CUSTOMIZING OPENAI GYM ENVIRONMENTS AND IMPLEMENTING REINFORCEMENT LEARNING AGENTS WITH STABLE BASELINES

WORK DEVELOPED BY
ALEXANDRE SOUSA
MARTA LONGO
SARA TÁBOAS

CONTENT

01 TAXI ENVIRONMENT: OVERVIEW

02 TAXI ENVIRONMENT

03 CHOSEN RL ALGORITHMS

04 CHANGES INTRODUCED

05 RESULT ANALYSIS

06 RESULT ANALYSIS

07 CONCLUSIONS

08 REFERENCES

TAXI ENVIRONMENT

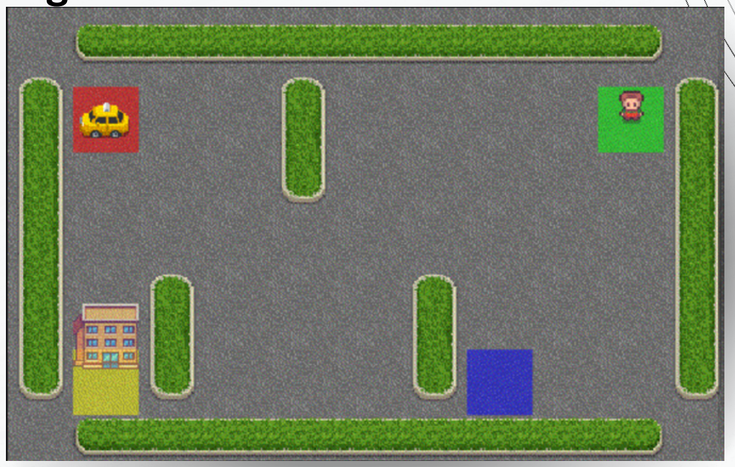
For this project we chose the Taxi-v3 environment. This environment is part of the *Toy Text* environments in *openAI Gym Documentation*.

OVERVIEW OF THE ENVIRONMENT

- **Grid World Layout:** 5x5 grid with four designated locations (**Red**, **Green**, **Yellow**, **Blue**)

- **Task:** When the episode starts, the taxi starts off at a random square and the passenger is at a random location. The taxi drives to the passenger's location, picks up the passenger, drives to the passenger's destination (another one of the four specified locations), and then drops off the passenger. Once the passenger is dropped off, the episode ends.

Fig 1. Taxi-v3-Env



TAXI ENVIRONMENT

States

There are 500 discrete states:

- 25 taxi positions
- 5 possible locations of the passenger (including the case when the passenger is in the taxi)
- 4 destination locations.
- Each state space is represented by the tuple: (taxi_row, taxi_col, passenger_location, destination)

- **Passenger locations:**
  - 0: R(ed)
  - 1: G(reen)
  - 2: Y(ellow)
  - 3: B(lue)
  - 4: in taxi

- **Destinations:**
  - 0: R(ed)
  - 1: G(reen)
  - 2: Y(ellow)
  - 3: B(lue)

Actions

There are 6 discrete deterministic actions:

- 0: move south
- 1: move north
- 2: move east
- 3: move west
- 4: pickup passenger
- 5: drop off passenger

Rewards

- -1 per step unless other reward is triggered.
- +20 delivering passenger.
- -10 executing “pickup” and “drop-off” actions illegally

CHOSEN RL ALGORITHMS

PPO (PROXIMAL POLICY OPTIMIZATION)

A2C (ADVANTAGE ACTOR-CRITIC)

DQN (DEEP Q-NETWORK)

We did hyperparameter tunning for each model.

