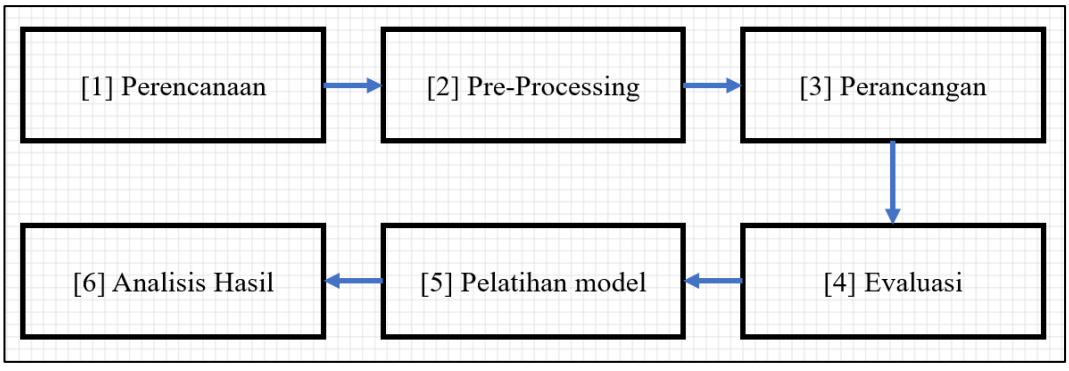
3. PERANCANGAN DAN IMPLEMENTASI

Bab ini menjelaskan perancangan program pelatihan agen *Deep Reinforcement Learning* menggunakan algoritma *Proximal Policy Optimization* (*PPO*) dan arsitektur *Convolutional Neural Network* (*CNN*) pada berbagai lingkungan permainan berbasis *OpenAI Gym*. Tujuan dari program ini adalah untuk mengevaluasi performa *PPO* secara umum, tanpa melakukan *hyperparameter tuning* maupun perubahan *reward shaping*, dalam penelitian dari berbagai jenis lingkungan dengan masing-masing karakteristik yang berbeda.

Program ini dibangun menggunakan bahasa pemrograman *Python*, dengan pustaka *Stable-Baselines3* sebagai alat utama untuk algoritma *PPO*. Evaluasi dilakukan terhadap lima lingkungan permainan: *Breakout*, *Space Invaders*, *Pong*, *Flappy Bird*, dan *Super Mario Bros*. Semua lingkungan permainan dipilih berdasarkan keragaman visual dan kompleksitas strategi, untuk menguji seberapa baik PPO dapat beradaptasi dalam berbagai skenario permainan. Proses pelatihan dilakukan hingga 5 juta langkah untuk masing-masing lingkungan, menggunakan *Vectorized Environments* yang berguna untuk mempercepat proses**.**



Gambar 3. 1 Alur Diagram Blok

Gambar 3.1 menunjukkan alur kerja sistem pelatihan agen reinforcement learning menggunakan metode *Proximal Policy Optimization* (PPO) pada berbagai environment permainan. Diagram ini terdiri dari enam tahap utama, dimulai dari pembuatan environment hingga analisis hasil pelatihan model.

1. *Perencanaan,* Tahap awal dimulai dengan perencanaan sistem pelatihan agen. Pada tahap ini, dilakukan pemilihan *environment* permainan yang akan digunakan, seperti *Space Invaders*, *Breakout*, *Pong*, *Flappy Bird*, dan *Super Mario Bros*. Selain itu, ditentukan pula tujuan pelatihan, metode yang digunakan, serta perangkat keras dan perangkat lunak yang dibutuhkan.
2. *Pre-Processing,* Observasi visual dari *environment* kemudian diproses terlebih dahulu agar lebih efisien untuk dipelajari oleh agen. Proses ini mencakup resize gambar ke resolusi tertentu, konversi ke grayscale, dan *frame stacking* untuk memberikan konteks temporal dari pergerakan objek di game.
3. Perancangan, Setelah data observasi diproses, tahap ini melibatkan perancangan arsitektur model yang akan digunakan oleh agen. Model yang dirancang berbasis *Convolutional Neural Network* (*CNN*) yang mampu mengenali pola visual dari observasi game.
4. Evaluasi, Selama proses pelatihan berlangsung, model dievaluasi secara berkala menggunakan *Tensorboard* dan *Stable-Baselines3* untuk memantau performa seperti reward rata-rata dan panjang episode. Selain itu, *CheckpointCallback* digunakan untuk menyimpan model terbaik selama pelatihan agar dapat dibandingkan atau digunakan kembali.
5. Pelatihan Model, Pada tahap ini, agen dilatih menggunakan metode PPO. Model akan menerima observasi dari environment, memilih aksi, dan menerima reward sebagai umpan balik. Pelatihan dilakukan melalui interaksi berulang antara agen dan environment.
6. Analisis Hasil, Setelah proses pelatihan selesai, model akhir disimpan dan dianalisis. Analisis ini mencakup kinerja model pada environment tertentu, seberapa baik agen beradaptasi, serta perbandingan reward antar environment. Tahap ini juga dapat digunakan untuk mengidentifikasi potensi peningkatan pada preprocessing atau arsitektur model.

3.1 Perencanaan

Tahap awal dalam penelitian ini dimulai dengan proses perencanaan sistem pelatihan agen *deep reinforcement learning*. Perencanaan mencakup penentuan *environment* permainan yang akan digunakan sebagai studi kasus, yaitu: *Space Invaders, Breakout, Pong, Flappy Bird,* dan *Super Mario Bros*. Game-game ini dipilih karena mewakili beragam dinamika visual dan tantangan strategi yang dapat menguji kemampuan generalisasi agen.

Metode pelatihan yang digunakan adalah *Proximal Policy Optimization* (*PPO*) berbasis *Convolutional Neural Network* (*CNN*), dengan implementasi melalui pustaka *Stable-Baselines3*. Tahap ini juga mencakup penetapan spesifikasi perangkat lunak dan perangkat keras yang diperlukan untuk mendukung proses pelatihan dan evaluasi model secara efisien.

