```
# Enhanced Curriculum Learning for Ant-v5 with Progressive Task Decomposition
# For Google Colab with visualization

# 1. Install dependencies
!apt-get install -y xvfb python-opengl ffmpeg > /dev/null 2>&1
!pip install gymnasium[mujoco] stable-baselines3 matplotlib pyvirtualdisplay > /dev/null 2>&1

# 2. Set up virtual display
from pyvirtualdisplay import Display
display = Display(visible=0, size=(1400, 900))
display.start()

# 3. Import required libraries
import os
import time
import numpy as np
import matplotlib.pyplot as plt
from IPython.display import display, clear_output, HTML
from base64 import b64encode
import gymnasium as gym
from stable_baselines3 import PPO
from stable_baselines3.common.callbacks import BaseCallback
from stable_baselines3.common.vec_env import DummyVecEnv, VecNormalize
from stable_baselines3.common.evaluation import evaluate_policy
from typing import List, Tuple, Optional

# 4. Create directories for saving
os.makedirs("./models", exist_ok=True)
os.makedirs("./plots", exist_ok=True)
os.makedirs("./vec_normalize", exist_ok=True)

# 5. Enhanced Ant Curriculum Wrapper
class ProgressiveAntCurriculum(gym.Wrapper):
    def __init__(self,
                 env: gym.Env,
                 initial_difficulty: float = 0.0,
                 max_difficulty: float = 1.0,
                 adaptation_speed: float = 0.01,
                 success_threshold: float = 0.8,
                 window_size: int = 20,
                 curriculum_stages: List[Tuple[str, float]] = None):
        """
        Enhanced curriculum wrapper with progressive task decomposition.

        Args:
            env: Gym environment to wrap
            initial_difficulty: Starting difficulty (0-1)
            max_difficulty: Maximum difficulty level
            adaptation_speed: How quickly to adjust difficulty
            success_threshold: Performance threshold for increasing difficulty
            window_size: Number of episodes to consider for performance evaluation
            curriculum_stages: List of (stage_name, target_difficulty) pairs
        """
        super().__init__(env)
```

```
        # Curriculum parameters
        self.difficulty = initial_difficulty
        self.max_difficulty = max_difficulty
        self.adaptation_speed = adaptation_speed
        self.success_threshold = success_threshold
        self.window_size = window_size

        # Curriculum stages for progressive learning
        self.curriculum_stages = curriculum_stages or [
            ("limb_coordination", 0.2),
            ("basic_movement", 0.4),
            ("obstacle_navigation", 0.6),
            ("perturbed_movement", 0.8),
            ("full_task", 1.0)
        ]
        self.current_stage = 0

        # Tracking
        self.recent_rewards = []
        self.episode_count = 0
        self.difficulty_history = []
        self.stage_history = []

        # Environment modifications
        self.original_init_params = self._get_env_params()
        self._apply_curriculum_modifications()

        print(f"Initialized ProgressiveAntCurriculum with stage: {self.curriculum_stages[self.current_stage][0]}")

    def _get_env_params(self) -> dict:
        """Get original environment parameters for reference"""
        return {
            'gravity': self.env.unwrapped.model.opt.gravity.copy(),
            'torso_mass': self.env.unwrapped.model.body_mass[1].copy(),
            'joint_damping': self.env.unwrapped.model.dof_damping[:8].copy(),
            'ctrl_range': self.env.unwrapped.model.actuator_ctrlrange[:8].copy()
        }

    def _apply_curriculum_modifications(self):
        """Apply modifications based on current curriculum stage"""
        stage_name, target_diff = self.curriculum_stages[self.current_stage]

        # Reset to original parameters first
        self._reset_to_original()

        # Apply stage-specific modifications
        if stage_name == "limb_coordination":
            # Easier limb control
            self.env.unwrapped.model.dof_damping[:8] *= 0.5  # Reduce joint damping
            self.env.unwrapped.model.actuator_ctrlrange[:8, :] *= 0.7  # Reduce action range

        elif stage_name == "basic_movement":
            # Normal physics but easier control
            self.env.unwrapped.model.dof_damping[:8] *= 0.7
            self.env.unwrapped.model.actuator_ctrlrange[:8, :] *= 0.85

        elif stage_name == "obstacle_navigation":
```

```
            # Add small random obstacles (simulated by perturbations)
            pass

        elif stage_name == "perturbed_movement":
            # Increased gravity and mass for robustness
            self.env.unwrapped.model.opt.gravity[2] *= 1.2  # Increase gravity
            self.env.unwrapped.model.body_mass[1] *= 1.2  # Increase torso mass

        elif stage_name == "full_task":
            # Original task parameters
            pass

        # Apply difficulty scaling within stage
        self._scale_difficulty(target_diff)

    def _reset_to_original(self):
        """Reset environment parameters to original values"""
        params = self.original_init_params
        self.env.unwrapped.model.opt.gravity[:] = params['gravity']
        self.env.unwrapped.model.body_mass[1] = params['torso_mass']
        self.env.unwrapped.model.dof_damping[:8] = params['joint_damping']
        self.env.unwrapped.model.actuator_ctrlrange[:8] = params['ctrl_range']

    def _scale_difficulty(self, target_diff: float):
        """Scale parameters based on difficulty within stage"""
