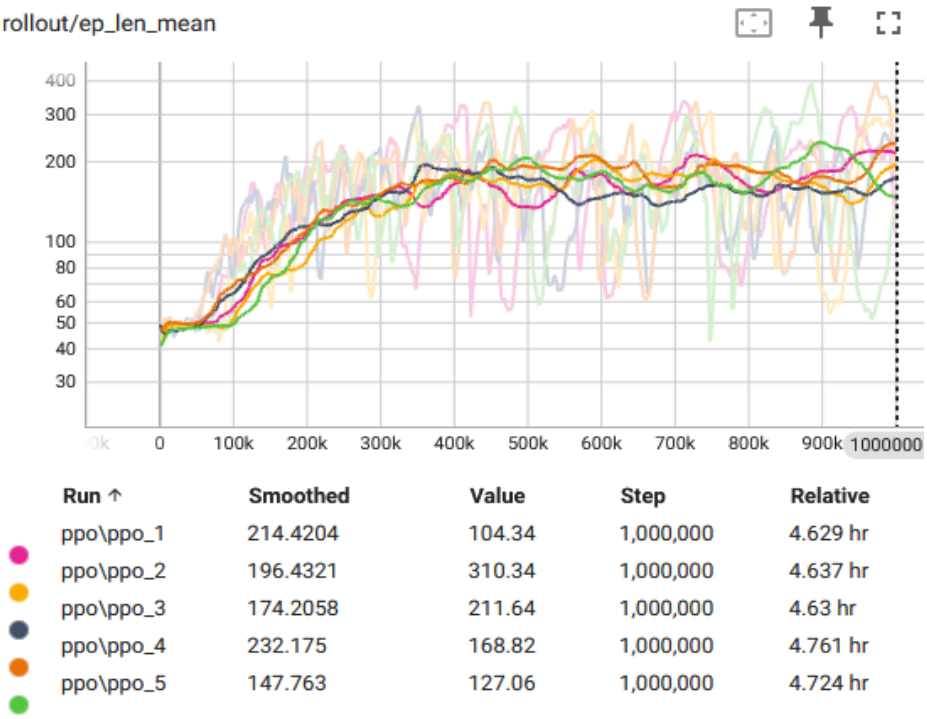
4.2.1 CartPole



Run	Smoothed	Value	Step	Relative
a2c\a2c_1	63.1747	49.54	1,000,000	4.642 hr
a2c\a2c_2	79.739	76.48	1,000,000	4.621 hr
a2c\a2c_3	103.2761	73.23	1,000,000	4.618 hr
a2c\a2c_4	114.8798	78.55	1,000,000	4.658 hr
a2c\a2c_5	64.2503	101.01	1,000,000	4.63 hr

Gambar 4.5 Grafik ep_len_mean Algoritma PPO hasil training CartPole

Gambar 4.5 menunjukkan grafik panjang episode (ep_len_mean) selama proses pelatihan algoritma *Advantage Actor-Critic* (A2C). Sumbu horizontal menunjukkan jumlah langkah *step*, sedangkan sumbu vertikal menunjukkan panjang rata-rata episode yang berhasil diselesaikan oleh agen. Semakin tinggi nilai *ep_len_mean*, maka semakin lama agen dapat bertahan dalam satu episode sebelum dinyatakan gagal, yang dalam konteks CartPole berarti semakin baik kemampuan agen menjaga keseimbangan *pole* di atas *cart*. Grafik ini merepresentasikan hasil pelatihan kedua algoritma pada lingkungan *CartPole* hingga mencapai 1 juta langkah interaksi antara agen dan lingkungan. Algoritma A2C mengalami peningkatan yang lebih lambat dan tidak stabil. Pola fluktuatif pada garis A2C juga menunjukkan bahwa agen lebih rentan terhadap ketidakkonsistenan dalam kebijakan yang dipelajari, sehingga kesulitan untuk mempertahankan performa tinggi secara stabil.



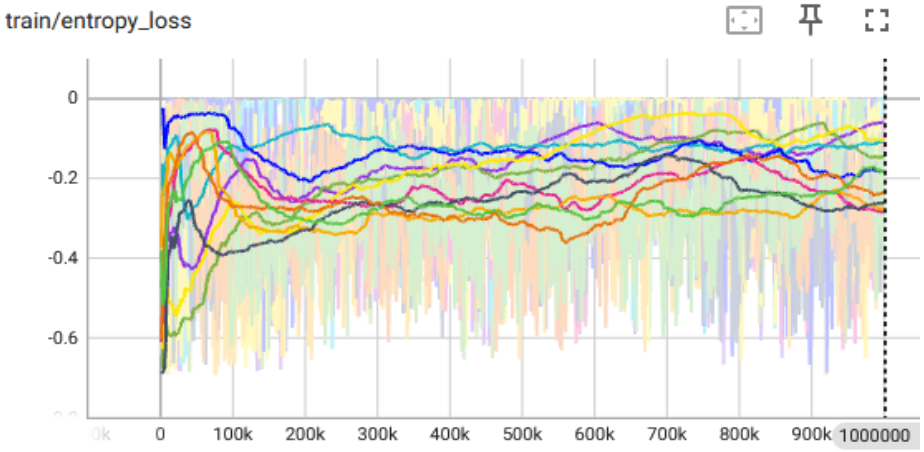
Run	Smoothed	Value	Step	Relative
ppo\ppo_1	214.4204	104.34	1,000,000	4.629 hr
ppo\ppo_2	196.4321	310.34	1,000,000	4.637 hr
ppo\ppo_3	174.2058	211.64	1,000,000	4.63 hr
ppo\ppo_4	232.175	168.82	1,000,000	4.761 hr
ppo\ppo_5	147.763	127.06	1,000,000	4.724 hr

Gambar 4.6 Grafik ep_len_mean Algoritma PPO hasil training CartPole

Gambar 4.6 menunjukkan grafik panjang episode (ep_len_mean) selama proses pelatihan algoritma *Proximal Policy Optimization* (PPO). Dari grafik tersebut terlihat bahwa algoritma PPO secara konsisten menunjukkan performa yang lebih baik dibandingkan A2C. PPO memperlihatkan peningkatan rerata panjang episode yang lebih cepat dan stabil sejak awal pelatihan, serta mempertahankan performa tinggi dengan fluktuasi yang tidak terlalu ekstrem hingga akhir pelatihan. Hal ini menunjukkan bahwa agen yang dilatih dengan PPO hampir mencapai batas maksimal panjang episode dalam lingkungan CartPole. Sebaliknya, algoritma A2C mengalami peningkatan yang lebih lambat dan tidak stabil. Pola fluktuatif pada garis A2C juga menunjukkan bahwa agen lebih rentan terhadap ketidakkonsistenan dalam kebijakan yang dipelajari, sehingga kesulitan untuk mempertahankan performa tinggi secara stabil.

Secara keseluruhan, grafik ini menunjukkan bahwa PPO lebih unggul dibandingkan A2C dalam konteks pelatihan pada lingkungan *CartPole*. Hal ini sesuai dengan karakteristik PPO yang mengutamakan kestabilan dalam

pembaruan kebijakan melalui mekanisme kliping, sehingga proses belajarnya lebih terarah dan efisien. Sementara itu, A2C yang tidak memiliki mekanisme pengendali serupa, cenderung menghasilkan performa yang kurang stabil dalam lingkungan dinamis seperti *CartPole*.

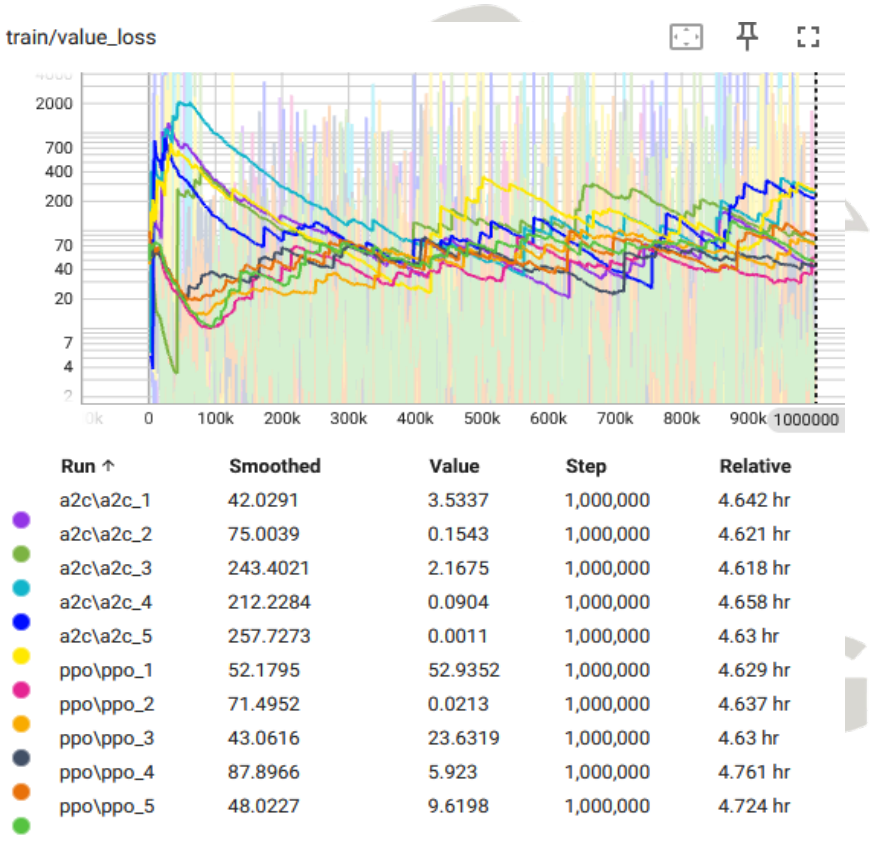


Run	Smoothed	Value	Step	Relative
a2c\a2c_1	-0.0615	-0.0175	1,000,000	4.642 hr
a2c\a2c_2	-0.1437	-0.0001	1,000,000	4.621 hr
a2c\a2c_3	-0.1086	-0.1677	1,000,000	4.618 hr
a2c\a2c_4	-0.1822	-0.0983	1,000,000	4.658 hr
a2c\a2c_5	-0.1033	-0.0519	1,000,000	4.63 hr
ppo\ppo_1	-0.2823	-0.307	1,000,000	4.629 hr
ppo\ppo_2	-0.2763	-0.3084	1,000,000	4.637 hr
ppo\ppo_3	-0.2607	-0.1591	1,000,000	4.63 hr
ppo\ppo_4	-0.2347	-0.1515	1,000,000	4.761 hr
ppo\ppo_5	-0.1849	-0.4088	1,000,000	4.724 hr

Gambar 4.7 Grafik entropy_loss hasil training CartPole

Gambar 4.7 memberikan gambaran mengenai tingkat eksplorasi agen selama proses pelatihan. Dalam konteks *reinforcement learning*, *entropy* digunakan sebagai ukuran seberapa acak keputusan yang diambil oleh agen. Nilai *entropy* yang tinggi menunjukkan bahwa agen masih aktif mencoba berbagai aksi (eksplorasi), sedangkan nilai yang rendah mengindikasikan bahwa agen mulai menetapkan pilihan aksi yang konsisten (eksploitasi).Berdasarkan grafik yang dihasilkan, algoritma PPO memperlihatkan nilai *entropy loss* yang lebih rendah

secara konsisten dibandingkan A2C, namun tetap dalam rentang yang stabil. Hal ini menandakan bahwa PPO menjaga eksplorasi dengan lebih hati-hati dan seimbang. Sementara itu, A2C terlihat cepat menurunkan nilai *entropy*-nya di awal pelatihan, sehingga agen cenderung terlalu cepat melakukan eksploitasi. Penurunan eksplorasiyang terlalu dini dapat menyebabkan agen belajar strategi yang kurang optimalkarena belum sempatmencoba cukup banyak variasiaksi. Dengan demikianpada aspek ini, PPO menunjukkan pendekatan eksplorasi yang lebih adaptif dan stabil sepanjang pelatihan.