3.1.1. Analisis Perangkat Lunak

Beberapa perangkat lunak yang dibutuhkan dalam pembangunan dan pelatihan agen PPO antara lain:

- Sistem Operasi: Windows 10
- Bahasa Pemrograman: Python 3.11.9
- Library dan Framework:
    - Stable-Baselines3 untuk implementasi algoritma PPO
    - Gymnasium, ALE-py, gym-super-mario-bros, dan gym-Flappy-Bird untuk lingkungan permainan
    - PyTorch dan Numpy sebagai backend utama untuk model CNN
    - Matplotlib, Tensorflow untuk visualisasi hasil pelatihan
- Editor/IDE: Visual Studio Code

3.1.2. Analisis Perangkat Keras

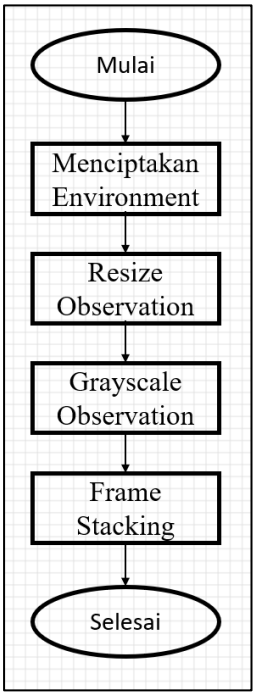
Untuk mendukung proses pelatihan yang intensif secara komputasi, berikut adalah contoh spesifikasi perangkat keras:

- CPU: AMD Ryzen 7 5700X
- GPU: NVIDIA RTX 4070 SUPER

- RAM: 32 GB
- Storage: SSD 500 GB untuk menyimpan data model, log pelatihan, dan hasil visualisasi
- Koneksi Internet: Stabil untuk instalasi pustaka dan dependensi

3.2 Pre-processing

Untuk menyederhanakan gambar yang kompleks dari *environment* dan menjaga konsistensi input antar *environment*, dilakukan proses *preprocessing* sebelum data visual diberikan ke agen. Hyperparameter dijalankan sesuai pengaturan bawaan dari *Stable-Baselines3* seperti yang ditunjukan pada Gambar 3.2.



Gambar 3. 2 Alur Preprocesing

3.2.1 Pre-Processing Model

Gambar 3.2 menggambarkan proses yang bertujuan untuk mengurangi dimensi input, menghilangkan informasi warna yang tidak relevan, serta memberikan konteks temporal agar agen dapat memahami pergerakan objek. Setiap environment dibungkus dalam serangkaian transformasi berikut:

- *ResizeObservation* menjadi 84×84

- *GrayscaleObservation* untuk mengubah warna ke skala abu-abu
- *FrameStack* untuk menyatukan 4 *frame* terakhir

```
env = ResizeObservation(env, (84, 84))
env = GrayscaleObservation(env, keep_dim=True)
env = FrameStack(env, num_stack=4)
```

Proses ini dilakukan secara modular di dalam fungsi pembuat environment dan digunakan oleh seluruh lingkungan.

3.2.2 *Hyperparameter Default*

Dalam pelatihan agen, terdapat sejumlah *hyperparameter* bawaan yang berasal dari konfigurasi bawaan *Gymnasium*. Beberapa parameter penting dijelaskan sebagai berikut:

- Observation Type: Secara bawaan, lingkungan ini menghasilkan observasi dalam bentuk citra berwarna dengan dimensi penuh. Pada penelitian ini observasi tersebut diubah melalui proses *preprocessing* menjadi citra *grayscale* berukuran (84, 84).
- Frameskip: Nilai bawaan dari parameter *frameskip* adalah 5. Nilai ini tidak diubah dalam penelitian, sehingga tetap menggunakan konfigurasi default.
- Repeat Action Probability: Lingkungan ini memiliki probabilitas bawaan sebesar 0.25. Nilai ini tidak diubah dan tetap digunakan sebagaimana pengaturan aslinya.

Algoritma *Proximal Policy Optimization* (*PPO*) memiliki beberapa parameter inti yang memengaruhi proses pelatihan. Dalam implementasi ini, semua hyperparameter dibiarkan pada nilai default sebagaimana ditentukan oleh pustaka Stable-Baselines3, yaitu:

- Learning Rate: 0.0003
- Clip Range: 0.2
- Batch Size: 64
- Number of Steps (n_steps): 2048
- Number of Epochs (n_epochs): 10

- Discount Factor (gamma): 0.99
- GAE Lambda (gae_lambda): 0.95
- Entropy Coefficient (ent_coef): 0.0
- Value Function Coefficient (vf_coef): 0.5
- Max Gradient Norm (max_grad_norm): 0.5
- Advantage Normalization: True
- Target KL Divergence (target_kl): None (tidak diaktifkan secara default)

Dengan demikian, kecuali untuk observasi yang diproses ulang menjadi *grayscale* dan perubahan pada ukurannya, semua hyperparameter lingkungan lainnya tetap mengacu pada pengaturan bawaan. Hal ini dilakukan untuk menjaga fokus penelitian pada pengujian metode *PPO* tanpa memodifikasi konfigurasi lingkungan secara signifikan.

3.3 Perancangan

Dalam penulisan ini, arsitektur agen menggunakan *Convolutional Neural Network* (CNN) yang diimplementasikan melalui pustaka *Stable-Baselines3* dengan metode *Proximal Policy Optimization* (*PPO*). *CNN* dipilih karena kemampuannya dalam mengekstrak fitur visual dari input gambar secara efisien, sementara *PPO* dikenal stabil dan efektif dalam melatih agen pada berbagai jenis environment.

Arsitektur *CNN* yang digunakan bersifat ringan namun cukup dalam untuk menangani input visual yang sudah disesuaikan berdasarkan ukuran serta ketentuan yang sudah dijelaskan sebelumnya.

3.3.1 Inisialisasi *Environment*

Setiap *environment* dalam penelitian ini diinisialisasi menggunakan sebuah program yang dirancang khusus agar kompatibel dengan arsitektur *PPO* dan dapat digunakan secara paralel oleh *SubprocVecEnv.*

Contoh inisialisasi dasar *environment*:

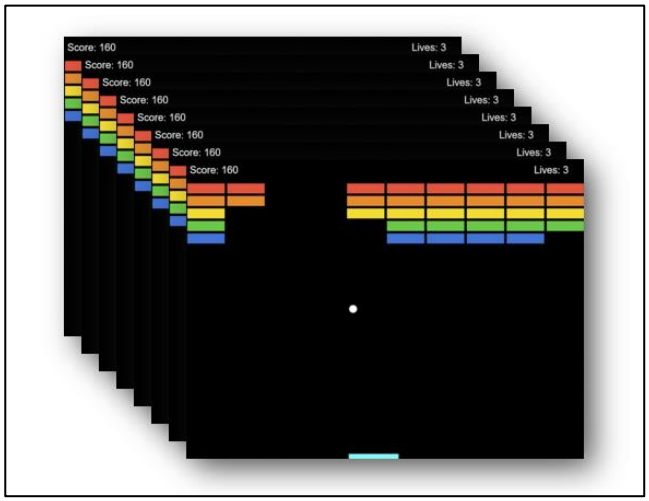
```
gym.register_envs(ale_py)
def make_env():
    env = gym.make("ALE/Breakout-v5")
    env = ResizeObservation(env, (84, 84))
    env = GrayscaleObservation(env, keep_dim=True)
    return env
```

3.3.2 Inisialisasi *PPO* Dan Kebijakan *CNN* (*CnnPolicy*)

Model *PPO* diinisialisasi menggunakan *CnnPolicy*. Arsitektur agen dibangun menggunakan kebijakan berbasis *CNN* yang telah disediakan oleh *CnnPolicy* dari pustaka *Stable-Baselines3*. Kebijakan ini secara default terdiri dari tiga lapisan konvolusi (convolutional layer) yang diikuti oleh satu lapisan teruhubung.

```
        model = PPO(
            policy="CnnPolicy",
            env=train_env,
          )
```