        # Linearly interpolate between stage parameters
        interp = min(self.difficulty / target_diff, 1.0)

        # Example: gradually increase gravity
        self.env.unwrapped.model.opt.gravity[2] = (
            self.original_init_params['gravity'][2] * (1 + 0.5 * interp))

        # Gradually increase torso mass
        self.env.unwrapped.model.body_mass[1] = (
            self.original_init_params['torso_mass'] * (1 + 0.3 * interp))

    def reset(self, **kwargs):
        self.episode_rewards = 0
        return self.env.reset(**kwargs)

    def step(self, action):
        # Apply action
        obs, reward, terminated, truncated, info = self.env.step(action)

        # Track rewards
        self.episode_rewards += reward

        # Apply perturbations if in later stages
        if self.curriculum_stages[self.current_stage][0] in ["perturbed_movement", "full_task"]:
            if np.random.random() < 0.1 * self.difficulty:  # 10% chance at max difficulty
                self._apply_perturbation()

        # Update curriculum if episode done
        if terminated or truncated:
            self._update_curriculum()

        return obs, reward, terminated, truncated, info
```

```
    def _apply_perturbation(self):
        """Apply controlled perturbation to torso"""
        try:
            # Random directional force scaled by difficulty
            force = np.random.uniform(-1, 1, 3) * 15 * self.difficulty
            self.env.unwrapped.data.xfrc_applied[1, :3] = force
        except Exception as e:
            print(f"Could not apply perturbation: {e}")

    def _update_curriculum(self):
        """Update curriculum stage and difficulty based on performance"""
        # Update tracking
        self.recent_rewards.append(self.episode_rewards)
        if len(self.recent_rewards) > self.window_size:
            self.recent_rewards.pop(0)

        self.episode_count += 1
        self.difficulty_history.append(self.difficulty)
        self.stage_history.append(self.current_stage)

        # Only update every window_size episodes
        if self.episode_count % self.window_size != 0:
            return

        if len(self.recent_rewards) < self.window_size:
            return

        # Calculate normalized performance (0-1)
        avg_reward = np.mean(self.recent_rewards)
        max_expected = 3000  # Approximate max reward for normalization
        normalized_perf = np.clip(avg_reward / max_expected, 0, 1)

        # Get current stage parameters
        stage_name, target_diff = self.curriculum_stages[self.current_stage]

        # Check if ready to advance to next stage
        if (self.current_stage < len(self.curriculum_stages) - 1 and
            self.difficulty >= target_diff * 0.9 and
            normalized_perf > self.success_threshold):

            self.current_stage += 1
            new_stage = self.curriculum_stages[self.current_stage][0]
            print(f"\nAdvancing to stage: {new_stage} (Performance: {avg_reward:.1f})")
            self._apply_curriculum_modifications()

        # Update difficulty within stage
        if normalized_perf > self.success_threshold:
            # Increase difficulty
            new_diff = min(self.difficulty + self.adaptation_speed, target_diff)
            if new_diff > self.difficulty:
                print(f"Increasing difficulty: {self.difficulty:.2f}→{new_diff:.2f}")
                self.difficulty = new_diff
        else:
            # Decrease difficulty
            new_diff = max(self.difficulty - self.adaptation_speed * 0.5, 0)
            if new_diff < self.difficulty:
```

```
                if new_diff < self.difficulty:
                    print(f"Decreasing difficulty: {self.difficulty:.2f}→{new_diff:.2f}")
                    self.difficulty = new_diff

            # Apply current difficulty
            self._scale_difficulty(target_diff)

    # 6. Training and Evaluation Functions
    def make_ant_env(env_id='Ant-v5', curriculum=False, seed=0):
        """Create environment factory"""
        def _init():
            env = gym.make(env_id)
            env.reset(seed=seed)
            if curriculum:
                env = ProgressiveAntCurriculum(env)
            return env
        return _init

    def train_ant_model(total_timesteps=200000, curriculum=True):
        """Train model with enhanced curriculum"""
        # Create environments
        env_fn = make_ant_env(curriculum=curriculum)
        env = DummyVecEnv([env_fn])
        env = VecNormalize(env, norm_obs=True, norm_reward=True)

        eval_env_fn = make_ant_env(curriculum=curriculum)
        eval_env = DummyVecEnv([eval_env_fn])

        # Create model
        model = PPO(
            "MlpPolicy",
            env,
            verbose=0,
            learning_rate=3e-4,
            n_steps=2048,
            batch_size=64,
            n_epochs=10,
            gamma=0.99,
            gae_lambda=0.95,
            clip_range=0.2,
            ent_coef=0.01,
            device='cpu'
        )

        # Callback for tracking
        callback = CurriculumTrackingCallback(
            eval_env,
            eval_freq=10000,
            n_eval_episodes=5
        )

        # Train
        print(f"Training {'with curriculum' if curriculum else 'standard'} for {total_timesteps} steps...")
        start_time = time.time()
        model.learn(total_timesteps=total_timesteps, callback=callback)
        print(f"Training completed in {(time.time()-start_time)/60:.1f} minutes")
```