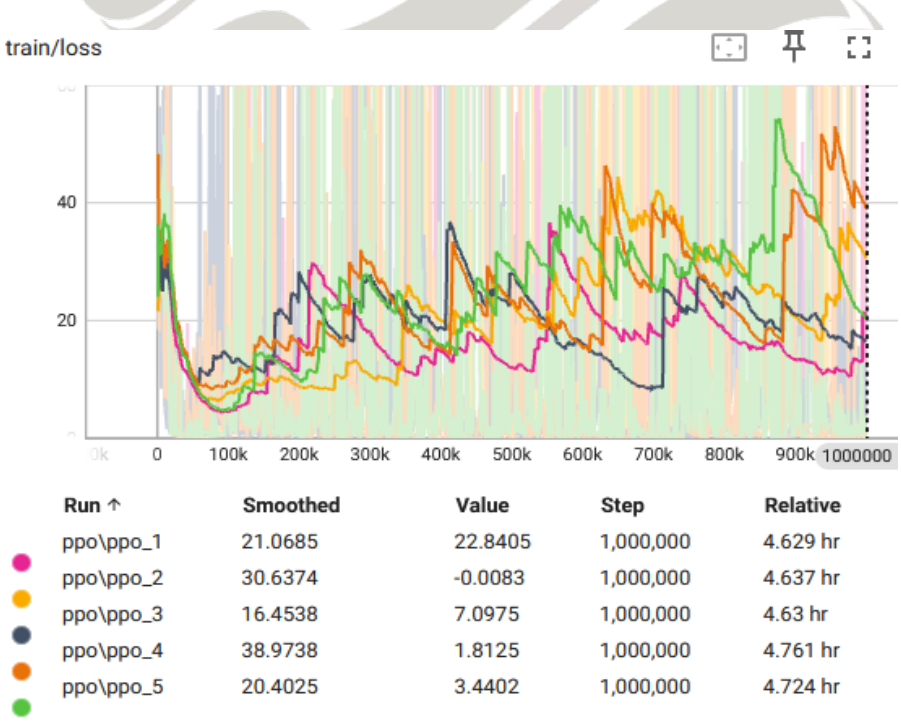


Run	Smoothed	Value	Step	Relative
a2c\a2c_1	42.0291	3.5337	1,000,000	4.642 hr
a2c\a2c_2	75.0039	0.1543	1,000,000	4.621 hr
a2c\a2c_3	243.4021	2.1675	1,000,000	4.618 hr
a2c\a2c_4	212.2284	0.0904	1,000,000	4.658 hr
a2c\a2c_5	257.7273	0.0011	1,000,000	4.63 hr
ppo\ppo_1	52.1795	52.9352	1,000,000	4.629 hr
ppo\ppo_2	71.4952	0.0213	1,000,000	4.637 hr
ppo\ppo_3	43.0616	23.6319	1,000,000	4.63 hr
ppo\ppo_4	87.8966	5.923	1,000,000	4.761 hr
ppo\ppo_5	48.0227	9.6198	1,000,000	4.724 hr

Gambar 4.8 Grafik value_loss hasil training CartPole

Gambar 4.8 merepresentasikan seberapa besar kesalahan prediksi yang dilakukan oleh komponen *critic* dalam memperkirakan nilai suatu keadaan (*state*). Dalam algoritma *reinforcement learning*, *critic* memiliki peran penting dalam mengevaluasi seberapa baik suatu aksi yang diambil berdasarkan nilai yang

diharapkan.Dari hasil grafik,terlihat bahwa algoritma A2C mengalami fluktuasi *value loss* yang sangat tinggi dan tidak stabil. Beberapa lonjakan nilainya bahkan mencapai,yang menandakan bahwa estimasinilai yang dihasilkan oleh jaringan *critic* pada A2C sering kali meleset jauh dari target sebenarnya. Sebaliknya, meskipun PPO juga menunjukkan lonjakan pada *value loss*, pola yang dihasilkan lebih stabil dan terkendali, dengan variasi yang cenderung lebih konsisten sepanjang pelatihan.Nilai rata-rata yang lebih tinggi pada PPO sebenarnya tidak selalu buruk, karena kestabilan pola menjadi indikator penting dalam proses pembelajaran yang sehat. Berdasarkan pengamatan ini, dapat disimpulkan bahwa PPO lebih mampu menjaga kestabilan dalam pembaruan fungsi nilai dibandingkan A2C, yang cenderung mengalami ketidakteraturan akibat pembaruan yang agresif dan tanpa mekanisme pengendali seperti *clipping*.

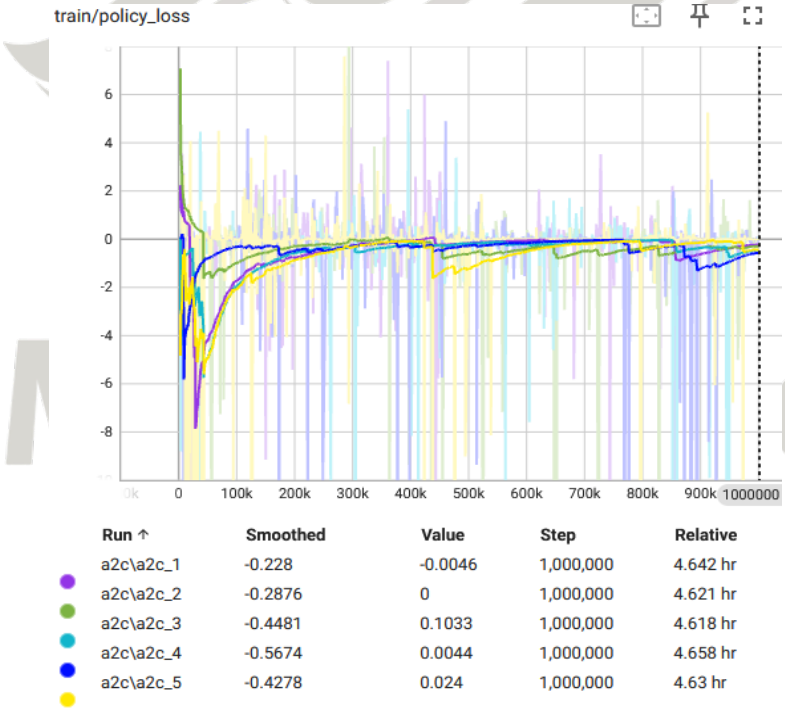


Run	Smoothed	Value	Step	Relative
ppo\ppo_1	21.0685	22.8405	1,000,000	4.629 hr
ppo\ppo_2	30.6374	-0.0083	1,000,000	4.637 hr
ppo\ppo_3	16.4538	7.0975	1,000,000	4.63 hr
ppo\ppo_4	38.9738	1.8125	1,000,000	4.761 hr
ppo\ppo_5	20.4025	3.4402	1,000,000	4.724 hr

Gambar 4.9 Grafik train loss hasil training CartPole

Gambar 4.9 menunjukkan nilai keseluruhan dari fungsi kerugian yang digunakan selama pelatihan model.Pada algoritma PPO, *train loss* merupakan kombinasi dari beberapa komponen, yaitu *policy loss*, *value loss*, dan penalti

entropy. Dari grafik yang ditampilkan,terlihat bahwa nilai train loss mengalami fluktuasi selama proses pelatihan, dengan beberapa lonjakan yang cukup tajam pada titik-titik tertentu. Meskipun demikian,pola keseluruhannya tetap berada dalam batas yang wajar dan tidak menunjukkan tanda-tanda *divergence*,yaitu kondisi di mana nilai loss meningkat secara tidak terkendali. Nilai *train loss* yang naik turun secara periodik merupakan hal yang umum dalam pelatihan *reinforcement learning*, terutama karena dinamika interaksi agen dengan lingkungan yang berubah seiring dengan perkembangan kebijakan. Selain itu, stabilitas garis *smoothed* menunjukkan bahwa meskipun terdapatariasi, model PPO tetap mampu menjaga arah pembelajaran yang konsisten dan tidak terganggu oleh perubahan sesaat. Secara keseluruhan,*train loss* pada PPO mencerminkan proses pelatihan yang relatif stabil dan menunjukkan kemajuan yang sejalan dengan peningkatan performa agen.