3.3.3 Vectorized Environment



Gambar 3. 3 Visualiasi Vectorized Environments

Gambar 3.3 menunjukan visualisasi *Vectorized* environments yang bertujuan untuk meningkatkan efisiensi pelatihan, yang mengunakan *SubprocVecEnv* dari pustaka *Stable-Baselines3* yang memungkinkan pelatihan pada beberapa environment secara paralel. Dalam implementasi ini, jumlah environment paralel yang digunakan adalah sebanyak 8. Setiap environment dibuat menggunakan fungsi make_train_env().

```
num_envs = 8
env_fns = [make_train_env() for _ in range(num_envs)]

train_env = SubprocVecEnv(env_fns)
```

Langkah ini memungkinkan agen untuk mengumpulkan lebih banyak data dari beberapa penjalanan agen secara sekaligus, sehingga mempercepat proses pelatihan.

3.3.4 VecFrameStack

Lingkungan ditambahkan dengan *VecFrameStack* untuk melakukan *frame stacking*, yaitu teknik menyatukan beberapa frame terakhir menjadi satu input. Teknik ini penting untuk memberikan informasi kepada agen, terutama pada game yang berbasis visual.

```
train_env = VecFrameStack(train_env, n_stack=4)
```

Dengan *n_stack*=4, berarti setiap input yang diterima agen merupakan gabungan dari 4 frame terakhir.

3.4 Evaluasi

Proses logging selama pelatihan menggunakan *TensorBoard* agar metrik seperti reward, loss dapat dimonitor secara langsung, Dan menggunakan fitur bawaan dari *Stable-Baselines3* lainnya. Serta demi proses pelatihan berjalan secara terkontrol dan hasil model dapat tersimpan secara berkala, digunakan beberapa callback dari pustaka Stable-Baselines3. Fungsi utama callback adalah untuk

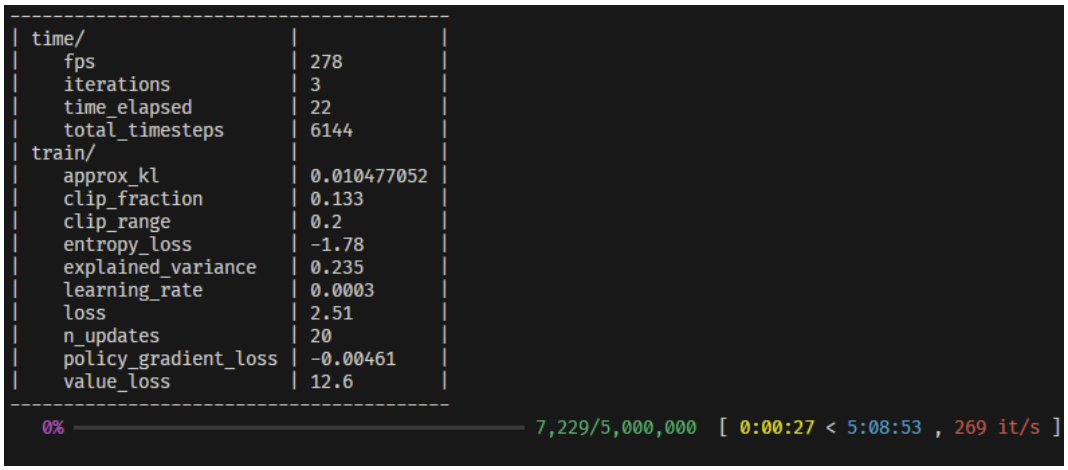
memantau performa model dan menyimpan checkpoint secara otomatis tanpa perlu intervensi manual.

3.4.1 Tensorboard

Lingkungan dijalankan dengan memanfaatkan fitur fitur dalam *tensorboard* untuk mencari hasil penelitian ini. Terutama pada *Reward* rata-rata, dan rata-rata lama ketahanan agen. Yang kemudian akan dibandingkan perkembangan nilai selama agen terus berjalan.

```
Model = PPO(tensorboard_log=log_dir)
```

3.4.2 Monitoring



Gambar 3. 4 Monitoring

Gambar 3.4 menggambarkan penampilan proses pelatihan, yang diawasi secara langsung melalui *verbose*=1 pada inisialisasi model *PPO*. Parameter ini mengaktifkan *logging* sederhana yang menampilkan informasi progres pelatihan.

Selain itu, *parameter progress_bar=True* pada metode *learn*() menambahkan visualisasi progress bar untuk memudahkan pemantauan sisa waktu pelatihan dan total langkah yang telah ditentukan.

```
model = PPO(verbose=1)
model.learn(
    total_timesteps=5_000_000,
    progress_bar=True
```

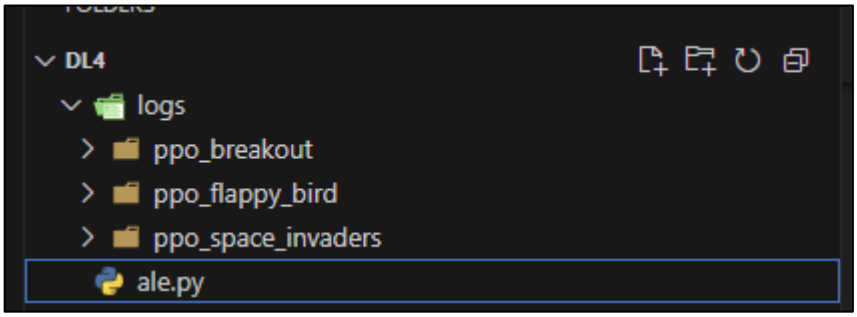
```
)
```

3.4.3 Callback

Callback ini digunakan untuk mengevaluasi performa agen secara berkala selama proses pelatihan. *CheckpointCallback* digunakan untuk menyimpan model secara rutin setiap interval tertentu.

```
model.learn(callback=CheckpointCallback(100_000//num_envs,
os.path.join(log_dir, "checkpoints"), "ppo_checkpoint"))
```