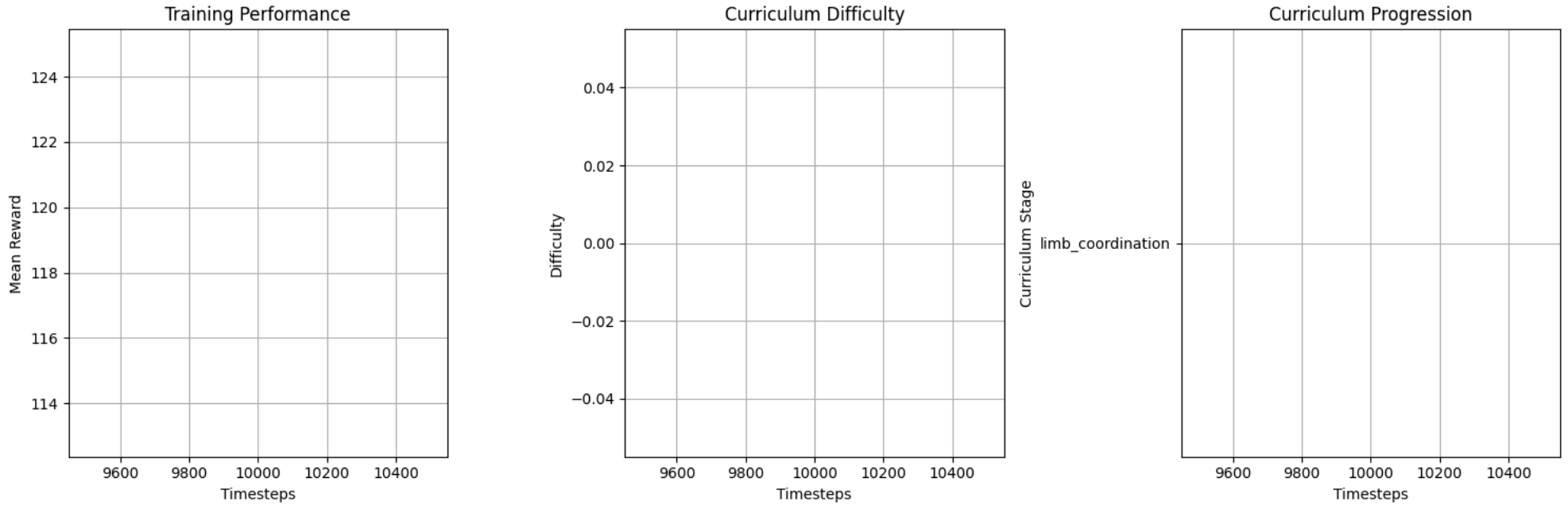
```
        # Save
        model.save(f"./models/ant_{'curriculum' if curriculum else 'standard'}")
        env.save(f"./vec_normalize/ant_{'curriculum' if curriculum else 'standard'}.pkl")

        return model, callback

    class CurriculumTrackingCallback(BaseCallback):
        """Enhanced tracking callback with curriculum visualization"""
        def __init__(self, eval_env, eval_freq=10000, n_eval_episodes=5):
            super().__init__()
            self.eval_env = eval_env
            self.eval_freq = eval_freq
            self.n_eval_episodes = n_eval_episodes
            self.rewards = []
            self.timesteps = []
            self.difficulties = []
            self.stages = []

        def _on_step(self):
            if self.n_calls % self.eval_freq == 0:
                # Evaluate
                mean_reward, _ = evaluate_policy(
                    self.model, self.eval_env, n_eval_episodes=self.n_eval_episodes)

                # Track metrics
                self.rewards.append(mean_reward)
                self.timesteps.append(self.n_calls)

                # Get curriculum info if available
                if hasattr(self.eval_env, 'envs'):
                    env = self.eval_env.envs[0]
                    if hasattr(env, 'difficulty_history') and env.difficulty_history:
                        self.difficulties.append(env.difficulty_history[-1])
                    if hasattr(env, 'current_stage') and hasattr(env, 'curriculum_stages'):
                        self.stages.append(env.curriculum_stages[env.current_stage][0])

                # Plot
                self._plot_progress()

            return True

        def _plot_progress(self):
            """Plot training progress with curriculum info"""
            plt.figure(figsize=(15, 5))

            # Reward plot
            plt.subplot(1, 3, 1)
            plt.plot(self.timesteps, self.rewards)
            plt.xlabel('Timesteps')
            plt.ylabel('Mean Reward')
            plt.title('Training Performance')
            plt.grid(True)

            # Difficulty plot
            if self.difficulties:
                plt.subplot(1, 3, 2)
                plt.plot(self.timesteps[:len(self.difficulties)], self.difficulties)
```

```
            plt.xlabel('Timesteps')
            plt.ylabel('Difficulty')
            plt.title('Curriculum Difficulty')
            plt.grid(True)

        # Stage plot
        if self.stages:
            plt.subplot(1, 3, 3)
            unique_stages = list(dict.fromkeys(self.stages))  # Preserve order
            stage_nums = [unique_stages.index(s) for s in self.stages]
            plt.plot(self.timesteps[:len(stage_nums)], stage_nums)
            plt.yticks(range(len(unique_stages)), unique_stages)
            plt.xlabel('Timesteps')
            plt.ylabel('Curriculum Stage')
            plt.title('Curriculum Progression')
            plt.grid(True)

        plt.tight_layout()
        display(plt.gcf())
        plt.close()

# 7. Visualization and Comparison
def visualize_ant(model, difficulty=0.5, steps=300):
    """Render ant behavior as video"""
    env = gym.make('Ant-v5', render_mode='rgb_array')
    env = ProgressiveAntCurriculum(env)
    env.difficulty = difficulty

    frames = []
    obs, _ = env.reset()
    for _ in range(steps):
        action, _ = model.predict(obs, deterministic=True)
        obs, _, terminated, truncated, _ = env.step(action)
        frames.append(env.render())
        if terminated or truncated:
            obs, _ = env.reset()

    env.close()

    # Save as video
    height, width, _ = frames[0].shape
    video_path = '/tmp/ant_curriculum.mp4'

    import subprocess
    cmd = [
        'ffmpeg', '-y', '-f', 'rawvideo',
        '-vcodec', 'rawvideo', '-s', f'{width}x{height}',
        '-pix_fmt', 'rgb24', '-r', '25', '-i', '-',
        '-c:v', 'libx264', '-pix_fmt', 'yuv420p',
        video_path
    ]
    process = subprocess.Popen(cmd, stdin=subprocess.PIPE)
    for frame in frames:
        process.stdin.write(frame.tobytes())
    process.stdin.close()
    process.wait()
```