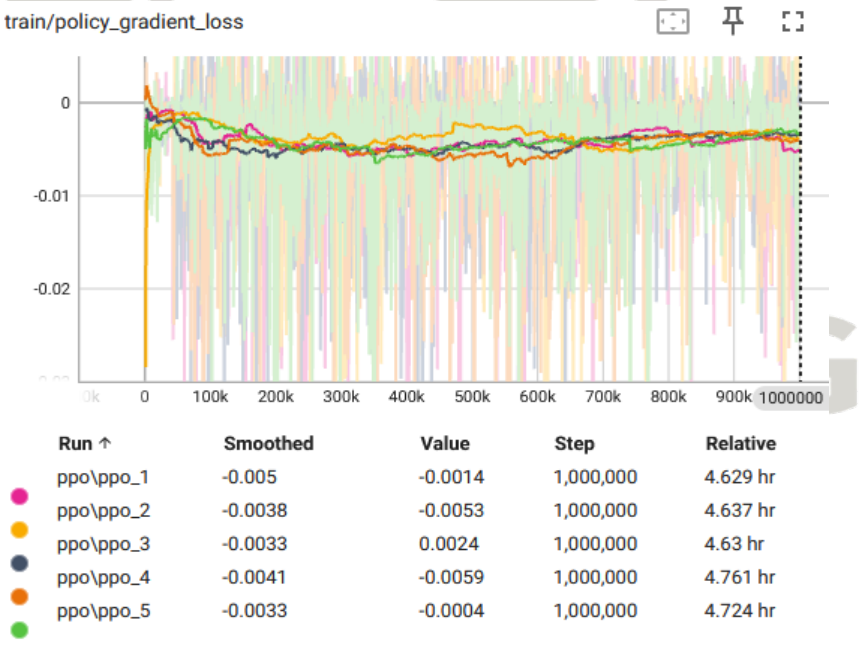


Run	Smoothed	Value	Step	Relative
a2c\a2c_1	-0.228	-0.0046	1,000,000	4.642 hr
a2c\a2c_2	-0.2876	0	1,000,000	4.621 hr
a2c\a2c_3	-0.4481	0.1033	1,000,000	4.618 hr
a2c\a2c_4	-0.5674	0.0044	1,000,000	4.658 hr
a2c\a2c_5	-0.4278	0.024	1,000,000	4.63 hr

Gambar 4.10 Grafik policy_loss hasil training CartPole

Gambar 4.10 mencerminkan seberapa besar kesalahan kebijakan yang terjadi selama pelatihan,yaitu seberapa jauh distribusi aksi yang dipilih oleh agen

menyimpang dari arah kebijakan yang diharapkan berdasarkan nilai *advantage*. Pada algoritma A2C, grafik ini menunjukkan fluktuasiyang sangat tinggi dan tidak stabil sepanjang proses pelatihan.Nilai *policy loss* sering kali mengalami spike negatif yang besar. Pola ini menunjukkan bahwa pembaruan kebijakan yang dilakukan oleh A2C sering kali terlalu drastis atau tidak terarah.Tidak adanya mekanisme pembatasan seperti clipping, sebagaimana terdapat dalam PPO, membuat A2C lebih rentan terhadap perubahan kebijakan yang ekstrim akibat sinyal pelatihan yang bervariasi. Meskipun pada beberapa titik nilai *smoothed* dari *policy loss* tampak mendekati nol, hal ini tidak sepenuhnya mencerminkan stabilitas,karena fluktuasiyang ekstrim tetap terjadisecara berkala.Kondisi ini menunjukkan bahwa A2C lebih sulit mencapai konvergensi kebijakan yang stabil dibandingkan PPO. Oleh karena itu, policy loss pada A2C menjadi indikator kelemahan dalam menjaga konsistensi arah pembelajaran kebijakan secara jangka panjang.

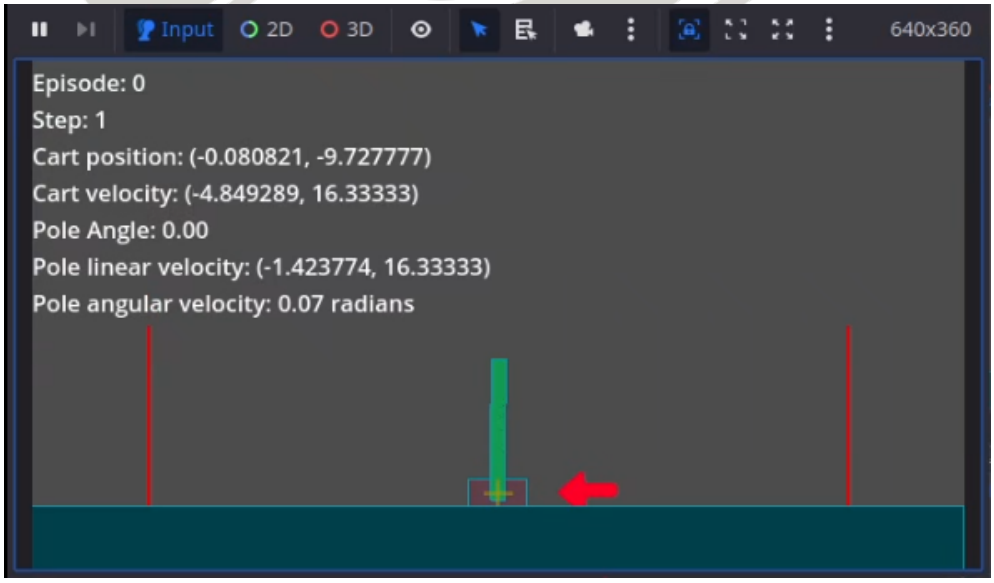


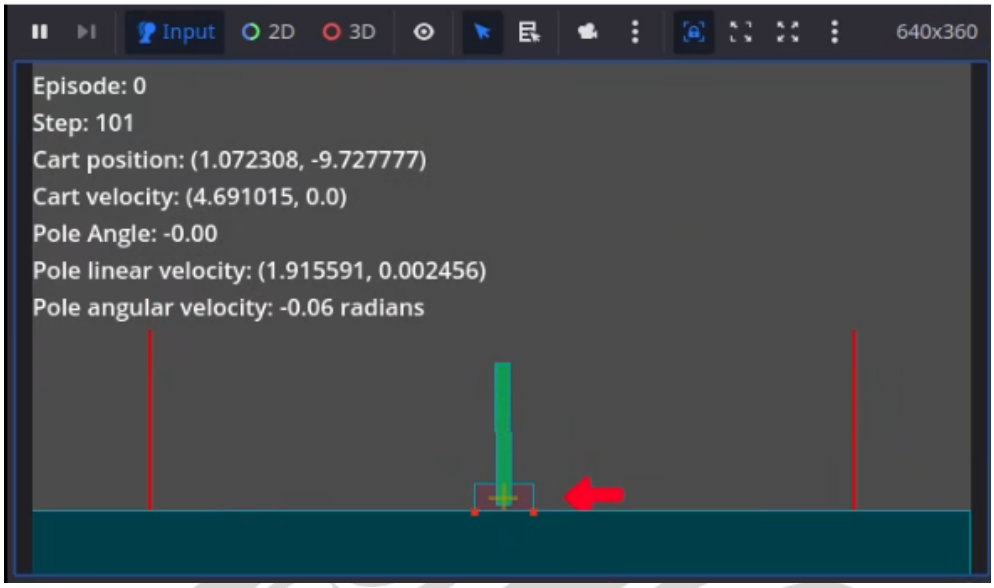
Run	Smoothed	Value	Step	Relative
ppo\ppo_1	-0.005	-0.0014	1,000,000	4.629 hr
ppo\ppo_2	-0.0038	-0.0053	1,000,000	4.637 hr
ppo\ppo_3	-0.0033	0.0024	1,000,000	4.63 hr
ppo\ppo_4	-0.0041	-0.0059	1,000,000	4.761 hr
ppo\ppo_5	-0.0033	-0.0004	1,000,000	4.724 hr

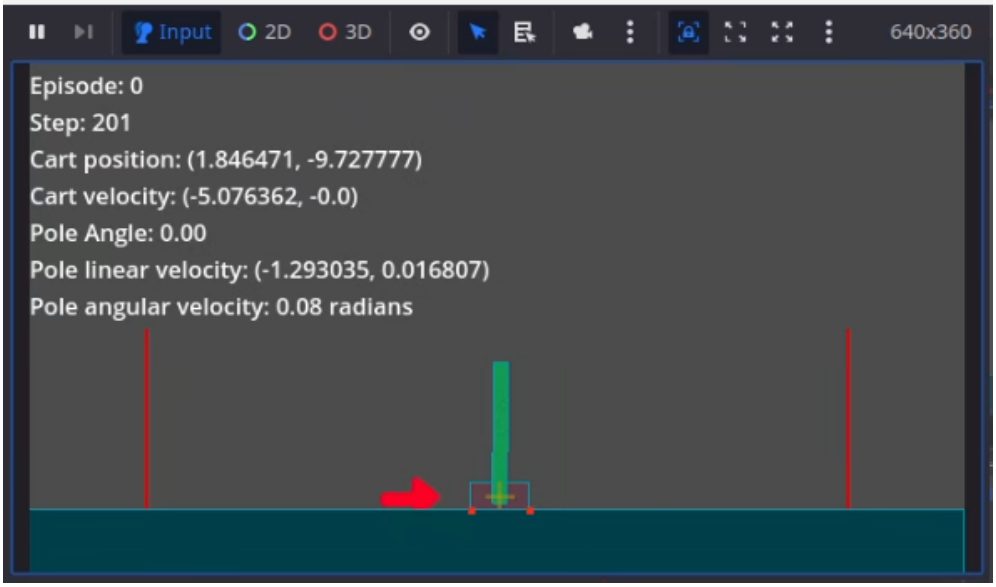
Gambar 4.11 Grafik policy_gradient_loss hasil training CartPole

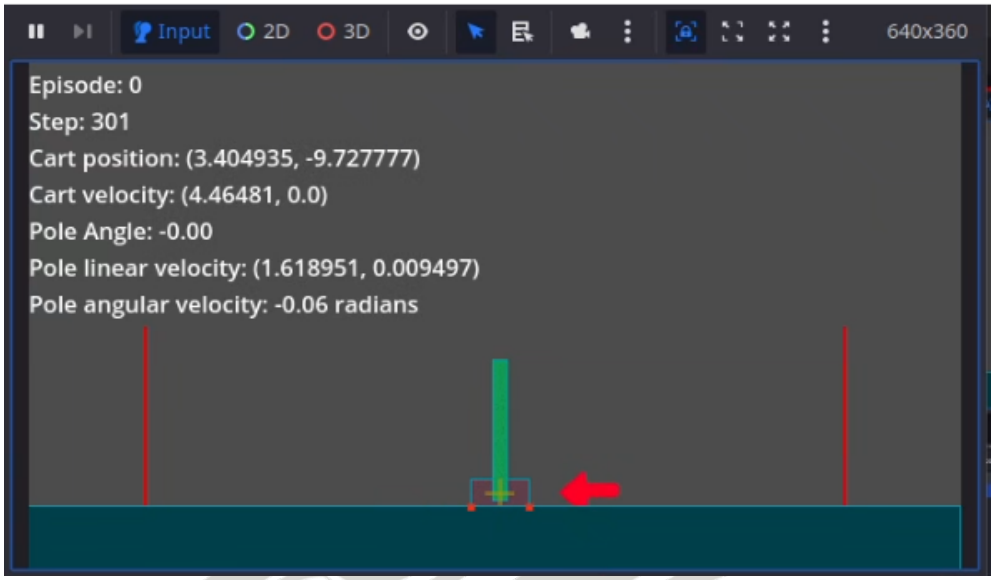
Gambar 4.11 menggambarkan besarnya pembaruan yang dilakukan terhadap kebijakan (*policy*) agen selama pelatihan, khususnya dalam konteks