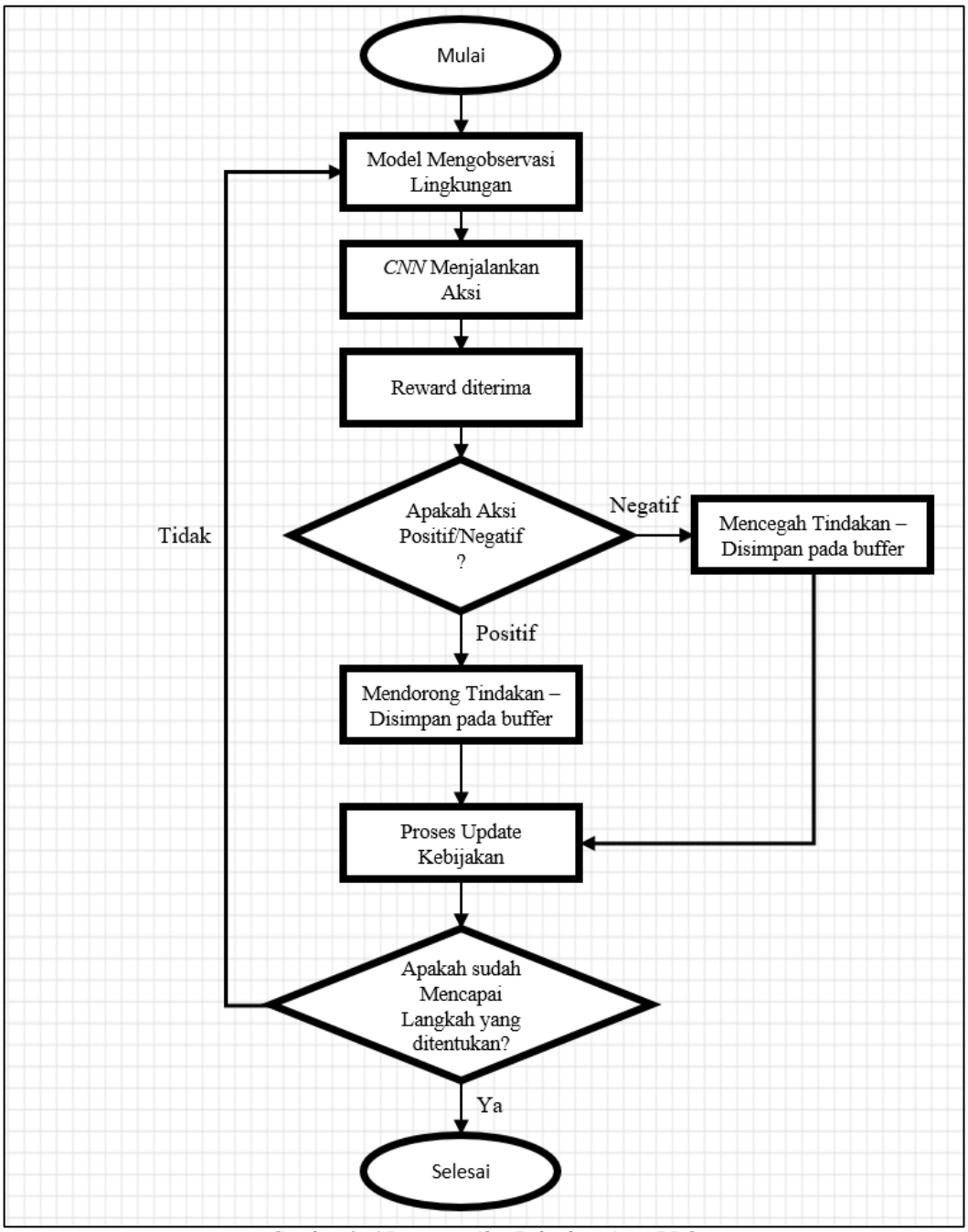
3.4.4 Struktur Direktori *Log*



Gambar 3. 5 Contoh Struktur Direktori

Selama pelatihan, data *log* dan *checkpoint* disimpan ke dalam struktur *folder* yang terpisah pada tiap *Environment*, seperti pada gambar 3.5. Hal ini bertujuan untuk menjaga kerapian dan memudahkan analisis hasil setiap eksperimen.

3.5 Pelatihan Model



Gambar 3. 6 Diagram Alur Pelaithan Agen PPO

Setelah semua komponen sistem dirancang, proses pelatihan agen dilakukan dengan mengintegrasikan seluruh elemen yang telah disiapkan, seperti preprocessing, inisialisasi environment, arsitektur *CNN*, serta *vectorized environment*. Pelatihan dilakukan menggunakan metode *Proximal Policy*

Optimization (PPO) yang bekerja secara iteratif berdasarkan interaksi agen dengan *environment*.

Agen memulai proses pelatihan dengan mengamati *environment* melalui data visual yang telah diproses. Input ini terdiri dari *grayscale* berukuran 84×84 piksel dan yang sudah diaplikasikan *frame stacking*, sehingga mencerminkan dinamika visual dari lingkungan permainan.

Berdasarkan observasi tersebut, model menghasilkan aksi melalui kebijakan berbasis *CNN*. Aksi ini kemudian dieksekusi oleh *environment*, yang akan membalas dengan *reward* serta *state* terbaru. *Reward* yang diterima tidak langsung diklasifikasikan sebagai "baik" atau "buruk", melainkan dikumpulkan dalam buffer selama beberapa langkah, untuk kemudian digunakan dalam proses optimasi.

Setelah sejumlah interaksi terkumpul, model menjalankan proses update kebijakan menggunakan algoritma *PPO*. Proses ini memperbarui parameter model dengan memaksimalkan fungsi objektif *PPO* yang mempertimbangkan *reward* kumulatif serta stabilitas perubahan kebijakan. Update ini dilakukan dalam *batch* yang besar dari pengalaman yang telah dikumpulkan.

Selama proses pelatihan, sistem melakukan monitoring kinerja menggunakan *TensorBoard* untuk mencatat metrik seperti *reward* rata-rata, *loss*, dan *entropy*. Selain itu, digunakan juga *callback* untuk menyimpan checkpoint model secara otomatis dan berkala. Hal tersebut memastikan bahwa hasil pelatihan tidak hilang jika pelatihan terganggu serta memungkinkan analisis performa pada berbagai tahapan pelatihan.

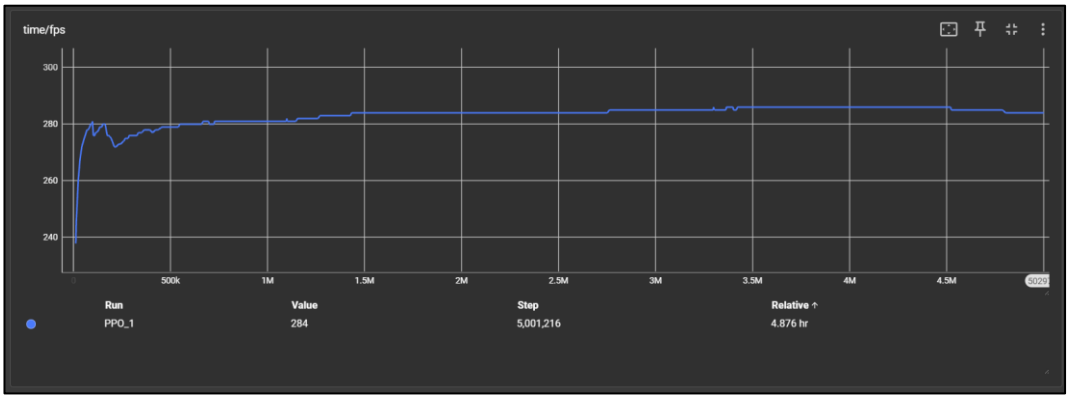
3.6 Hasil Analisis

Tahap ini membahas hasil dari proses pelatihan agen menggunakan metode *Proximal Policy Optimization* (*PPO*) dengan arsitektur *CNN* pada berbagai *environment* visual. Evaluasi dilakukan setiap 800.000 langkah hingga total 5 juta langkah Perubahan nilai reward dari setiap checkpoint dibandingkan dalam bentuk persentase perubahan untuk merepresentasikan nilai efektivitas pembelajaran model. Selain itu, analisis dilakukan terhadap dua aspek utama: efektivitas