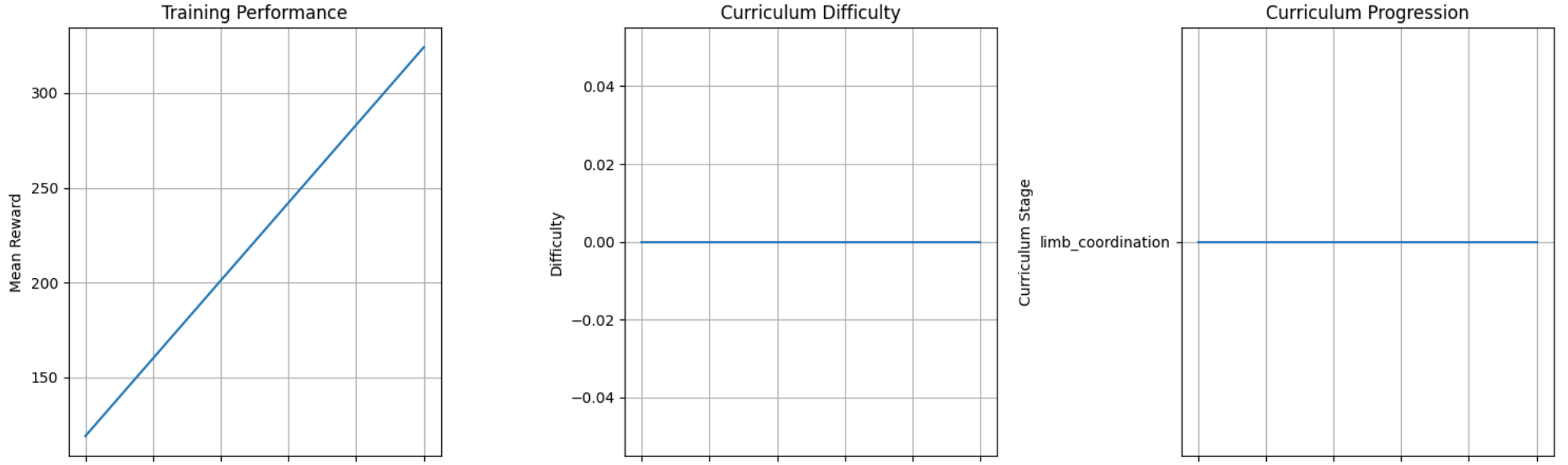
```
        # Display
        mp4 = open(video_path,'rb').read()
        data_url = "data:video/mp4;base64," + b64encode(mp4).decode()
        return HTML(f"""
        <video width=600 controls>
            <source src="{data_url}" type="video/mp4">
        </video>
        """)

    def compare_models(curriculum_model, standard_model):
        """Compare curriculum vs standard models"""
        difficulties = [0.0, 0.25, 0.5, 0.75, 1.0]
        curr_rewards = []
        std_rewards = []

        print("Evaluating models at different difficulties...")

        for diff in difficulties:
            # Curriculum model
            env = gym.make('Ant-v5')
            env = ProgressiveAntCurriculum(env)
            env.difficulty = diff
            mean_reward, _ = evaluate_policy(curriculum_model, env, n_eval_episodes=5)
            curr_rewards.append(mean_reward)
            env.close()

            # Standard model
            env = gym.make('Ant-v5')
            env = ProgressiveAntCurriculum(env)
            env.difficulty = diff
            mean_reward, _ = evaluate_policy(standard_model, env, n_eval_episodes=5)
            std_rewards.append(mean_reward)
            env.close()

            print(f"Difficulty {diff:.2f}: Curriculum={curr_rewards[-1]:.1f}, Standard={std_rewards[-1]:.1f}")

        # Plot comparison
        plt.figure(figsize=(10, 6))
        plt.plot(difficulties, curr_rewards, 'o-', label='Curriculum')
        plt.plot(difficulties, std_rewards, 's-', label='Standard')
        plt.xlabel('Difficulty Level')
        plt.ylabel('Mean Reward')
        plt.title('Performance Across Difficulties')
        plt.legend()
        plt.grid(True)
        plt.show()

        # Calculate metrics
        curr_avg = np.mean(curr_rewards)
        std_avg = np.mean(std_rewards)
        improvement = (curr_avg - std_avg) / abs(std_avg) * 100

        print(f"\nResults:")
        print(f"- Curriculum average reward: {curr_avg:.1f}")
        print(f"- Standard average reward: {std_avg:.1f}")
        print(f"- Improvement: {improvement:.1f}%")
```