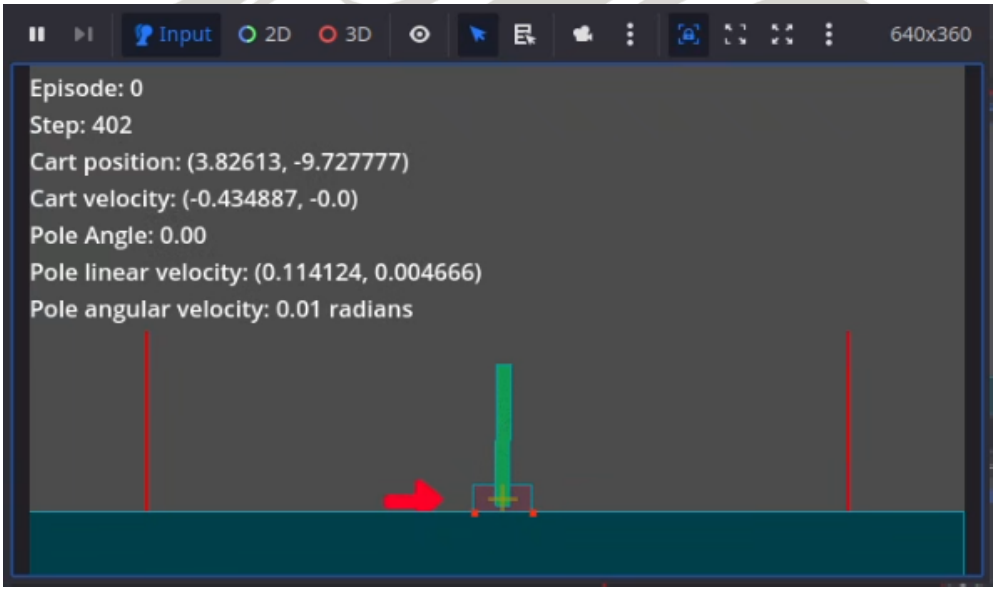
algoritma PPO. Nilai loss ini berasal dari hasil perhitungan gradien kebijakan yang digunakan untuk memperbarui model, dan berperan penting dalam mengarahkan agen menuju kebijakan yang lebih optimal. Dari hasil grafik, terlihat bahwa nilai *policy gradient loss* berada pada kisaran yang sangat kecil dan secara bertahap menurun seiring berjalannya pelatihan. Hal ini mengindikasikan bahwa kebijakan yang dimiliki agen telah semakin stabil, sehingga tidak lagi membutuhkan pembaruan yang besar. Nilai *smoothed* yang mendekati nol pada tahap akhir pelatihan menunjukkan bahwa proses pembelajaran telah mencapai titik konvergensi, di mana agen sudah menemukan strategi yang efektif dalam menyelesaikan tugas pada lingkungan *CartPole*. Stabilitas ini merupakan ciri khas dari PPO yang memiliki mekanisme *clipping*, yaitu pembatasan terhadap perubahan kebijakan yang terlalu drastis. Dengan demikian, *policy gradient loss* yang rendah dan stabil ini menjadi indikator bahwa PPO berhasil membangun kebijakan yang efisien dan konsisten.

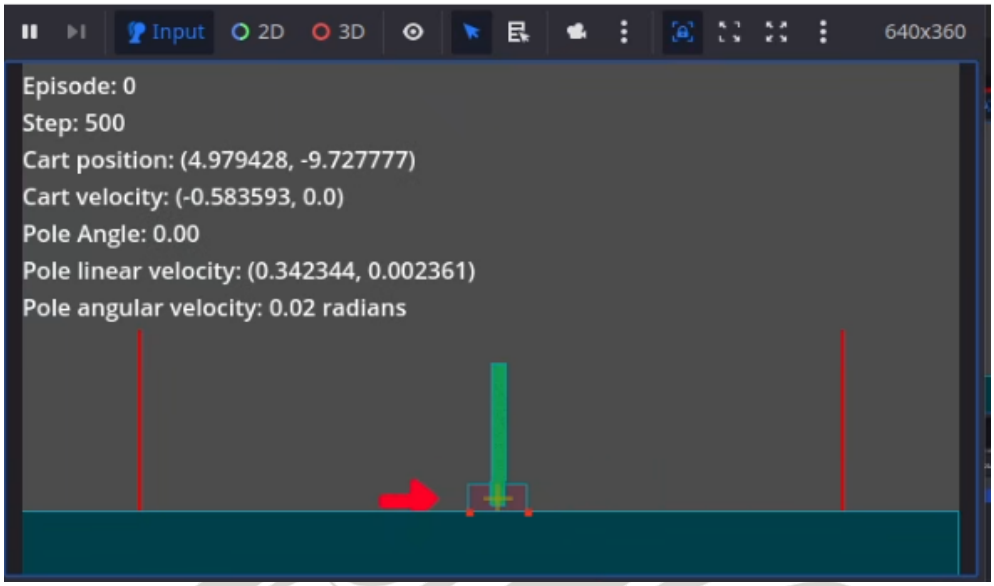



Input
2D
3D
640x360
Episode: 0
Step: 101
Cart position: (1.072308, -9.727777)
Cart velocity: (4.691015, 0.0)
Pole Angle: -0.00
Pole linear velocity: (1.915591, 0.002456)
Pole angular velocity: -0.06 radians


Input
2D
3D
640x360
Episode: 0
Step: 201
Cart position: (1.846471, -9.727777)
Cart velocity: (-5.076362, -0.0)
Pole Angle: 0.00
Pole linear velocity: (-1.293035, 0.016807)
Pole angular velocity: 0.08 radians

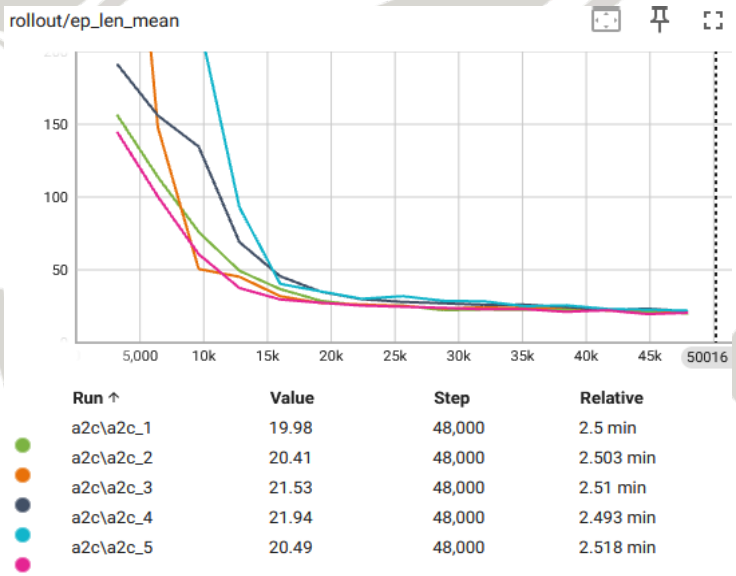

Input
2D
3D
640x360
Episode: 0
Step: 301
Cart position: (3.404935, -9.727777)
Cart velocity: (4.46481, 0.0)
Pole Angle: -0.00
Pole linear velocity: (1.618951, 0.009497)
Pole angular velocity: -0.06 radians


Input
2D
3D
640x360
Episode: 0
Step: 402
Cart position: (3.82613, -9.727777)
Cart velocity: (-0.434887, -0.0)
Pole Angle: 0.00
Pole linear velocity: (0.114124, 0.004666)
Pole angular velocity: 0.01 radians



Gambar 4.12 Agen CartPole Dengan Algoritma PPO Setiap 100 Frame

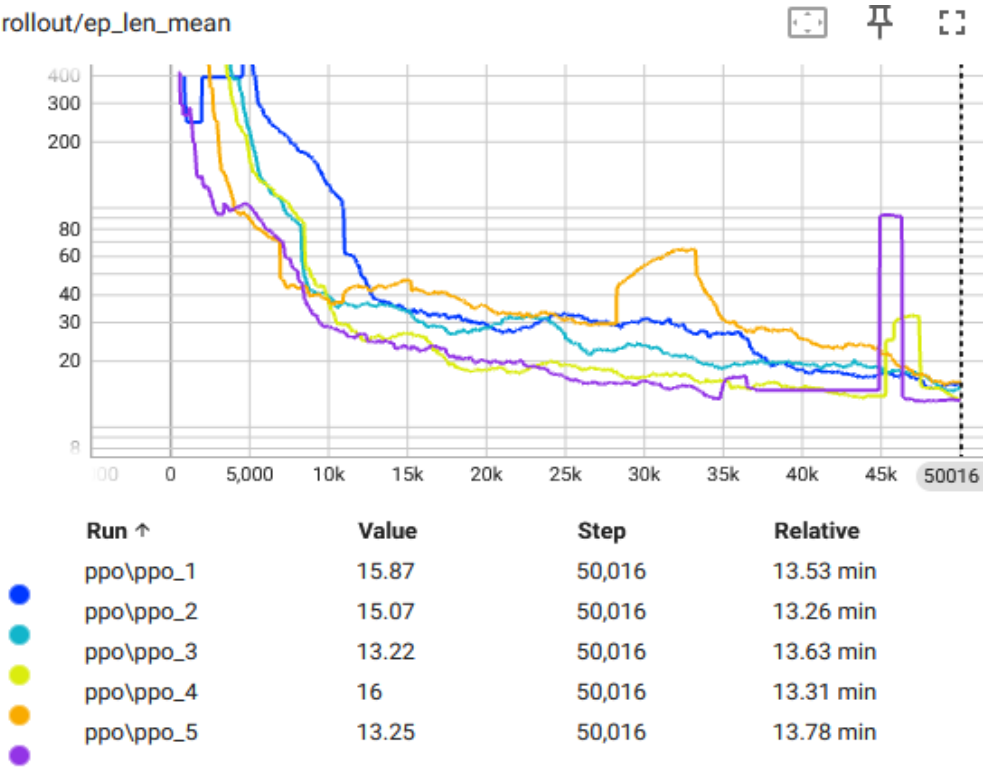
4.2.2 CliffWalking



Gambar 4.13 Grafik ep_len_mean Algoritma A2C hasil training CliffWalking

Gambar 4.13 memperlihatkan grafik hasil pelatihan algoritma A2C pada lingkungan game CliffWalking. Pada grafik tersebut, sumbu vertikal merepresentasikan rata-rata panjang episode, sementara sumbu horizontal

menunjukkan jumlah langkah pelatihan (*steps*). Di tahap awal pelatihan, algoritma A2C masih berada dalam fase eksplorasiyang ditandai dengan panjang episode yang cukup tinggi. Kondisi ini mengindikasikan bahwa agen belum berhasil menemukan jalur tercepat menuju tile finish. Namun, setelah melewati sekitar 5.000 langkah pelatihan, terjadi penurunan tajam pada panjang episode. Penurunan ini menunjukkan bahwa agen mulai menemukan jalur optimal dan mulai menyelesaikan tugas dengan cara yang lebih efisien.