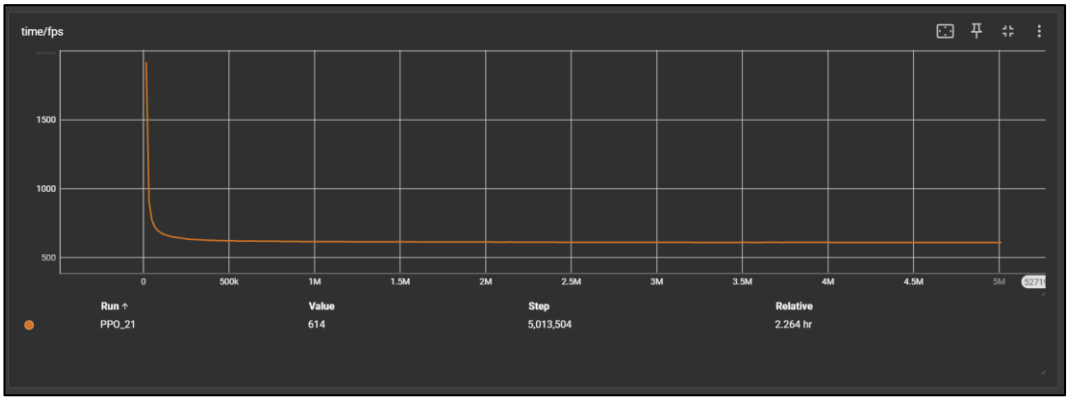
penggunaan *vectorized environment* (SubprocVecEnv) dan perbandingan performa agen antar *game*. Hasil disajikan dalam bentuk evaluasi numerik, visualisasi grafik, serta interpretasi tren berdasarkan pertumbuhan reward dan durasi episode.

3.6.1 Analisis Pengaruh *Vectorized Environment*

Penggunaan *vectorized environment* menunjukkan dampak signifikan terhadap efisiensi waktu pelatihan agen. Dengan menjalankan beberapa environment secara paralel menggunakan *SubprocVecEnv*, waktu total pelatihan berhasil dikurangi secara drastis dibandingkan pelatihan dengan environment standar.



Gambar 3. 7 Grafik waktu tanpa vectorized environment



Gambar 3. 8 Grafik waktu dengan vectorized environment

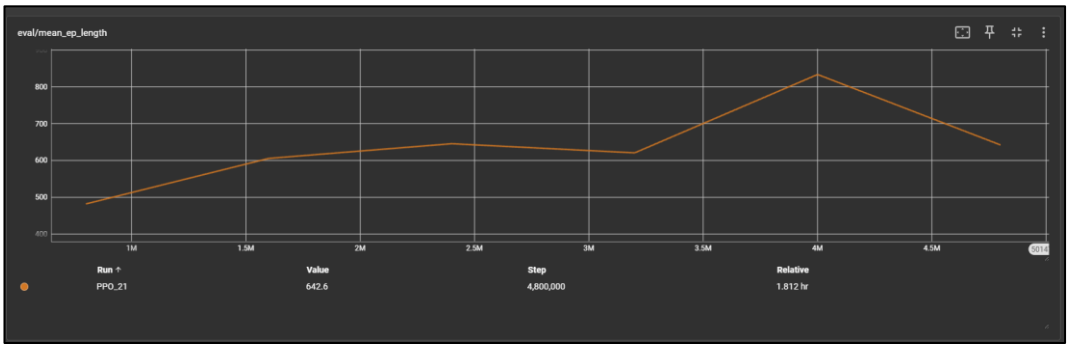
Dalam Gambar 3.7 serta 3.8, pelatihan dengan *vectorized environment* menggunakan 8 *environment paralel* memakan waktu sekitar 2.264 jam (2 jam, 15 menit, dan 50 detik) sedangkan pelatihan tanpa vectorized environment membutuhkan waktu hingga 4.789 jam (4 jam, 52 menit, 33 detik) untuk

menyelesaikan jumlah langkah yang sama, yaitu 5 juta langkah. Hal ini menandakan adanya penghematan waktu pelatihan sebesar 52,71% atau 2 jam, 36 menit, dan 43 detik, yang membuktikan bahwa *vectorization* mempercepat proses secara signifikan.

3.6.2 Perbandingan Performa Antar *Game*

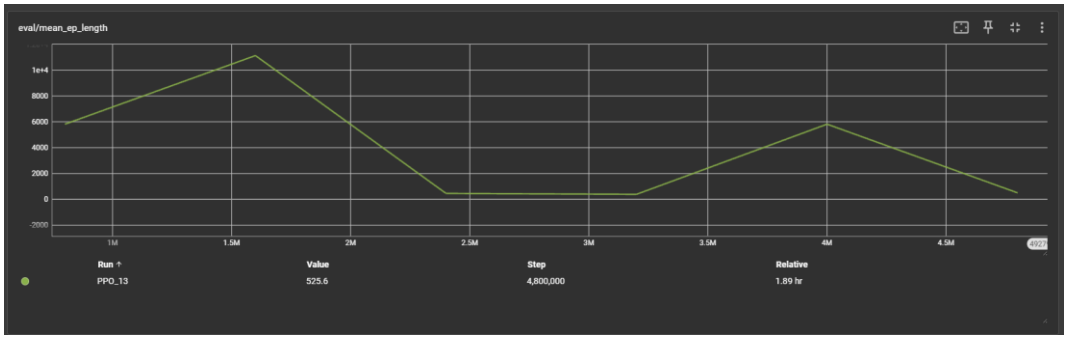
Pendekatan ini dipilih karena rentang *reward* dari setiap *environment* berbeda secara signifikan, terutama akibat tidak dilakukan *hyperparameter tuning* maupun *reward scaling*. Dengan demikian, meskipun nilai absolut *reward* antar *environment* tidak sebanding, performa agen tetap dapat dibandingkan secara lebih seimbang berdasarkan persentase tingkat peningkatannya.