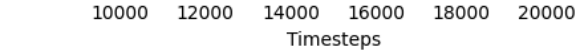
```
        # Robustness (performance drop from easy to hard)
        curr_drop = curr_rewards[0] - curr_rewards[-1]
        std_drop = std_rewards[0] - std_rewards[-1]
        robustness_improvement = (std_drop - curr_drop) / std_drop * 100

        print(f"\nRobustness:")
        print(f"- Curriculum performance drop: {curr_drop:.1f}")
        print(f"- Standard performance drop: {std_drop:.1f}")
        print(f"- Robustness improvement: {robustness_improvement:.1f}%")

    # 8. Main Experiment
    def run_experiment():
        """Run full curriculum learning experiment"""
        # Train curriculum model
        print("=== TRAINING CURRICULUM MODEL ===")
        curriculum_model, _ = train_ant_model(total_timesteps=200000, curriculum=True)

        # Train standard model
        print("\n=== TRAINING STANDARD MODEL ===")
        standard_model, _ = train_ant_model(total_timesteps=200000, curriculum=False)

        # Compare models
        print("\n=== COMPARING MODELS ===")
        compare_models(curriculum_model, standard_model)

        # Visualize
        print("\n=== VISUALIZING CURRICULUM MODEL ===")
        display(visualize_ant(curriculum_model, difficulty=0.5))

        print("\n=== VISUALIZING STANDARD MODEL ===")
        display(visualize_ant(standard_model, difficulty=0.5))

    # Run the experiment
    if __name__ == "__main__":
        run_experiment()
```