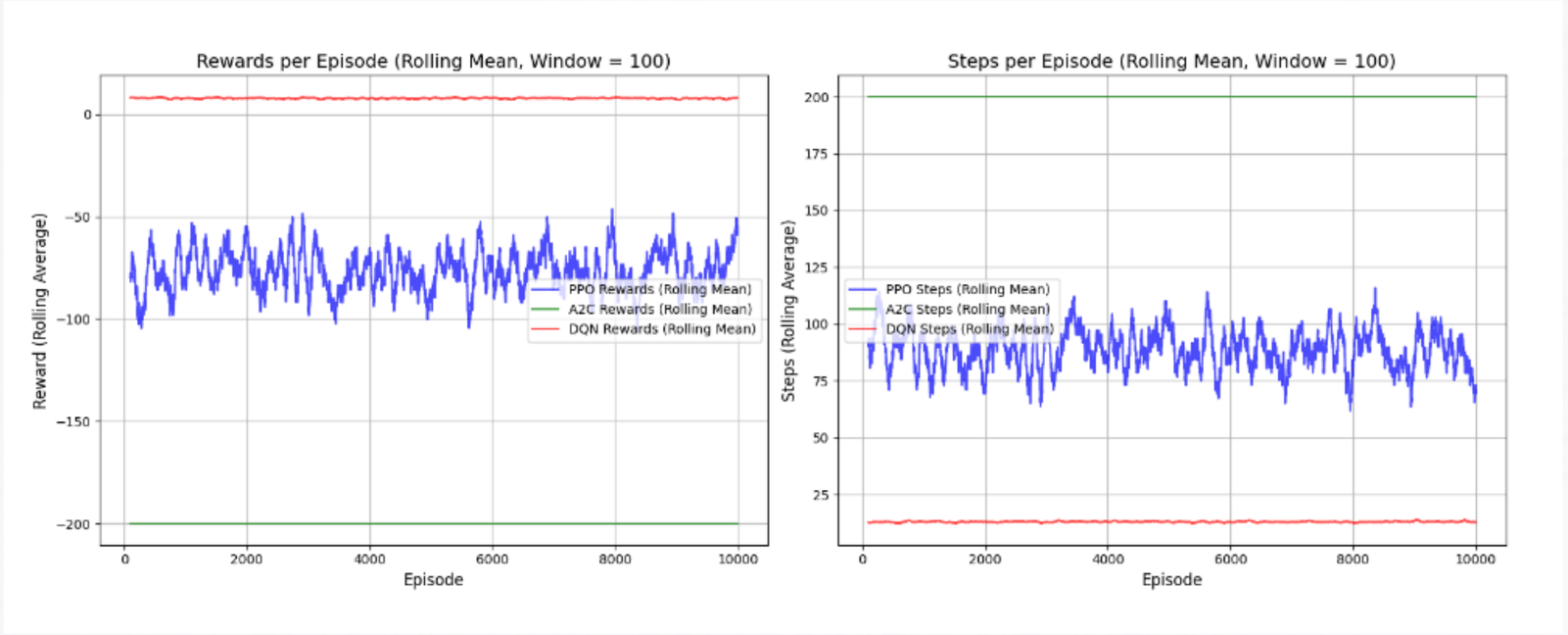


Fig 2. RL Models tested on baseline-model

CHANGES INTRODUCED IN THE AGENTS ACTIONS

ACTIONS

Using the **CustomActionWrapper** that inherites the **ActionWrapper** from the *Gym Documentation* to alter the libraries deterministic actions. The alterations made to the deterministic actions in the Gym library expand the action space by introducing diagonal movements. Specifically, four new actions (6 through 9) are defined to allow diagonal navigation on a 2D grid.

REWARDS

Using the **CustomRewardWrapper** that inherites the **RewardWrapper** from the *Gym Documentation* to alter the libraries rewards. If the taxi revisits a recently visited place the penalty is bigger.

COMBINED

Here we combine the two changes.

ANALYSIS OF RESULTS - PPO

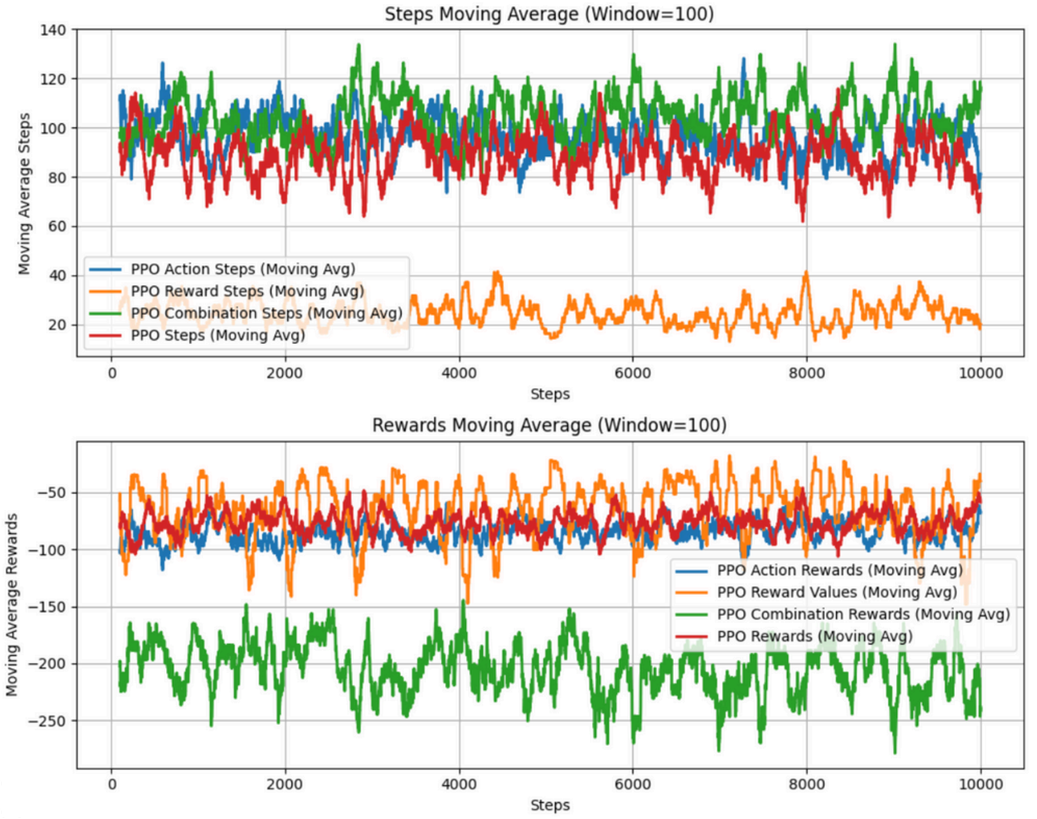
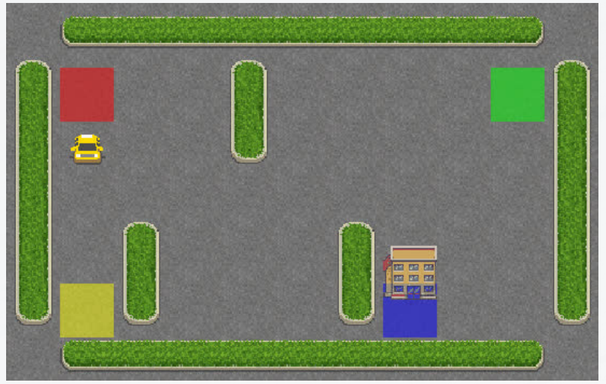
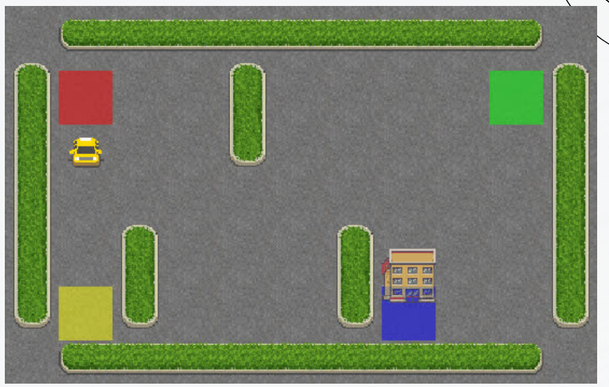