3.6.3 Grafik *Length*



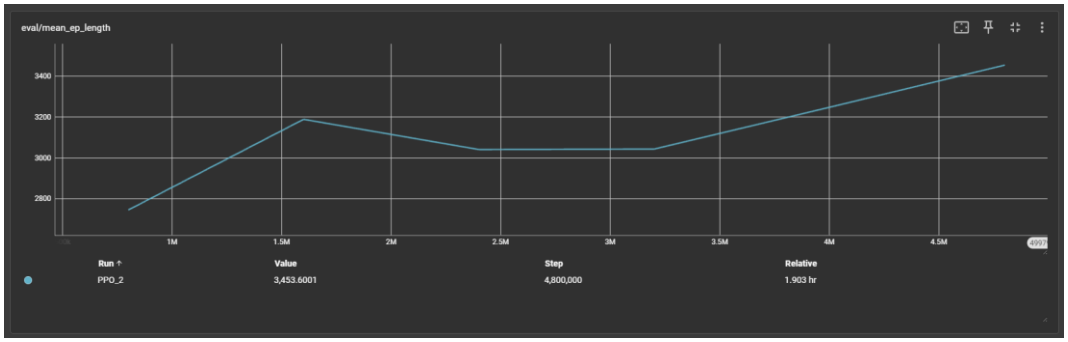
Gambar 3. 9 Grafik mean_ep_length Space Invaders

Gambar 3.9 merupakan grafik pada *game Space Invaders*, nilai rata-rata durasi *environment* meningkat dari 482.8 pada evaluasi awal menjadi 833.6 pada langkah ke-4.000.000, menghasilkan peningkatan sebesar +72,66%. Sedangkan pada evaluasi akhir (langkah ke-4.800.000), nilainya berada di angka 642.6, atau perubahan +33,10% dibandingkan nilai awal.



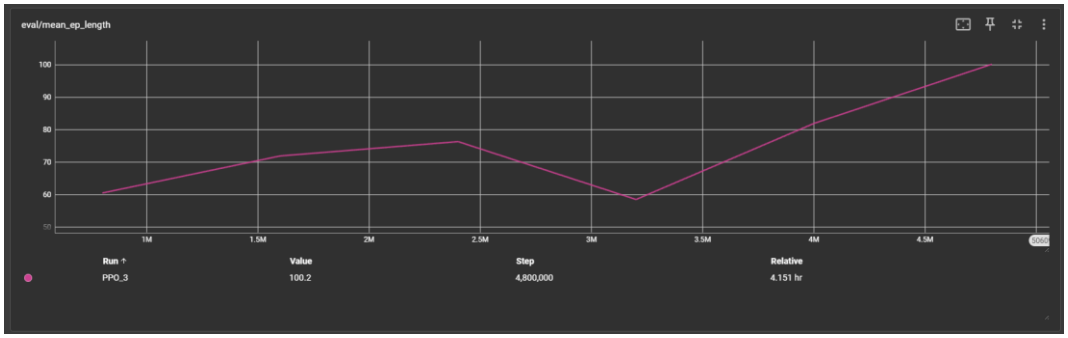
Gambar 3. 10 Grafik mean_ep_length Breakout

Pada gambar 3.10 yang menunjukkan nilai rata-rata durasi untuk *Breakout*, terlihat bahwa performa model mengalami degradasi atau penurunan seiring waktu. Nilai awal sebesar 5828 secara konsisten menurun, mencapai titik terendah 412.4 pada langkah ke-3.200.000. Sampai evaluasi akhir pada langkah ke-4.800.000, nilai episode length hanya mencapai 525.6, yang berarti total penurunan sebesar -90,98% dari nilai awal. Penurunan tajam ini mengindikasikan bahwa model semakin gagal mempertahankan strategi permainan yang stabil dan efektif seiring bertambahnya jumlah langkah pelatihan.



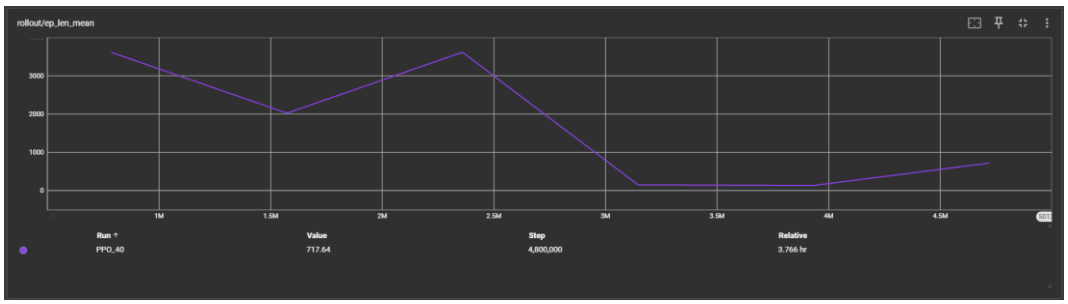
Gambar 3. 11 Grafik mean_ep_length Pong

Gambar 3.11 menunjukan grafik untuk game *Pong*, dengan nilai awal sebesar 2747 meningkat menjadi 3454 pada evaluasi akhir (langkah ke-4.800.000), menunjukkan kenaikan sebesar +25,74%.



Gambar 3. 12 Grafik mean_ep_length Flappy Bird

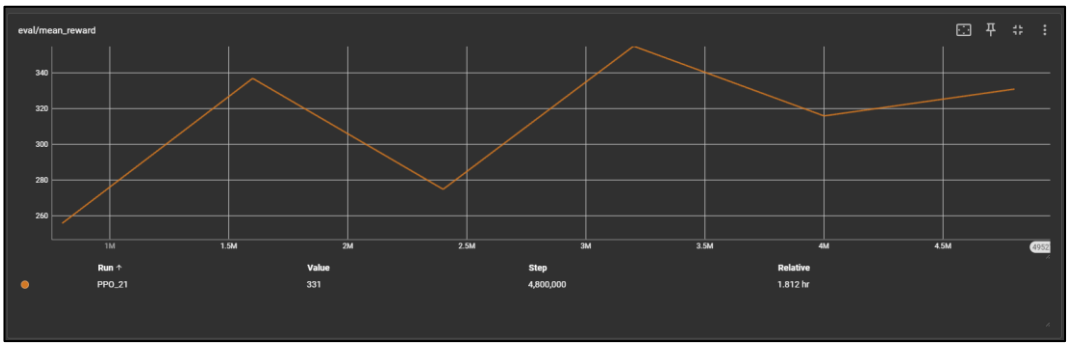
Gambar 3.12 menunjukkan game *Flappy Bird*, nilai awal sebesar 60.6 meningkat secara konsisten hingga mencapai 100.2 pada evaluasi akhir, yang berarti peningkatan sebesar +65,35% dari nilai awal.