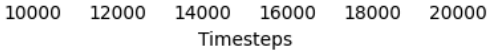
```
=== TRAINING CURRICULUM MODEL ===
Initialized ProgressiveAntCurriculum with stage: limb_coordination
Initialized ProgressiveAntCurriculum with stage: limb_coordination
Training with curriculum for 200000 steps...
/usr/local/lib/python3.11/dist-packages/stable_baselines3/common/evaluation.py:67: UserWarning: Evaluation environment is not wrapped with a ``Monitor`` wrapper. This may result in reporting mo
  warnings.warn(
```

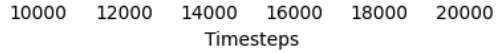



```
/usr/local/lib/python3.11/dist-packages/stable_baselines3/common/evaluation.py:67: UserWarning: Evaluation environment is not wrapped with a ``Monitor`` wrapper. This may result in reporting mo
  warnings.warn(
```

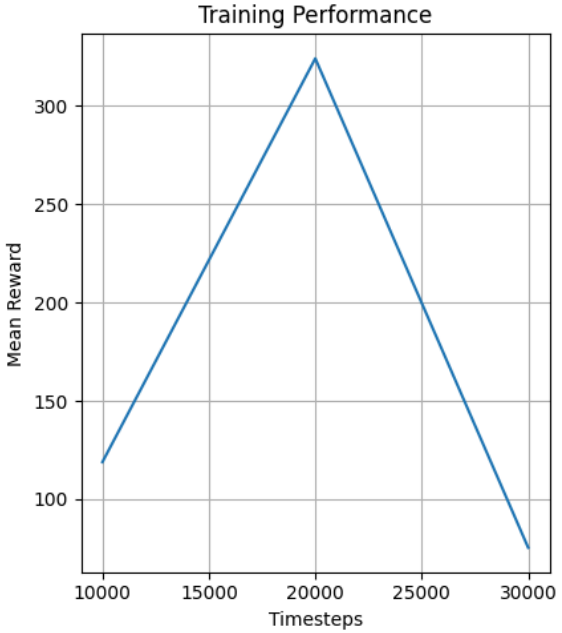


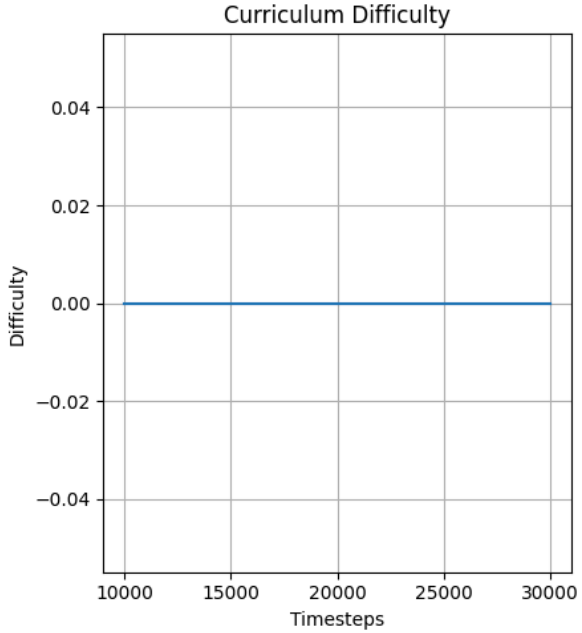


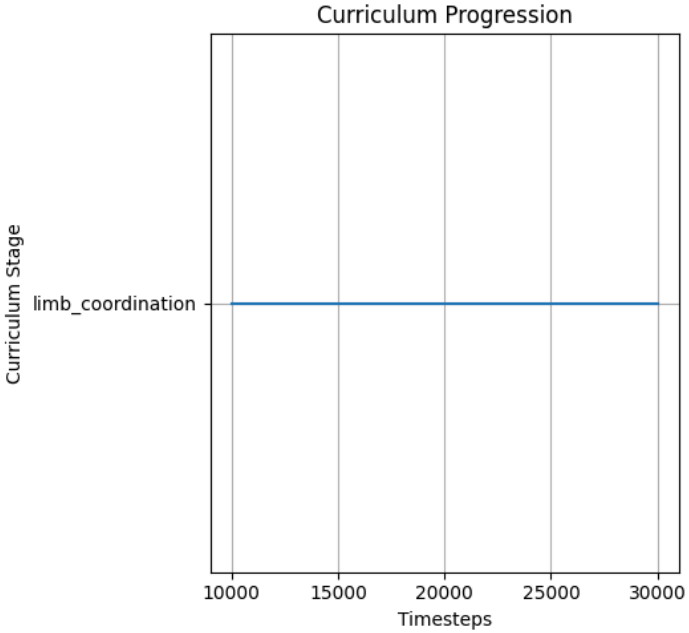




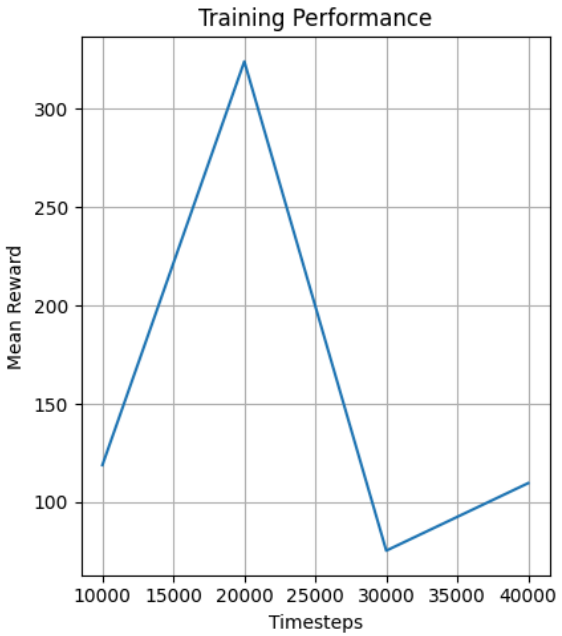

```
/usr/local/lib/python3.11/dist-packages/stable_baselines3/common/evaluation.py:67: UserWarning: Evaluation environment is not wrapped with a ``Monitor`` wrapper. This may result in reporting mo
  warnings.warn(
```