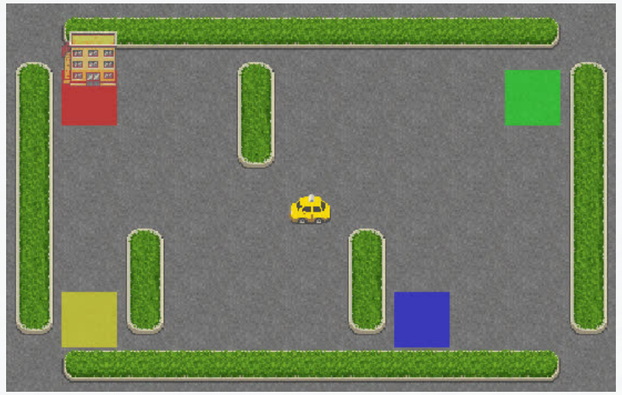
Fig 3. Comparison of all the models and PPO



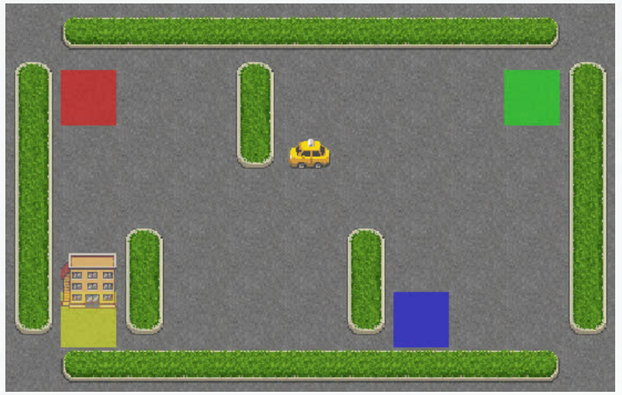
Original environment



Action Custom

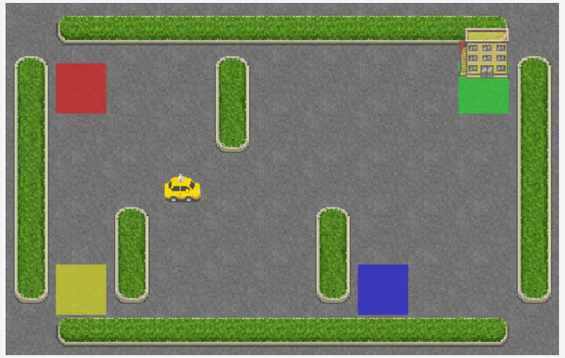


Reward Custom

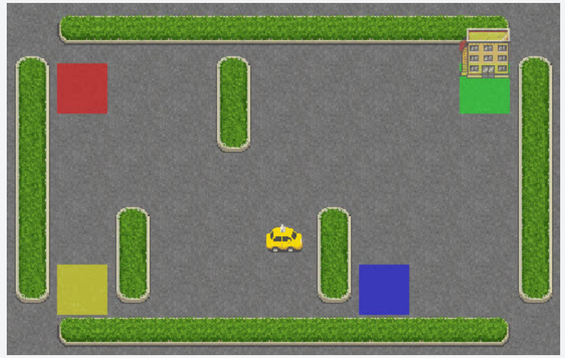


Combination Custom

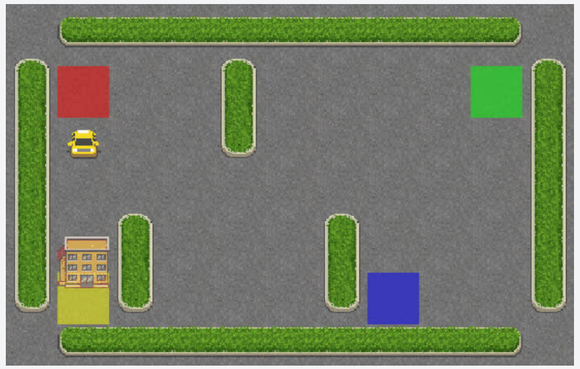
ANALYSIS OF RESULTS - DQN



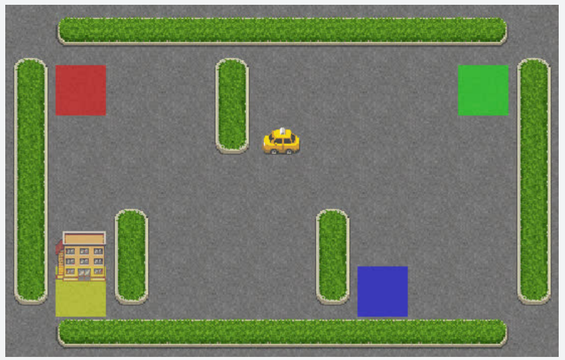
Original Environment



Action Custom



Reward Custom



Combination Custom

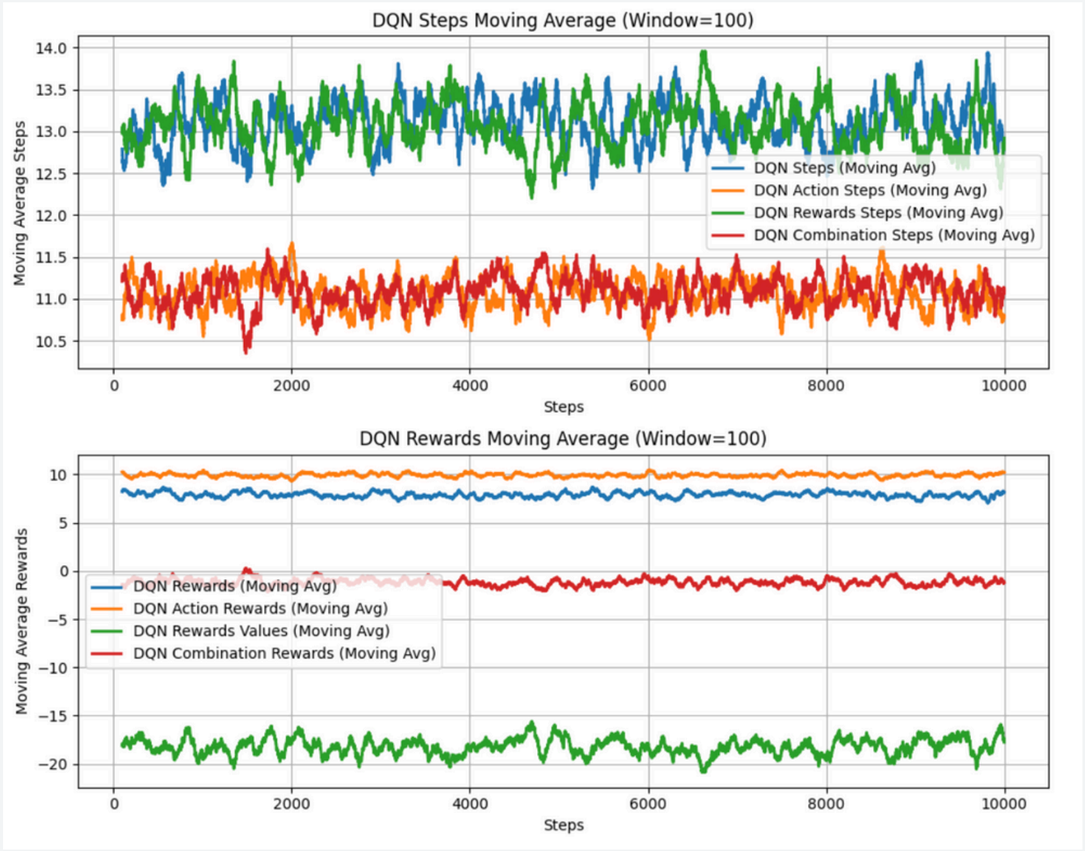


Fig 4. Comparison of all the models and PPO

REFERENCES

- **OpenAI Gym Documentation:** https://www.gymlibrary.dev
- **Stable-Baselines3 Documentation:** https://stable-baselines3.readthedocs.io/en/master/
- https://github.com/openai/gym/blob/master/gym/envs/toy_text/taxi.py

ALGORITHMS - EXECUTION DETAILS

```python
def param_tuning_multithread(model_class, params, total_timesteps, n_eval_episodes, verbose, max_threads):

    param_grid = list(ParameterGrid(params))
    best_reward = -float('inf')
    best_params = None


    with ThreadPoolExecutor(max_threads) as executor:
        futures = [
            executor.submit(evaluate_params, model_class, param_comb, total_timesteps, n_eval_episodes, verbose)
            for param_comb in param_grid
        ]

        for future in as_completed(futures):
            try:
                mean_reward, param_comb = future.result()
                if mean_reward > best_reward:
                    best_reward = mean_reward
                    best_params = param_comb
            except Exception as e:
                print(f"Erro ao processar parâmetros: {e}")

    print(f"Best parameters: {best_params}")
    print (f"Best reward: {best_reward}")
    return best_params, best_reward
```