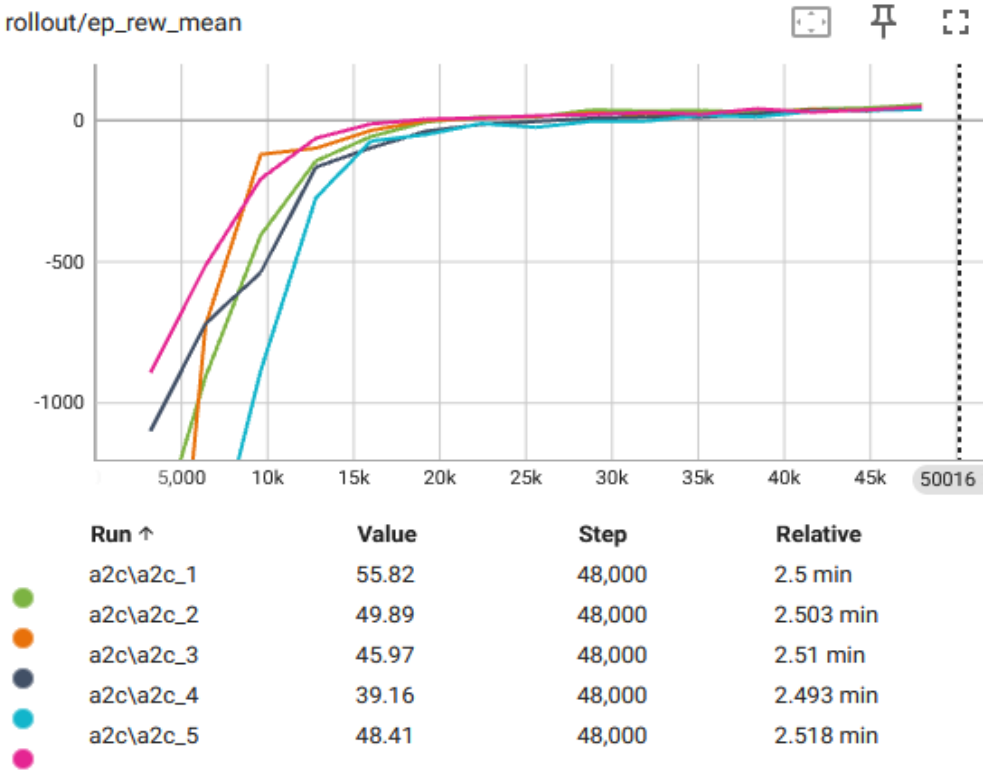


Run	Value	Step	Relative
ppo\ppo_1	15.87	50,016	13.53 min
ppo\ppo_2	15.07	50,016	13.26 min
ppo\ppo_3	13.22	50,016	13.63 min
ppo\ppo_4	16	50,016	13.31 min
ppo\ppo_5	13.25	50,016	13.78 min

Gambar 4.14 Grafik ep_len_mean Algoritma PPO hasil training CliffWalking

Gambar 4.14 menunjukkan grafik hasil pengujian algoritma PPO di game CliffWalking.Di awal pelatihan, algoritma PPO masih menjalankan eksplorasi dengan panjang episode yang relatif tinggi. Hal ini menunjukkan agen belum mampu menemukan jalur optimal menuju tujuan *tile finish* sama seperti algoritma A2C. Setelah sekitar 5.000 langkah, terjadi penurunan drastis pada panjang

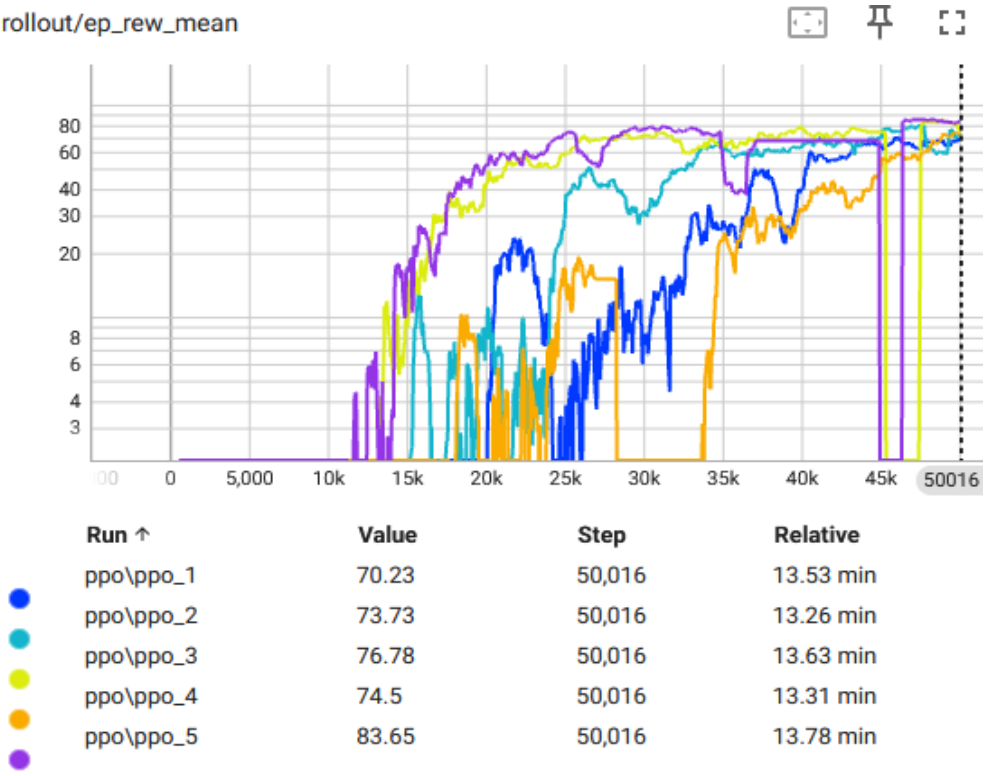
episode, menandakan bahwa agen sudah menemukan lokasi *tile finish* dan mulai belajar untuk menyelesaikan tugas dengan lebih efisien.



Run ↑	Value	Step	Relative
a2c\a2c_1	55.82	48,000	2.5 min
a2c\a2c_2	49.89	48,000	2.503 min
a2c\a2c_3	45.97	48,000	2.51 min
a2c\a2c_4	39.16	48,000	2.493 min
a2c\a2c_5	48.41	48,000	2.518 min

Gambar 4.15 Grafik ep_rew_mean Algoritma A2C hasil training CliffWalking

Gambar 4.15 menunjukkan grafik perkembangan rata-rata *reward* yang diterima oleh agen selama pelatihan. Di awal pelatihan, reward masih sangat rendah. Ini disebabkan oleh agen yang masih mengeksplorasi lingkungan dan sering terkena penalti karena jatuh ke *tile cliff*. Setelah sekitar 10.000 langkah, baik A2C mulai menunjukkan peningkatan signifikan dalam rata-rata reward.

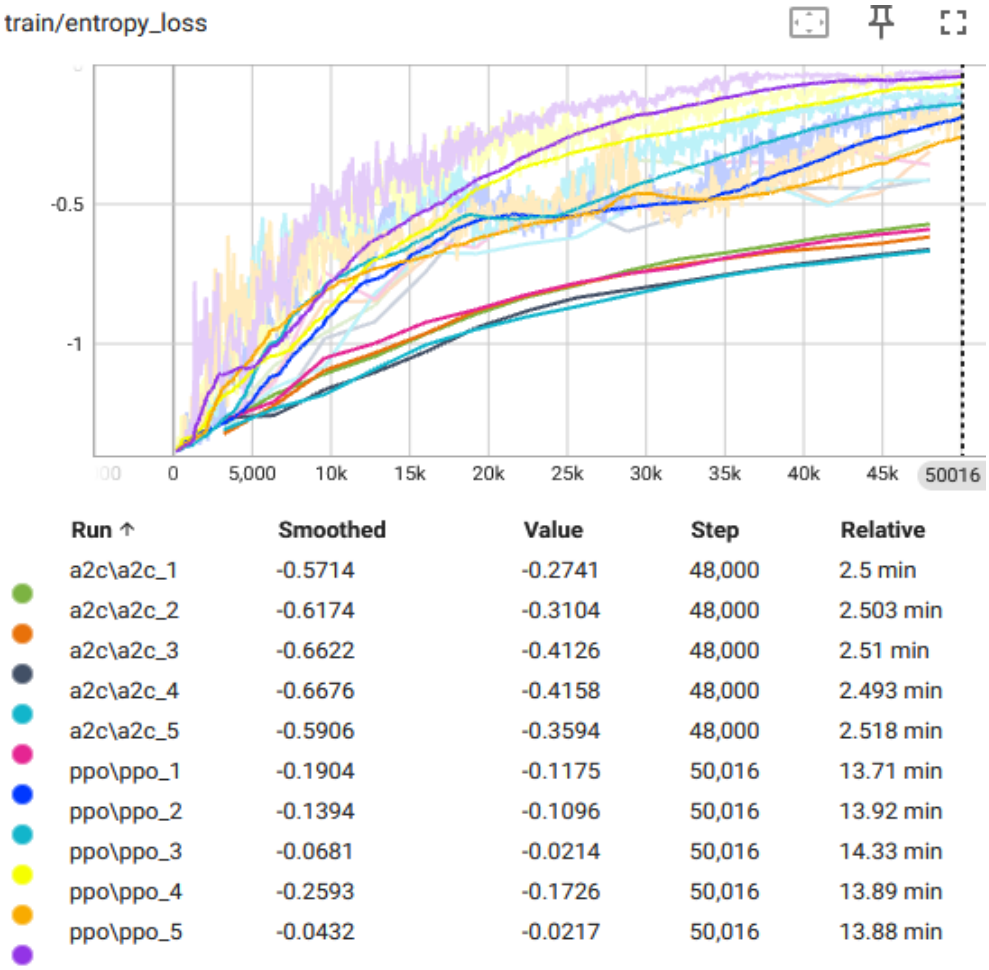


Run	Value	Step	Relative
ppo\ppo_1	70.23	50,016	13.53 min
ppo\ppo_2	73.73	50,016	13.26 min
ppo\ppo_3	76.78	50,016	13.63 min
ppo\ppo_4	74.5	50,016	13.31 min
ppo\ppo_5	83.65	50,016	13.78 min

Gambar 4.16 Grafik ep_rew_mean Algoritma PPO hasil training CliffWalking

Gambar 4.16 menunjukkan grafik perkembangan rata-rata *reward* yang diterima oleh agen selama pelatihan. Di awal pelatihan (0–25.000 langkah), reward masih tidak ada. Ini disebabkan oleh agen yang masih mengeksplorasi lingkungan dan sering terkena penalti karena jatuh ke *tile cliff*. Setelah sekitar 30.000 langkah, baik A2C maupun PPO mulai menunjukkan peningkatan signifikan dalam rata-rata reward.