Gambar 3. 13 Grafik mean_ep_length Super Mario Bros

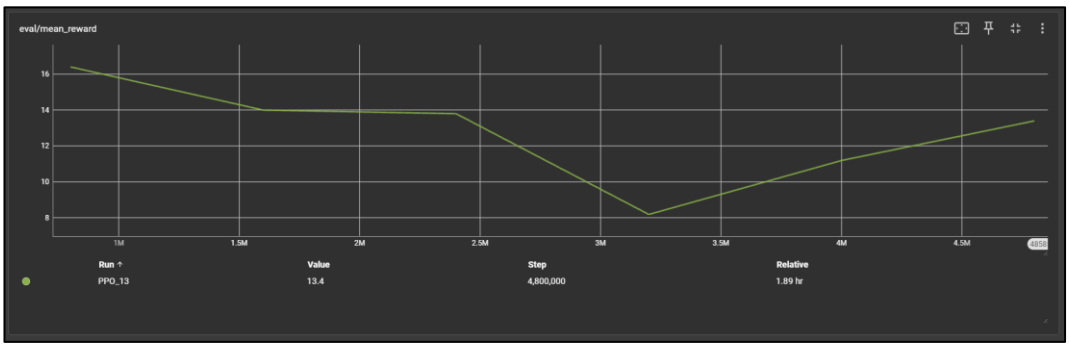
Pada gambar 3.13, terdapat grafik *Super Mario Bros*, dengan nilai awal sebesar 3610.49, mengalami naik turun, hingga lalu mengalami penurunan drastis mencapai 132.0. Penurunan sebesar -96.34%

3.6.4 Grafik *Reward*



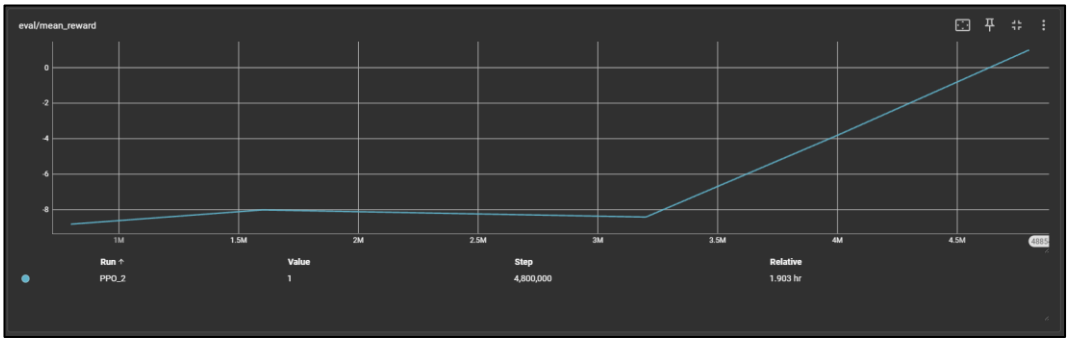
Gambar 3. 14 Grafik mean_reward Space invaders

Pada gambar 3.14, game *Space Invaders* menggambarkan *reward* rata-rata yang diperoleh agen meningkat dari 256.0 pada awal pelatihan menjadi 355.0 sebagai nilai tertinggi, yang dicapai sebelum evaluasi akhir. Meskipun pada evaluasi akhir reward sedikit menurun ke 331.0, tren ini tetap mencerminkan peningkatan sebesar +38,67% dari nilai awal, menandakan bahwa agen berhasil mengembangkan strategi yang lebih efektif seiring bertambahnya waktu pelatihan.



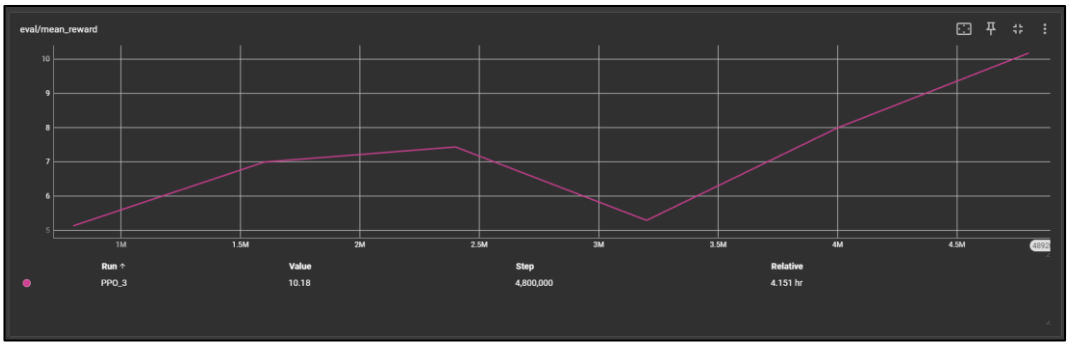
Gambar 3. 15 Grafik mean_reward Breakout

Pada gambar 3.15 performa *reward* agen pada game *Breakout* mengalami penurunan. Nilai awal *reward* sebesar 16.4 turun ke titik terendah 8.2 selama proses pelatihan, dan hanya mencapai 13.4 pada evaluasi akhir. Hal ini menunjukkan adanya penurunan -50.00% dari nilai awal terhadap nilai terburuk yang dicapai, mencerminkan bahwa strategi agen menjadi semakin tidak efektif dalam mendapatkan *reward* di lingkungan ini.



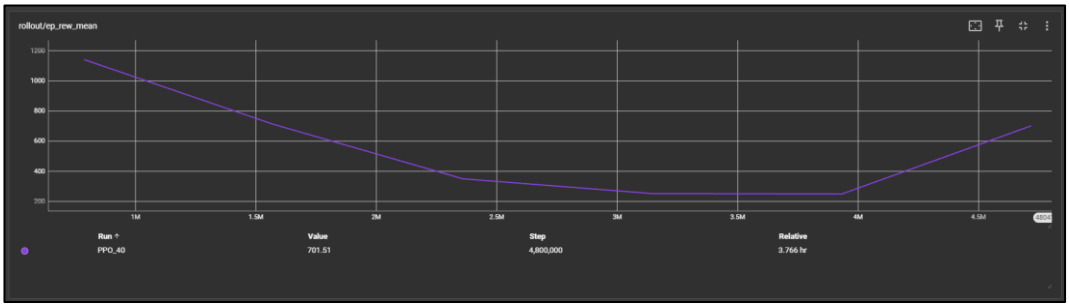
Gambar 3. 16 Grafik mean_reward Pong

Pada gambar 3.16, menunjukkan game *Pong* dengan *reward* awal dimulai dari angka negatif yakni -8.8. Selama pelatihan, performa meningkat drastis hingga mencapai nilai maksimum 1.0 yang juga menjadi nilai pada evaluasi akhir. Dengan demikian, terjadi peningkatan sebesar +111,36%, menandakan perbaikan performa signifikan dari agen yang awalnya selalu kalah menjadi mampu memenangkan pertandingan dalam sebagian besar episode.



Gambar 3. 17 Grafik mean_reward Flappy Bird

Gambar 3.17 mengambarkan *Reward* rata-rata pada game *Flappy Bird* menunjukkan tren positif yang konsisten. Agen memulai dengan reward 5.14 dan meningkat secara bertahap hingga mencapai 10.18 pada akhir pelatihan. Peningkatan sebesar +98.05% ini menunjukkan bahwa agen semakin mahir dalam mempertahankan kelangsungan hidup dan menghindari rintangan secara efisien.