Hyperparameter Tunning with MultiProcessing

```python
best_parameters_ppo =  {
    'batch_size': 32,
    'ent_coef': 0.1, 'gamma': 0.95,
    'learning_rate': 0.001,
    'n_steps': 512
}
```

PPO best parameters

```python
best_parameters_a2c = {
    'ent_coef': 0.0,
    'gamma': 0.99,
    'learning_rate': 0.0001,
    'max_grad_norm': 1.0,
    'n_steps': 50,
    'vf_coef': 0.75
}
```

A2C best parameters

```python
best_parameters_dqn = {
    'batch_size': 32,
    'buffer_size': 100000,
    'exploration_final_eps': 0.1,
    'gamma': 0.95,
    'learning_rate': 0.001
}
```

DQN best parameters

ENVIRONMENT CUSTOMIZATION - EXECUTION DETAILS

Customized Action Wrapper

```python
class CustomActionWrapper(ActionWrapper):
    def __init__(self, env):
        super().__init__(env)
        self.env = env
        self.action_space = gym.spaces.Discrete(env.action_space.n + 4) # Add 4 diagonal actions

    def action(self, action):
        """Transform the custom action into a base action if necessary."""
        if action in [6, 7, 8, 9]:
            # Diagonal actions are handled in the `step` method.
            return None  # Indicate custom handling
        return action  # Pass through for base actions


    def step(self, action):
        grid_size = 5

        # Get the taxi position
        encoded_state = self.env.unwrapped.s

        taxi_row, taxi_col, pass_idx, dest_idx = self.env.unwrapped.decode(encoded_state)

        # Handle diagonal actions
        if action == 6:  # Move South-East
            new_row = min(taxi_row + 1, grid_size - 1)
            new_col = min(taxi_col + 1, grid_size - 1)
        elif action == 7:  # Move South-West
            new_row = min(taxi_row + 1, grid_size - 1)
            new_col = max(taxi_col - 1, 0)
        elif action == 8:  # Move North-East
            new_row = max(taxi_row - 1, 0)
            new_col = min(taxi_col + 1, grid_size - 1)
        elif action == 9:  # Move North-West
            new_row = max(taxi_row - 1, 0)
            new_col = max(taxi_col - 1, 0)
        else:
            # Pass through base actions to the original step method
            return super().step(action)

        # Validate the move (check for walls or invalid spaces)
        if self.env.unwrapped.desc[new_row, new_col] != b' ':  # Assume `b' '` indicates valid space
            # Invalid move, no state change
            reward = -1  # Same as base environment for invalid moves
            done = False
            obs = self.env.unwrapped.s
        else:
            # Update the state manually for diagonal actions
            self.env.unwrapped.s = self.env.unwrapped.encode(new_row, new_col, pass_idx, dest_idx)

            # Compute reward manually
            reward = -1  # Default reward for non-goal moves

            # Check if the new state is terminal
            done = self.env.unwrapped.s == self.env.unwrapped.encode(
                dest_idx // grid_size, dest_idx % grid_size, pass_idx, dest_idx
            )

            # Get updated observation
            obs = self.env.unwrapped.s

        truncated = False  # Taxi environment doesn't use truncation

        # Return updated information
        return obs, reward, done, truncated, {}
```

Customized Reward Wrapper

```python
class CustomRewardWrapper(RewardWrapper):
    def __init__(self, env, penalty=1.0):
        super().__init__(env)
        self.env = env
        self.visited_positions = set()  # Set to store visited positions
        self.penalty = penalty

    def step(self, action):
        obs, reward, done, _, info = self.env.step(action)


        # Get the taxi position
        encoded_state = self.env.unwrapped.s

        taxi_row, taxi_col, pass_idx, dest_idx = self.env.unwrapped.decode(encoded_state)
        # Get the taxi's position (assuming obs contains the position as (x, y) coordinates)
        taxi_pos = (taxi_row, taxi_col)  # The position is stored in the first two elements of the observation

        # Check if the agent revisits a position
        if taxi_pos in self.visited_positions:
            # Apply a penalty if the taxi revisits a position
            reward -= self.penalty
        else:

            # Mark the current position as visited
            self.visited_positions.add(taxi_pos)

        # If the agent successfully picks up or drops off a passenger, we don't penalize
        if done and 'pickup' in info and info['pickup']:
            self.visited_positions.clear()  # Reset visited positions after task completion

        return obs, reward, done, _, info
```