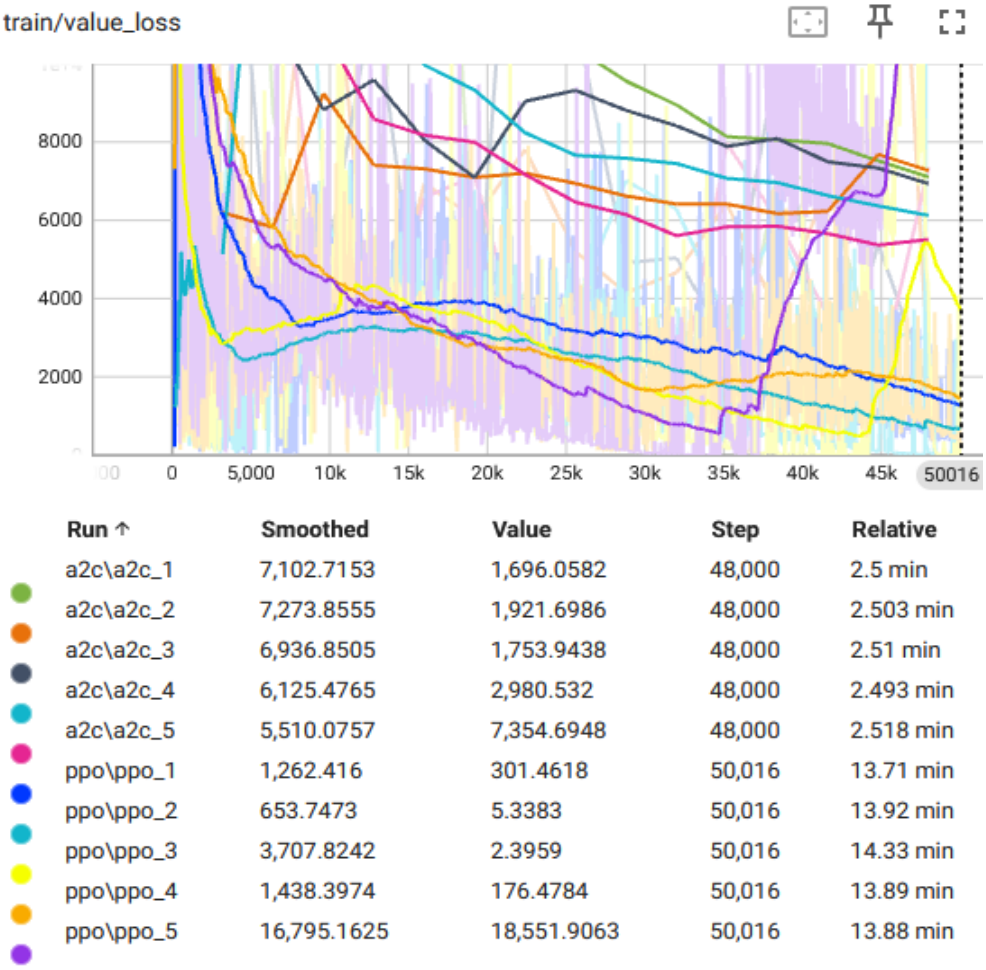
PPO sempat mengalami kenaikan dan fluktuasi yang lebih tajam, namun setelah 35.000 langkah keduanya mencapai stabilitas performa. Pada akhir pelatihan, kedua algoritma menghasilkan rata-rata reward yang tinggi dan serupa (sekitar 68–70 poin), menandakan bahwa agen berhasil mempelajari strategi optimal untuk mencapai tujuan sambil menghindari penalti.



Run	Smoothed	Value	Step	Relative
a2c\a2c_1	-0.5714	-0.2741	48,000	2.5 min
a2c\a2c_2	-0.6174	-0.3104	48,000	2.503 min
a2c\a2c_3	-0.6622	-0.4126	48,000	2.51 min
a2c\a2c_4	-0.6676	-0.4158	48,000	2.493 min
a2c\a2c_5	-0.5906	-0.3594	48,000	2.518 min
ppo\ppo_1	-0.1904	-0.1175	50,016	13.71 min
ppo\ppo_2	-0.1394	-0.1096	50,016	13.92 min
ppo\ppo_3	-0.0681	-0.0214	50,016	14.33 min
ppo\ppo_4	-0.2593	-0.1726	50,016	13.89 min
ppo\ppo_5	-0.0432	-0.0217	50,016	13.88 min

Gambar 4.17 Grafik entropy_loss hasil training CliffWalking

Gambar 4.17 menampilkan grafik *entropy loss* selama proses pelatihan model pada lingkungan *CliffWalking*. Pada awal pelatihan, kedua algoritma mengalami penurunan *entropy loss* yang signifikan, yang menunjukkan tingginya eksplorasi. Seiring waktu, *entropy loss* meningkat perlahan mendekati nol, menandakan bahwa agen mulai mengambil tindakan dengan lebih pasti berdasarkan pengetahuan yang diperolehNilai rata-rata *entropy loss* dari PPO sedikit lebih rendah daripada A2C, meskipun perbedaannya tidak terlalu besar.

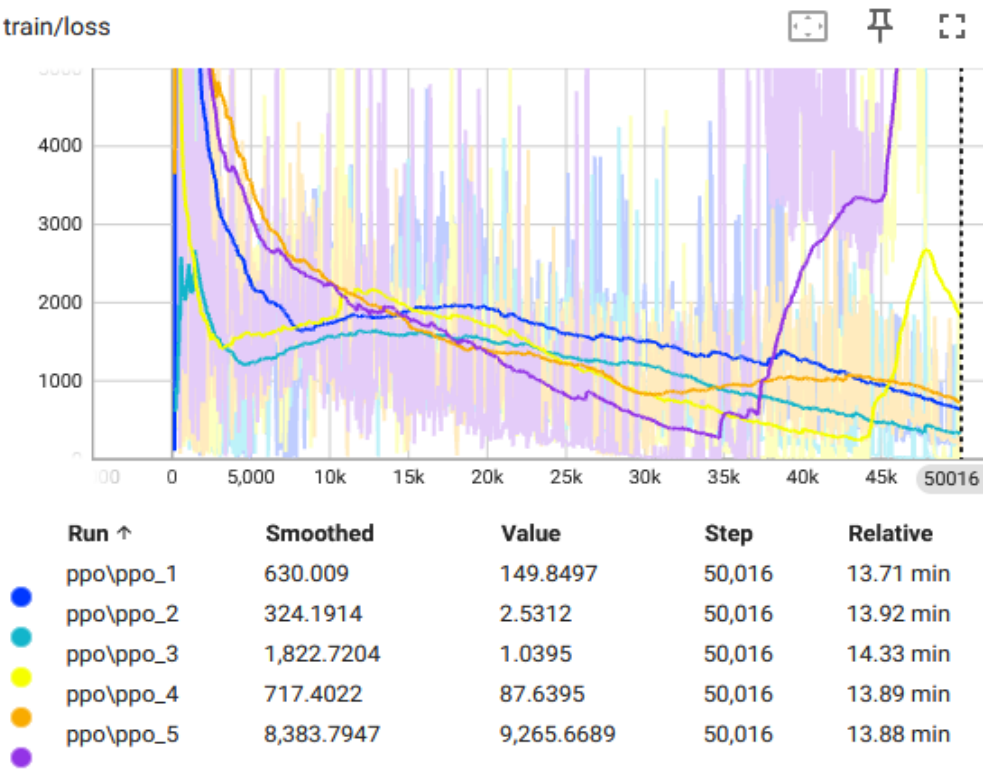


Run	Smoothed	Value	Step	Relative
a2c\a2c_1	7,102.7153	1,696.0582	48,000	2.5 min
a2c\a2c_2	7,273.8555	1,921.6986	48,000	2.503 min
a2c\a2c_3	6,936.8505	1,753.9438	48,000	2.51 min
a2c\a2c_4	6,125.4765	2,980.532	48,000	2.493 min
a2c\a2c_5	5,510.0757	7,354.6948	48,000	2.518 min
ppo\ppo_1	1,262.416	301.4618	50,016	13.71 min
ppo\ppo_2	653.7473	5.3383	50,016	13.92 min
ppo\ppo_3	3,707.8242	2.3959	50,016	14.33 min
ppo\ppo_4	1,438.3974	176.4784	50,016	13.89 min
ppo\ppo_5	16,795.1625	18,551.9063	50,016	13.88 min

Gambar 4.18 Grafik value_loss hasil training CliffWalking

Gambar 4.18 menampilkan grafik *value loss* selama proses pelatihan model pada lingkungan *CliffWalking*. Dari grafik terlihat bahwa kedua algoritma mengalami fluktuasi nilai *value loss* yang cukup besar, terutama di awal pelatihan, yang kemudian secara bertahap mulai mengecil seiring bertambahnya langkah pelatihan. Meskipun demikian, PPO cenderung menghasilkan value loss yang lebih tinggi secara keseluruhan dibandingkan A2C.Nilai *value loss* yang lebih tinggi pada PPO menunjukkan bahwa estimasi nilai dari model tersebut cenderung memilikideviasi yang lebih besar terhadap nilai target, dibandingkan dengan A2C. Meskipun begitu, perbedaan ini tidak serta-merta menunjukkan keunggulan mutlak dari salah satu algoritma, karena efektivitas pembelajaran juga

dipengaruhi oleh faktor lain seperti stabilitas kebijakan dan kemampuan eksplorasi. Secara keseluruhan, grafik ini menggambarkan dinamika pembelajaran fungsi nilai oleh kedua algoritma,dengan pola fluktuasi tinggi yang wajar dalam pelatihan pembelajaran penguatan, terutama di lingkungan seperti *CliffWalking*.