Gambar 3. 18 Grafik mean_reward Super Mario Bros

Gambar 3.18 menggambarkan game *Super Mario Bros*, yang menunnjukan nilai awal sebesar 3610.49, mengalami penurunan hingga lalu mengalami penurunan mencapai 249.0. Terjadi Penurunan sebesar -78.17%.

3.6.5 Tabel Perbandingan

Tabel 3. 1 Mean Episode Length

No	Nama Environment	Nilai awal	Nilai akhir	Nilai terendah	Nilai terbesar	Nilai persentase Perbedaan diantara nilai awal terhadap nilai dengan nilai signifikan
1	Space Invaders	482.8	642.6	482.8	833.6	+72.66%
2	Breakout	5258.0	525.6	412.4	11123.0	-92.92%
3	Pong	2747.0	3454.0	2747.0	3454.0	+25.74%
4	Flappy Bird	60.6	100.2	58.6	100.2	+65.35%
5	Super Mario Bros	3610.49	717.64	132.0	3614.77	-96.34%

Tabel 3. 2 Mean Reward

No	Nama Environment	Nilai awal	Nilai akhir	Nilai terendah	Nilai terbesar	Nilai persentase Perbedaan diantara nilai awal terhadap nilai dengan nilai signifikan
1	Space Invaders	256.0	331.0	256.0	355.0	+38.67%
2	Breakout	16.4	13.4	8.2	16.4	-50.00%
3	Pong	-8.8	1.0	-8.8	1.0	+111.36%
4	Flappy Bird	5.14	10.18	5.14	10.18	+98.05%
5	Super Mario Bros	1140.65	701.51	249.0	1140.65	-78.17%

Berdasarkan hasil evaluasi pada Tabel 3.1 serta 3.2, hasil metrik mean episode length dan mean reward yang dilakukan setiap 800.000 langkah hingga total 5 juta langkah, diperoleh hasil performa yang bervariasi tergantung pada karakteristik masing-masing *environment*. Interpretasi berikut menggabungkan kedua metrik untuk memberikan gambaran yang lebih utuh terhadap kemampuan agen dalam setiap game.

3.6.6 *Space Invaders*

Agen menunjukkan peningkatan bertahap dalam durasi *environment* (episode length meningkat dari 482.8 ke 833.6) dan akumulasi skor (reward meningkat dari 256.0 ke 355.0). Hal ini menunjukkan bahwa agen tidak hanya

mampu bertahan lebih lama tetapi juga semakin efektif dalam menyerang dan mencetak poin, yang merupakan tanda pembelajaran yang stabil.

3.6.7 *Breakout*

Performa agen Menurun dalam hal efisiensi permainan, ditunjukkan oleh penurunan episode length drastis dari 5828 ke 412.4. Serta, *reward* mengalami penurunan dari 16.4 ke 8.2. Ini menandakan bahwa agen *PPO* kurang efektif pada *Environment Breakout*, efektivitas dalam mencetak skor belum optimal kemungkinan karena strategi terlalu agresif atau kurang presisi dalam menghancurkan blok.

3.6.8 *Pong*

Agen mengalami perbaikan performa signifikan, dari reward negatif (-8.8) ke positif (1.0), yang mengindikasikan perubahan dari sering kalah menjadi lebih sering menang. Ini juga didukung oleh peningkatan episode length dari 2747 ke 3454, menunjukkan permainan yang lebih seimbang dan kompetitif.

3.6.9 *Flappy Bird*

Durasi rata-rata meningkat dari 60.6 ke 100.2, dan reward naik dari 5.14 ke 10.18. Karena *reward* dalam environment ini berkorelasi langsung dengan durasi rata-rata *Environment*, kedua metrik saling memperkuat dan menegaskan bahwa agen berhasil belajar menghindari rintangan secara konsisten.

3.6.10 *Super Mario Bros*

Agen menunjukkan penurunan performa yang signifikan, dengan *episode length* turun drastis dari 3610.49 ke 132.0 dan reward menurun dari 1140.65 ke 249.0. Meskipun sempat mencapai puncak reward 3614.77, ketidakstabilan ini mengindikasikan agen gagal beradaptasi dengan kompleksitas lingkungan.

3.6.11 Inti Analisis

Hasil evaluasi menunjukkan bahwa efektivitas pelatihan agen sangat bergantung pada karakteristik tiap environment. Pong mencatat peningkatan tertinggi secara konsisten, baik dari sisi reward (+111.36%) maupun episode length (+25.74%), mengindikasikan kemampuan agen dalam memahami pola permainan. Flappy Bird juga menunjukkan hasil kuat di kedua metrik, sedangkan Space Invaders memiliki durasi episode tinggi namun reward yang hanya naik moderat (+38.67%). Sebaliknya, Breakout dan Super Mario Bros menunjukkan tren negatif di kedua sisi. Dengan Breakout menghasilkan penurunan nilai episode length hingga -92.92% dan reward sebesar -50%, dan Super Mario Bros dengan penurunan sebesar -96.34% dan -78.17%. Hal ini menandakan bahwa panjang durasi tidak selalu berbanding lurus dengan performa, terutama pada game dengan struktur reward dan dinamika unik. Oleh karena itu, interpretasi data memerlukan pemahaman kontekstual dari tiap environment, bukan hanya berdasarkan angka absolut.

3.6.12 Keuntungan dan Kekurangan

Hasil pelatihan agen menggunakan metode *Proximal Policy Optimization* (*PPO*) pada berbagai *environment* menunjukkan sejumlah keuntungan signifikan. Pertama, pada *environment* seperti *Pong* dan *Flappy Bird*, agen mampu menunjukkan pola pembelajaran yang stabil, ditandai dengan peningkatan *reward* dan *episode* length yang konsisten. Ini mengindikasikan bahwa algoritma PPO efektif dalam lingkungan yang memiliki struktur *reward* yang jelas. Selain itu, penerapan *vectorized environment* memungkinkan proses eksplorasi dan pelatihan berjalan lebih efisien dengan waktu yang relatif lebih singkat. Keunggulan lain terletak pada kemampuannya mengadaptasi berbagai jenis input visual dengan *preprocessing* yang sesuai seperti *grayscaling, resizing*, dan *frame stacking*, yang terbukti membantu dalam mengidentifikasi fitur penting dari *environment*.

Kekurangan dapat dilihat pada environment dengan kompleksitas tinggi dan dinamika *reward* yang tidak stabil, seperti *Breakout* dan *Super Mario Bros*. Penurunan performa agen secara drastis di kedua environment tersebut

menunjukkan bahwa PPO tidak selalu mampu menangani tantangan strategi berlapis, navigasi lingkungan kompleks dengan optimal. Kekurangan lain dapat di observasi pada kebutuhan sumber daya yang sangat tinggi yang berdampak pada limitasi menjalankan *vectorized environment* yang lebih tinggi.