PPO TRAINING PERFORMANCE - RESULTS

Fig 5. Baseline Model training results with PPO

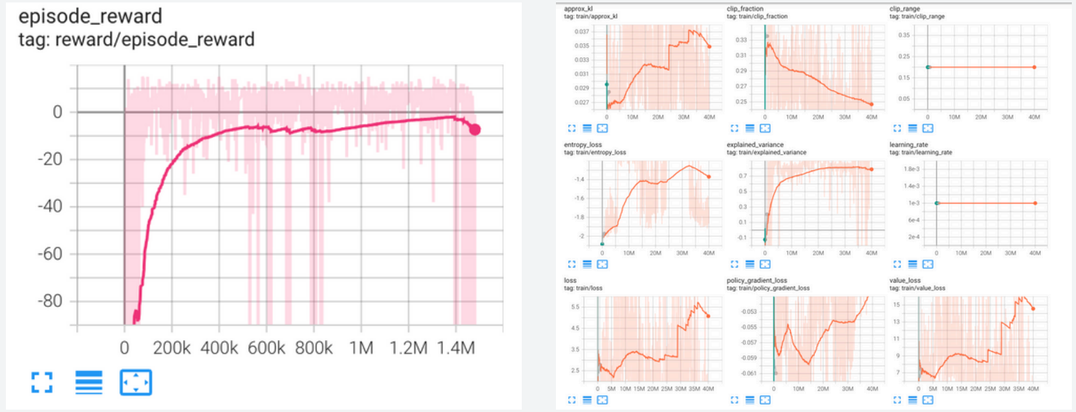


Fig 6. Action Custom training results with PPO

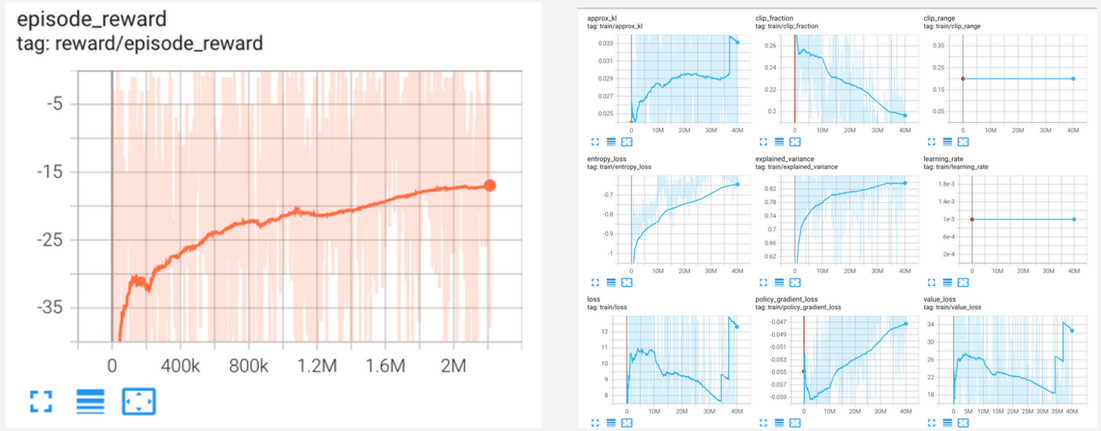


Fig 7. Reward Custom training results with PPO

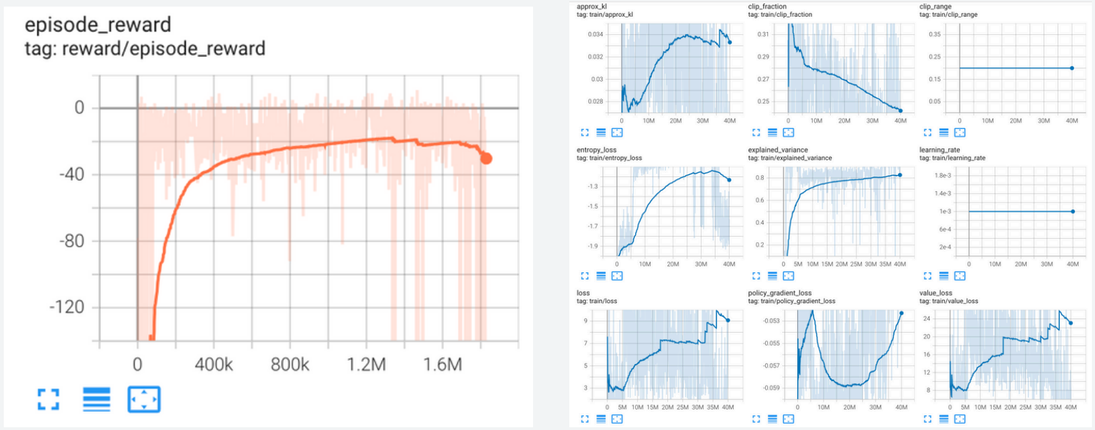


Fig 8. Action + Reward Custom training results with PPO