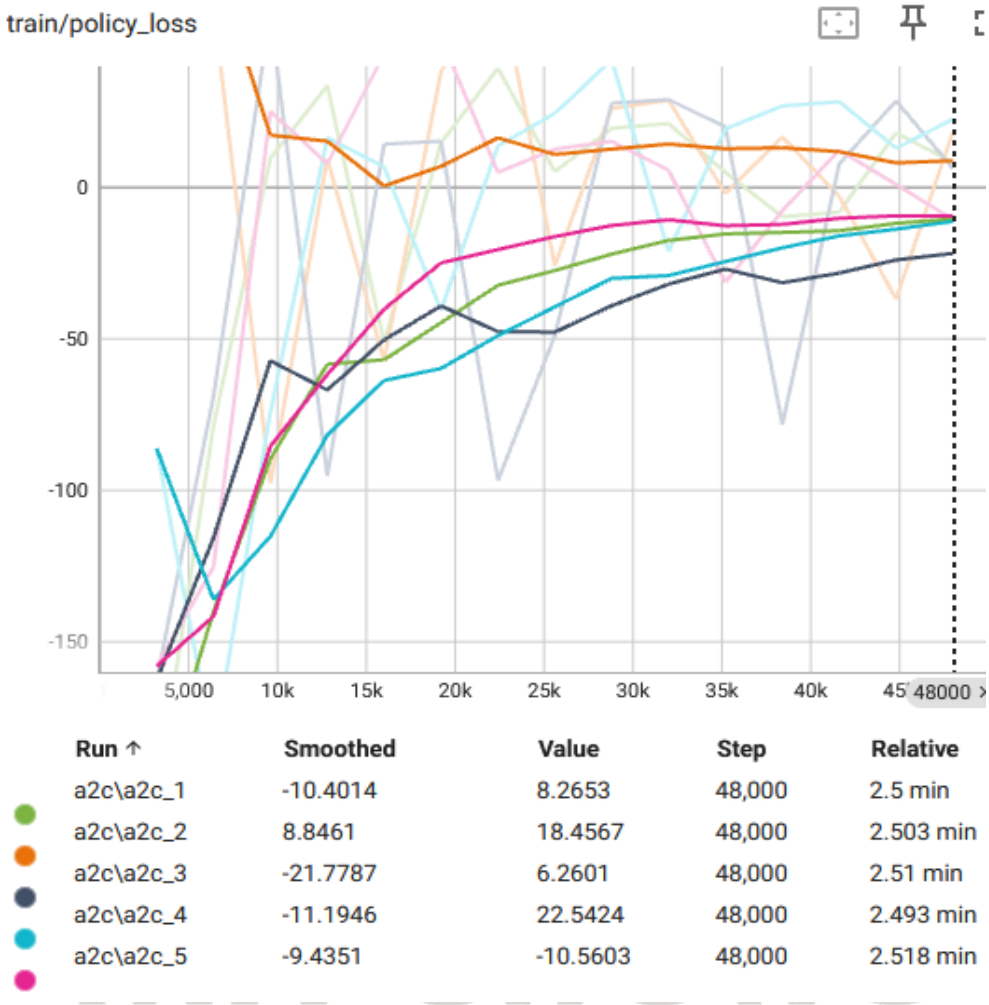


Run	Smoothed	Value	Step	Relative
ppo\ppo_1	630.009	149.8497	50,016	13.71 min
ppo\ppo_2	324.1914	2.5312	50,016	13.92 min
ppo\ppo_3	1,822.7204	1.0395	50,016	14.33 min
ppo\ppo_4	717.4022	87.6395	50,016	13.89 min
ppo\ppo_5	8,383.7947	9,265.6689	50,016	13.88 min

Gambar 4.19 Grafik train loss hasil training CliffWalking

Gambar 4.19 menampilkan grafik *train loss* selama proses pelatihan model pada lingkungan *CliffWalking*. Grafik memperlihatkan bahwa kedua algoritma mengalamifluktuasi nilai *loss* yang cukup tinggi, terutama pada tahap awal pelatihan. Nilai train loss perlahan menurun dan mulai stabil seiring bertambahnya langkah pelatihan,menandakan bahwa modelmulai belajar dari lingkungan secara lebih konsisten.Pada tahap akhir pelatihan PPO menunjukkan penurunan variasi *loss*. Secara keseluruhan, grafik ini menunjukkan algoritma berhasil belajar dan menunjukkan tren penurunan *train loss*,dengan perbedaan performa yang tidak terlalu mencolok.Fluktuasi tinggi di awal mengindikasikan

proses eksplorasi dan penyesuaian terhadap lingkungan,yang kemudian stabil seiring meningkatnya jumlah langkah pelatihan.

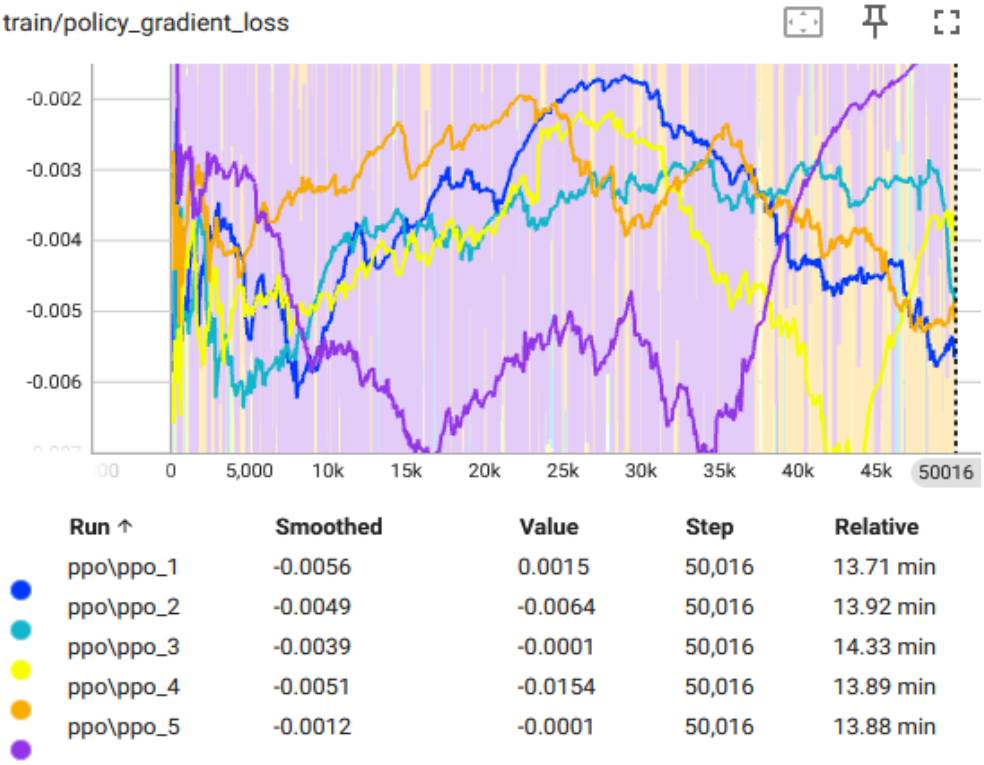


Run	Smoothed	Value	Step	Relative
a2c\a2c_1	-10.4014	8.2653	48,000	2.5 min
a2c\a2c_2	8.8461	18.4567	48,000	2.503 min
a2c\a2c_3	-21.7787	6.2601	48,000	2.51 min
a2c\a2c_4	-11.1946	22.5424	48,000	2.493 min
a2c\a2c_5	-9.4351	-10.5603	48,000	2.518 min

Gambar 4.20 Grafik policy_loss hasil training CliffWalking

Gambar 4.20 menampilkan grafik policy loss selama proses pelatihan algoritma A2C pada lingkungan CliffWalking.Sumbu horizontal menunjukkan jumlah langkah pelatihan (*steps*),sedangkan sumbu vertikalmenunjukkan nilai *policy loss*. Secara umum, *policy loss* mencerminkan seberapa besar kesalahan dalam kebijakan yang diambil oleh agen, semakin rendah nilainya (mendekati nol dari arah negatif), semakin baik performa agen dalam memilih tindakan.

Pada awal pelatihan, sebagian besar run menunjukkan nilai policy loss yang sangat rendah, yang menandakan bahwa agen masih banyak melakukan kesalahan dalam menentukan tindakan. Namun, seiring berjalannya waktu, terlihat adanya tren peningkatan nilaipolicy loss yang mendekatinol, terutama setelah melewati 10.000 langkah. Hal ini menunjukkan bahwa agen mulai belajar dari pengalaman dan melakukan pembaruan kebijakan yang lebih efektif.

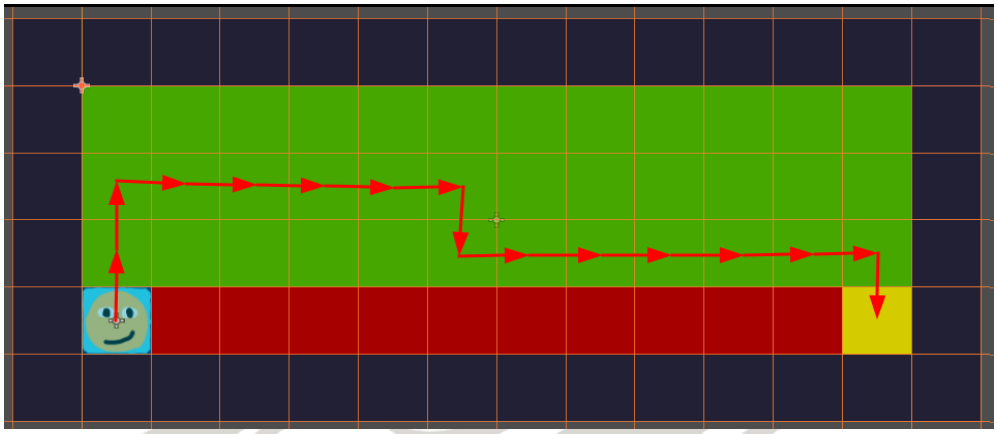


Run	Smoothed	Value	Step	Relative
ppo\ppo_1	-0.0056	0.0015	50,016	13.71 min
ppo\ppo_2	-0.0049	-0.0064	50,016	13.92 min
ppo\ppo_3	-0.0039	-0.0001	50,016	14.33 min
ppo\ppo_4	-0.0051	-0.0154	50,016	13.89 min
ppo\ppo_5	-0.0012	-0.0001	50,016	13.88 min

Gambar 4.21 Grafik policy_gradient_loss hasil training CliffWalking

Gambar 4.21 menunjukkan grafik *policy gradient loss* selama proses pelatihan algoritma pada lingkungan *CliffWalking*. Dari grafik terlihat bahwa nilai *policy gradient loss* untuk kedua algoritma berfluktuasicukup tinggi sepanjang proses pelatihan.Meskipun keduanya memiliki rentang fluktuasi yang mirip, nilai *smoothed loss* dari PPO sedikit lebih rendah dibandingkan dengan A2C,yang menunjukkan bahwa PPO melakukan pembaruan kebijakan yang cenderung lebih kecil secara rata-rata. Fluktuasi tinggi ini merupakan hal yang umum dalam

pelatihan agen pembelajaran penguatan karena agen terus-menerus menyesuaikan tindakannya berdasarkan pengalaman baru. Secara keseluruhan, grafik ini menunjukkan bahwa baik A2C maupun PPO berhasil menjalankan proses pembaruan kebijakan, meskipun masih terdapat dinamika yang tidak stabil hingga akhir pelatihan.