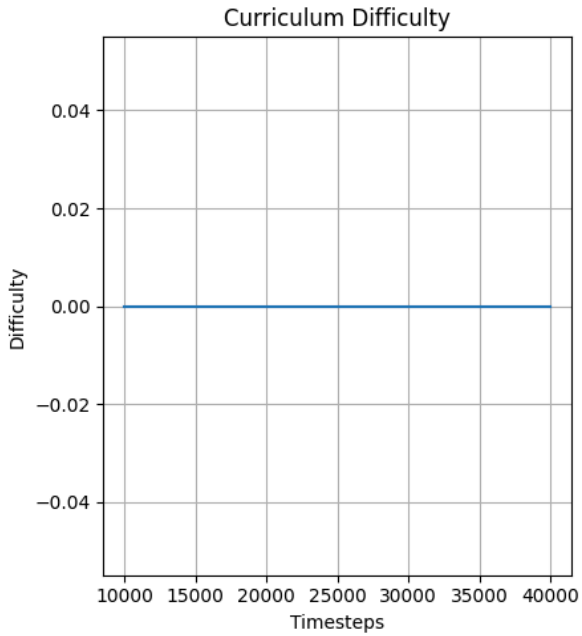


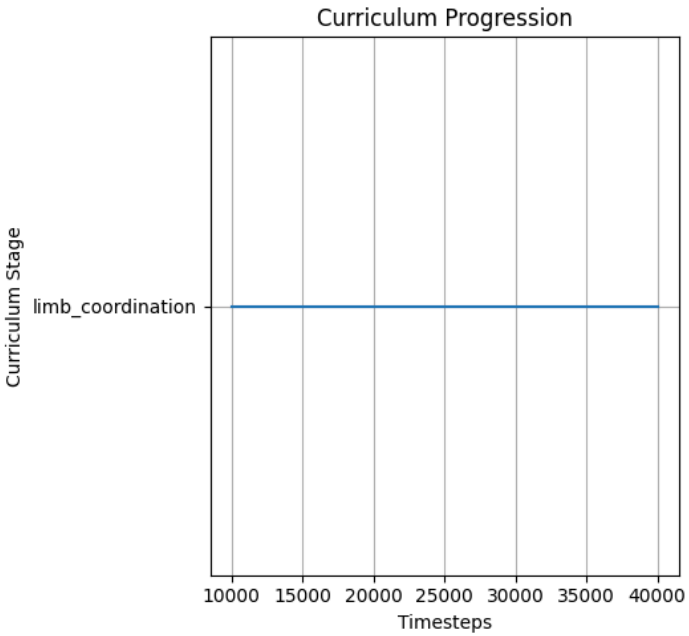




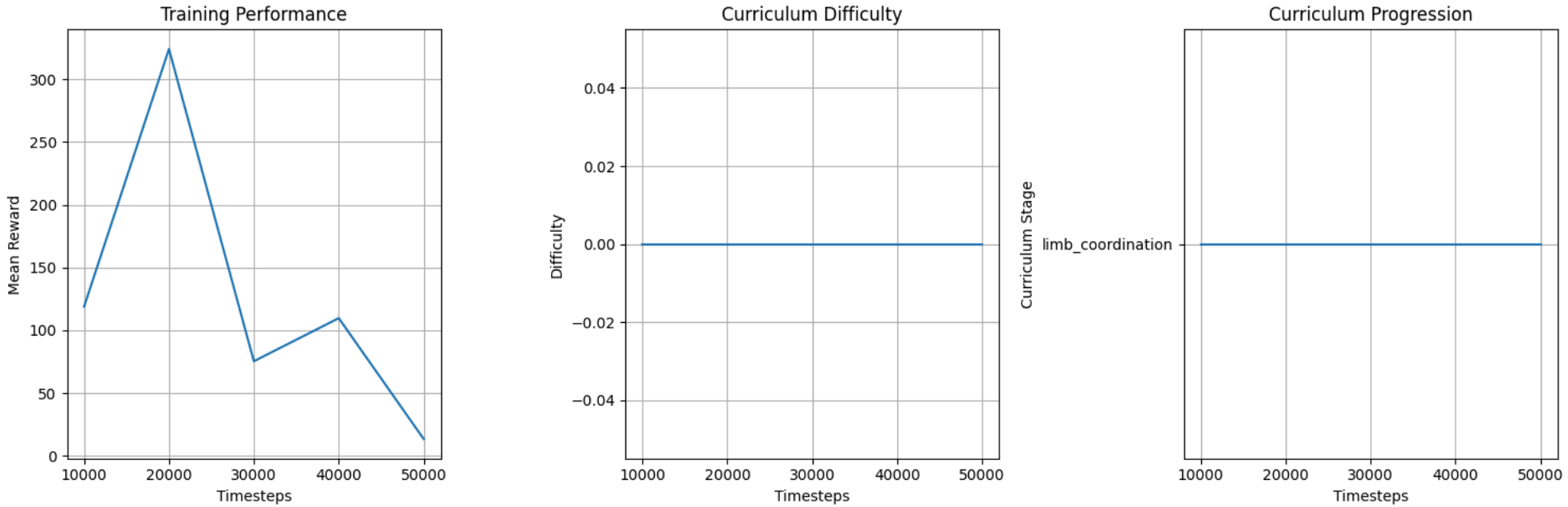

```
/usr/local/lib/python3.11/dist-packages/stable_baselines3/common/evaluation.py:67: UserWarning: Evaluation environment is not wrapped with a ``Monitor`` wrapper. This may result in reporting mo
  warnings.warn(
```