A2C TRAINING PERFORMANCE - RESULTS

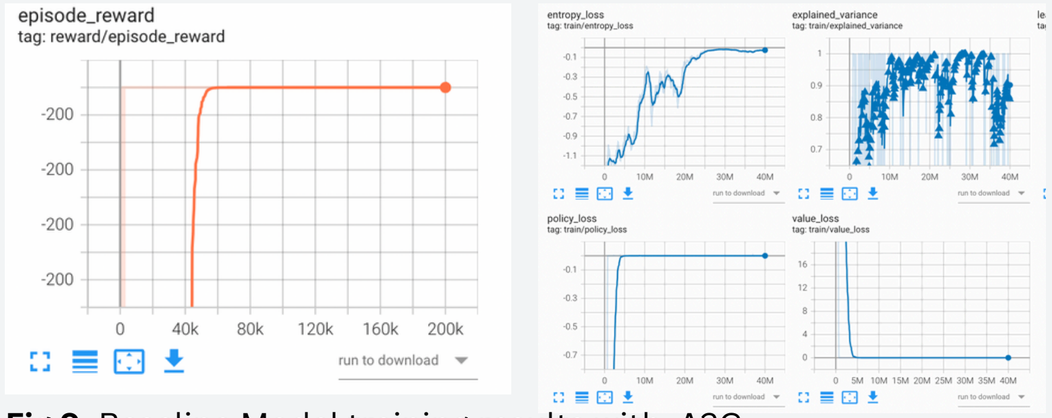


Fig 9. Baseline Model training results with A2C

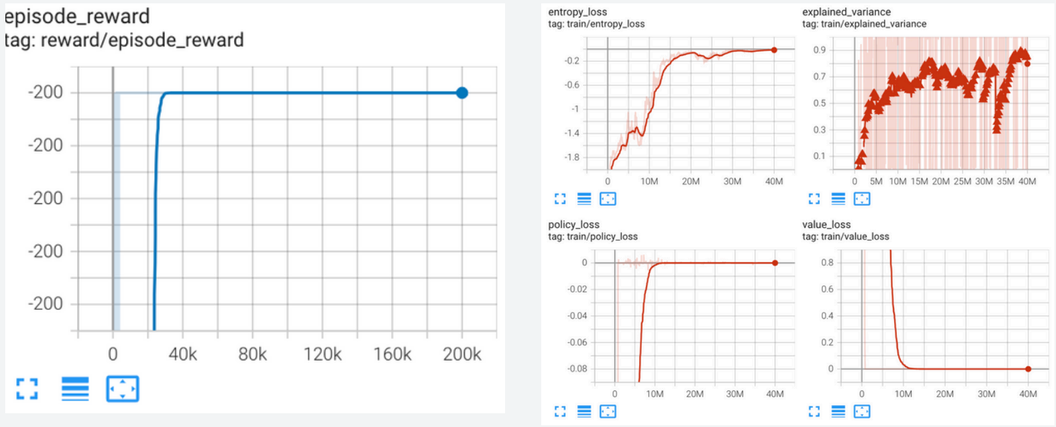


Fig 10.Action training results with A2C

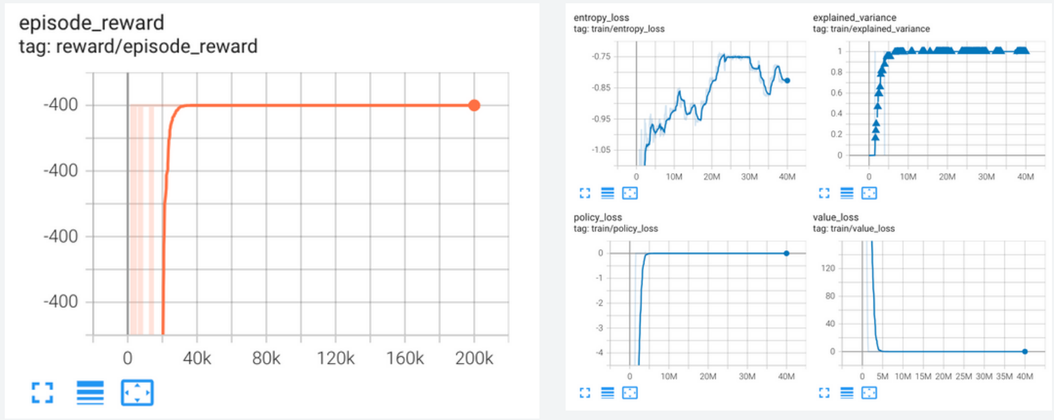


Fig 11. Reward Custom training results with A2C

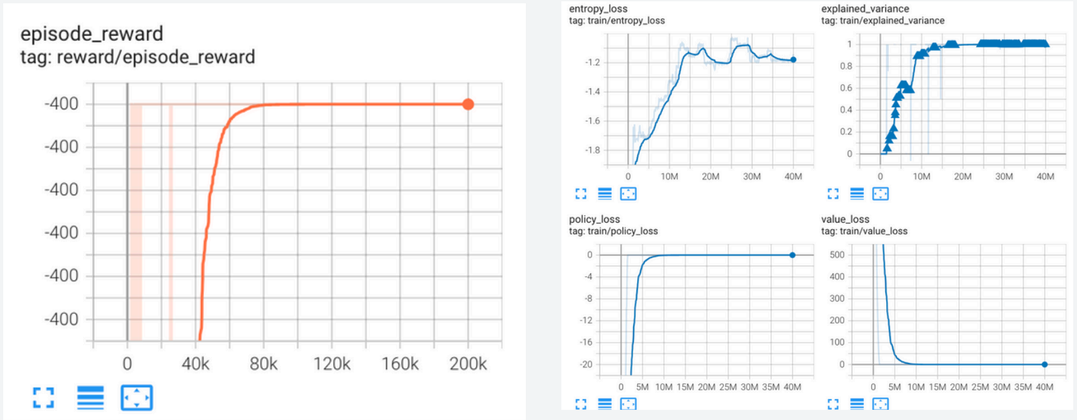


Fig 12. Action + Reward Custom training results with A2C