Gambar 4.22 Jalur Algoritma PPO Agen CliffWalking

4.3 Analisis dan Pembahasan

Pengujian signifikansi statistik dilakukan menggunakan Welch's t-test dengan tingkat signifikansi 5% ($\alpha = 0.05$). Jika nilai p-value < 0.05, maka terdapat perbedaan yang signifikan antara hasil algoritma A2C dan PPO. Sebaliknya, jika p-value ≥ 0.05, maka perbedaan yang ada tidak signifikan secara statistik. Hipotesis yang digunakan adalah sebagai berikut:

- H_0: Rata-rata hasil A2C sama dengan rata-rata hasil PPO (tidak ada perbedaan signifikan).
- H_1: Rata-rata hasil A2C berbeda dengan rata-rata hasil PPO (terdapat perbedaan signifikan).

Tabel 4.2 Hasil ep_len_mean Training Game CartPole

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
Deviasi	76.50	216.97	151.37	113.14	21.69	115.93

```
import scipy.stats as stats

a2c = [50.32, 88.06, 69.46, 85.86, 103.90]
ppo = [126.82, 305.03, 220.83, 199, 125.59]

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

```
CartPole ep_len_mean
t-statistic: -3.352572217194514
p-value: 0.023071227297450002
```

Gambar 4.21 Hasil Uji T ep_len_mean CartPole Menggunakan Python

Berdasarkan hasilperhitungan menghasilkan nilai sebesar -3,35 dengan p-value 0,023. Karena nilai p lebih kecil dari tingkat signifikansi 0,05, maka perbedaan hasil rata-rata lamanya agen dalam satu episode antara metode A2C dan PPO dinyatakan signifikan secara statistik. Hal ini menunjukkan bahwa performa PPO secara konsisten lebih tinggi atau dapat menyeimbangkan tiang lebih lama dibandingkan A2C pada *game CartPole* dengan rata-rata deviasi sebanyak 115.93 lebih lama.

Tabel 4.3 Hasil ep_len_mean Training Game CliffWalking

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
Deviasi	4.11	5.34	8.31	5.94	7.24	6.19

```python
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
```

Gambar 4.22 Hasil Uji T ep_len_mean CliffWalking Menggunakan Python

Berdasarkan hasil perhitungan menghasilkan nilai **t** sebesar 8.66 dengan p-value 0,00008. Karena nilai p lebih kecil dari tingkat signifikansi0,05, maka perbedaan hasil rata-rata lamanya agen dalam satu episode antara metode A2C dan PPO dinyatakan signifikan secara statistik dalam *game CliffWalking*.

Tabel 4.4 Hasil ep_rew_mean Training Game CliffWalking

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
Deviasi	14.41	26.89	28.53	44.49	25.32	27.93

```python
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
```

Gambar 4.23 Hasil Uji T ep_rew_mean CliffWalking Menggunakan Python

Berdasarkan hasilperhitungan menghasilkan nilai sebesar -7,95 dengan p-value 0,00005. Karena nilai p lebih kecil dari tingkat signifikansi0,05, maka perbedaan hasil rata-rata *reward* per episode antara metode A2C dan PPO dinyatakan signifikan secara statistik. Hal ini menunjukkan bahwa PPO mendapatkan *reward* lebih banyak per episode daripada algoritma A2C dalam *game CliffWalking* dengan rata-rata deviasi sebanyak 27.93 reward.

BAB V
KESIMPULAN DAN SARAN

5.1 Kesimpulan

Berdasarkan hasilimplementasidan evaluasi yang telah dilakukan,dapat disimpulkan bahwa plugin *AgentRL* berhasil diimplementasikan pada *game CartPole* dan *CliffWalking* menggunakan dua algoritma pembelajaran penguatan, yaitu A2C (Advantage Actor-Critic) dan PPO (Proximal Policy Optimization). Hasil pelatihan menunjukkan bahwa pada game CartPole, algoritma PPO memberikan performa dalam menyeimbangkan tiang lebih baik dibandingkan A2C dengan rata-rata deviasi 115.93 lebih lama.

Sementara itu, pada game CliffWalking, kedua algoritma tidak menunjukkan perbedaan yang signifikan dalam hal panjang episode. Namun, terdapat perbedaan signifikan pada nilaireward per episode antara keduanya,di mana algoritma PPO memperoleh rata-rata deviasi durasi episode 6.19 lebih pendek dan deviasi reward 27.93 lebih banyak daripada A2C. Hal ini mengindikasikan bahwa PPO memilikiperforma yang lebih baik dibandingkan dengan A2C, terutama dalam *game CartPole* dan *CliffWalking*.

5.2 Saran

Penelitian ini dapat dikembangkan lebih lanjut dengan menerapkan pendekatan yang sama pada jenis permainan atau environment lainnya yang memiliki tingkat kompleksitas berbeda, seperti permainan berbasis strategi, navigasi dalam peta labirin, atau simulasi robotika. Hal ini bertujuan untuk menguji sejauh mana fleksibilitas *Godot*dan *plugin AgentRL* dalam menangani berbagai skenario *reinforcementlearning*. Selain itu, disarankan untuk menguji algoritma RL lainnya seperti *Deep Q-Network (DQN)* dan *Soft Actor-Critic (SAC)* guna membandingkan performa dan efisiensi masing-masing algoritma dalam menyelesaikan tugas yang lebih kompleks.

Daftar Pustaka

Barto, A. G., & Sutton, R. S. (1983). Neuronlike adaptive elements that can solve difficult learning control problems. http://incompleteideas.net/papers/barto-sutton-anderson-83.pdf

Colas, C., Sigaud, O., & Oudeyer, P. Y. (2019). A Hitchhiker's Guide to Statistical Comparisons of Reinforcement Learning Algorithms. https://arxiv.org/abs/1904.06979

Demšar, J. (2006). Statistical comparisons of classifiers over multiple data sets. Journal of Machine Learning Research. https://www.jmlr.org/papers/volume7/demsar06a/demsar06a.pdf

Diederichs, E. (2019). Reinforcement learning: A technical introduction – Part I. ResearchGate. https://www.researchgate.net/publication/335242067_Reinforcement_Learning_-_A_Technical_Introduction

Godot documentation – Latest stable release. https://docs.godotengine.org

Gosset, W. S. (1908). The probable error of a mean. Biometrika, 6(1). https://www.jstor.org/stable/2331554

Holfeld, J. (2023). On the relevance of the Godot Engine in the indie game development industry. arXiv preprint arXiv:2401.01909. https://www.researchgate.net/publication/383116776_On_the_relevance_of_the_Godot_Engine_in_the_indie_game_development_industry

Juliani, A., Berges, V. P., Vckay, E., Gao, Y., Henry, H., Mattar, M., & Lange, D. (2018). Unity: A general platform for intelligent agents. arXiv preprint arXiv:1809.02627. https://arxiv.org/pdf/1809.02627

Kumar, S. (2020). Balancing a CartPole system with reinforcement learning – A tutorial. arXiv preprint arXiv:2006.04938. https://arxiv.org/pdf/2006.04938

Millington, I., & Funge, J. (2009). Artificial intelligence for games (2nd ed.). CRC Press. https://spada.uns.ac.id/pluginfile.php/629724/mod_resource/content/1/gameng_AIFG.pdf

Mishra, S., & Arora, A. (2024, March 21). Balancing a Cart Pole using reinforcement learning in OpenAI Gym environment. Proceedings of the International Conference on Innovative Computing & Communication (ICICC 2024). SSRN. https://ssrn.com/abstract=4768234

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Harley, T., Lillicrap, T., Silver, D., & Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. arXiv preprint arXiv:1602.01783. https://arxiv.org/pdf/1602.01783

Montgomery,D. C., & Runger, G. C. (2014). Applied Statistics and Probability for Engineers. Wiley.

Nadimpalli, A. V., & Rattani, A. (2022). On improving cross-dataset generalization of deepfake detectors. ResearchGate. https://www.researchgate.net/publication/359881391_On_Improving_Cross-dataset_Generalization_of_Deepfake_Detectors

Raffin, A., Hill, A., Gleave, A., Kanervisto, A., Ernestus, M., & Dormann, N. (2021). Stable-Baselines3:Reliable reinforcementlearning implementations. Journal of Machine Learning Research, 22(268), 1–8. https://www.jmlr.org/papers/volume22/20-1364/20-1364.pdf

Rahim, A. A., Siregar, R. H., & Rahim, R. (2022). Performance comparison of reinforcement learning algorithms in the CartPole game using Unity ML-Agents. Journal of Theoretical and Applied Information Technology, 102(16), 218–227. https://www.jatit.org/volumes/Vol102No16/7Vol102No16.pdf

Ruxton, G. D. (2006). The unequal variance t-test is an underused alternative to Student's t-test and the Mann–Whitney U test. Behavioral Ecology.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal policy optimization algorithms.arXiv preprint arXiv:1707.06347. https://arxiv.org/pdf/1707.06347

Surriani, A., & Santoso, B. (2021). Reinforcement learning for cart pole inverted pendulum system. ResearchGate. https://www.researchgate.net/profile/Atikah-Surriani/publication/357438224_Reinforcement_Learning_for_Cart_Pole_Inverted_Pendulum_System

Sutton, R. S., & Barto, A. G. (2018). Reinforcementlearning: An introduction (2nd ed.). MIT Press. https://web.stanford.edu/class/psych209/Readings/SuttonBartoIPRLBook2ndEd.pdf

Torrado, R. R., Royo, D. M., Cabañes, A. G., Gómez, J. A. M., Martínez, H. R., & Julián, V. (2021). Godot reinforcement learning agents. arXiv preprint arXiv:2112.03636. https://arxiv.org/pdf/2112.03636

Towers, M., Gao, Z., Krummenacher,J., Kumar, P., & Schulman, J. (2024). Gymnasium: A standard interface for reinforcementlearning environments. arXiv preprint arXiv:2407.17032. https://doi.org/10.48550/arXiv.2407.17032

Wang, D., & Snooks, R. (2021). Artificial intuitions of generative design: An approach based on reinforcementlearning. Dalam Proceedings of the 2020 DigitalFUTURES (hal. 189–198). Springer Singapore. https://doi.org/10.1007/978-981-33-4400-6_18

Welch, B. L. (1947). The generalization of Student's problem when several different population variances are involved. Biometrika.

Wu, Y., Tucker, G., & Nachum, O. (2022). A2C is a special case of PPO. arXiv preprint arXiv:2205.09123. https://arxiv.org/pdf/2205.09123

Zhong, L. (2023). Comparison of Q-learning and SARSA reinforcement learning models on Cliff Walking problem.In Proceedings of DAI-23. Atlantis Press. https://www.atlantis-press.com/proceedings/dai-23/125998063