DQN TRAINING PERFORMANCE - RESULTS

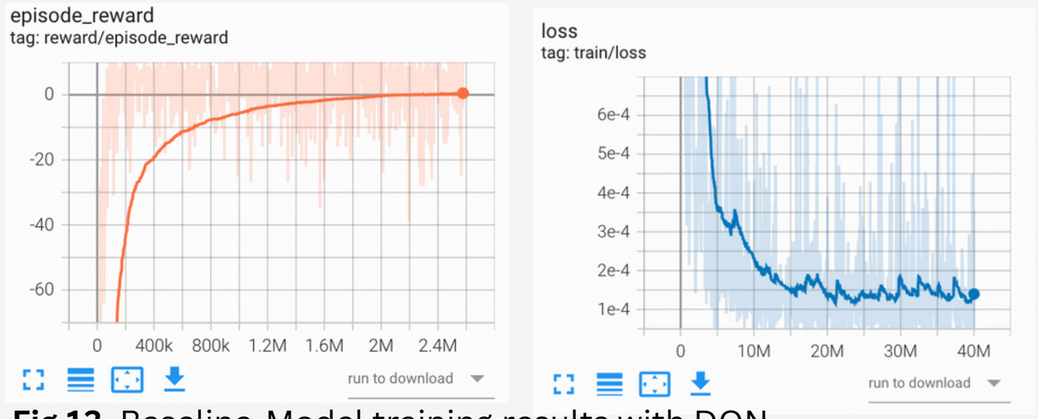


Fig 13. Baseline-Model training results with DQN

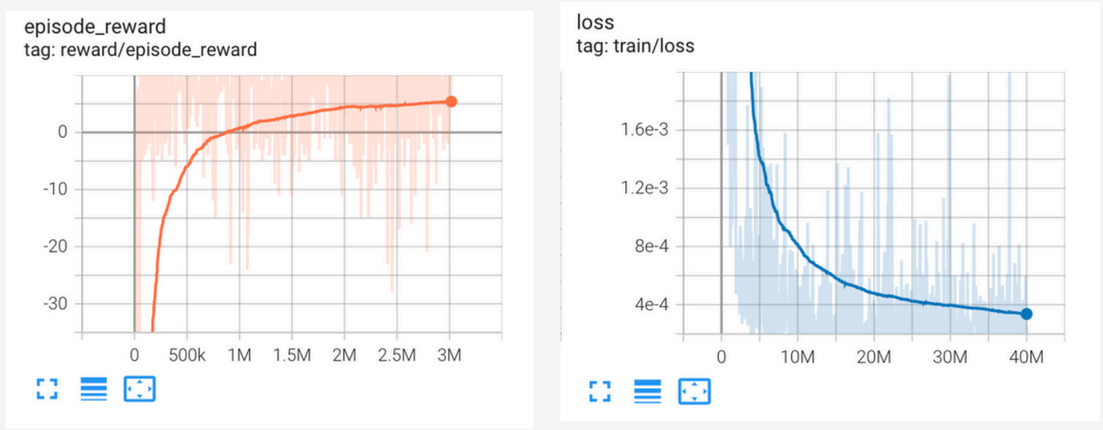


Fig 14. Action Custom training results with DQN

Fig 15. Reward Custom training results with DQN

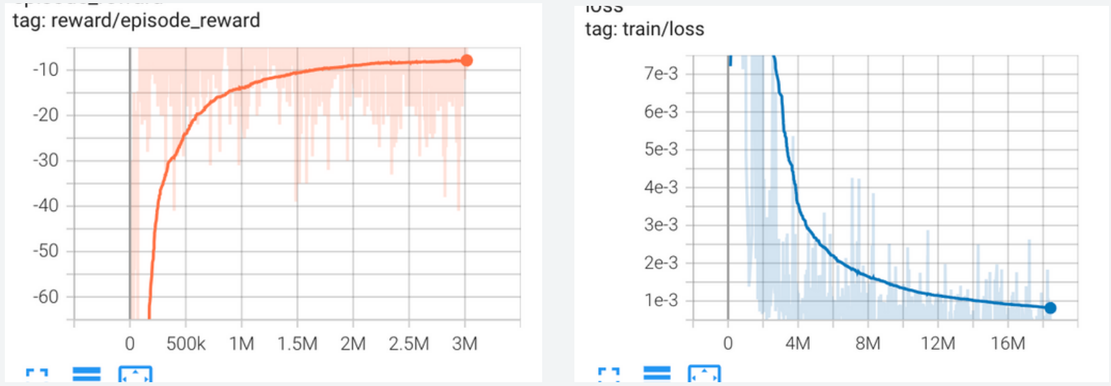


Fig 16. Reward + Action Custom training results with DQN