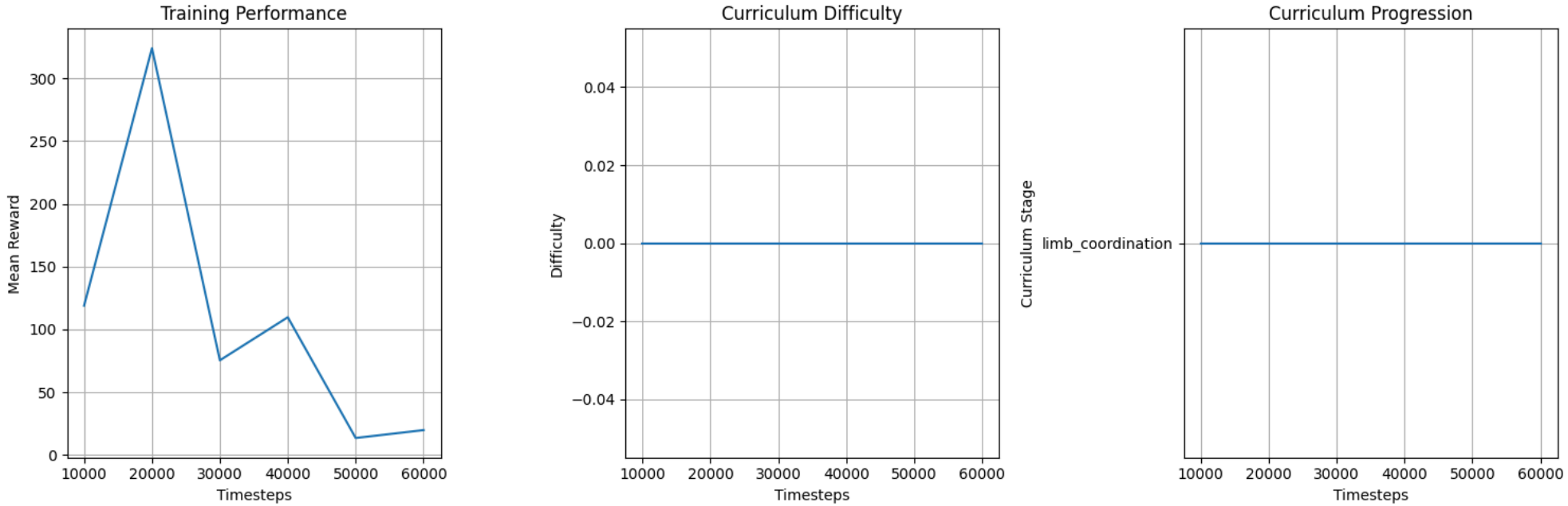




```
/usr/local/lib/python3.11/dist-packages/stable_baselines3/common/evaluation.py:67: UserWarning: Evaluation environment is not wrapped with a ``Monitor`` wrapper. This may result in reporting mo
  warnings.warn(
```

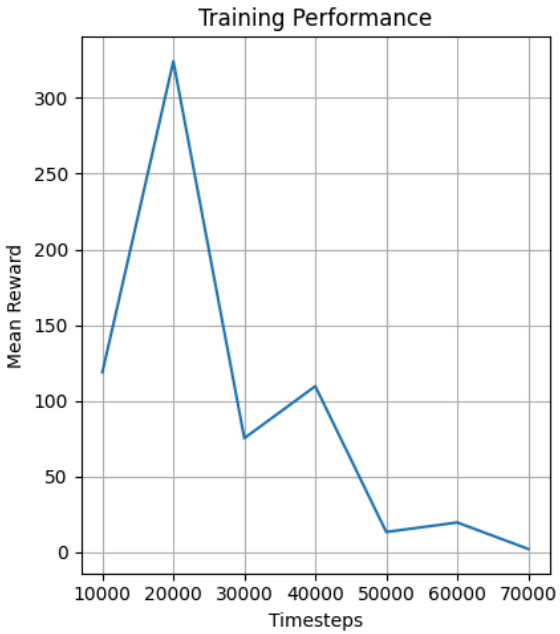


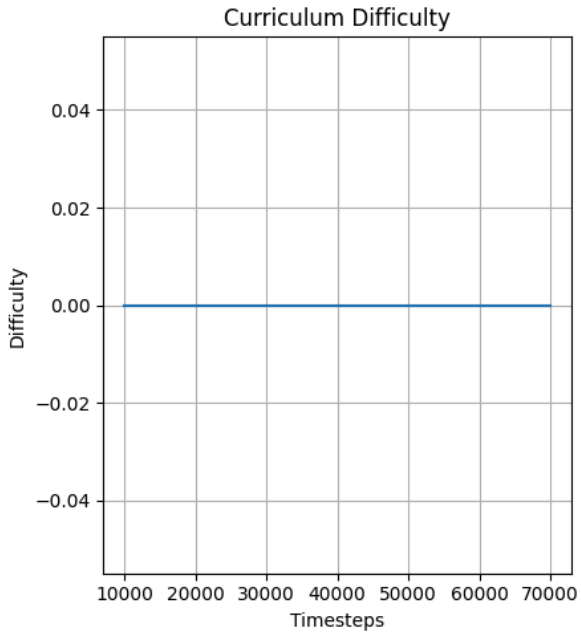

```
/usr/local/lib/python3.11/dist-packages/stable_baselines3/common/evaluation.py:67: UserWarning: Evaluation environment is not wrapped with a ``Monitor`` wrapper. This may result in reporting mo
  warnings.warn(
```

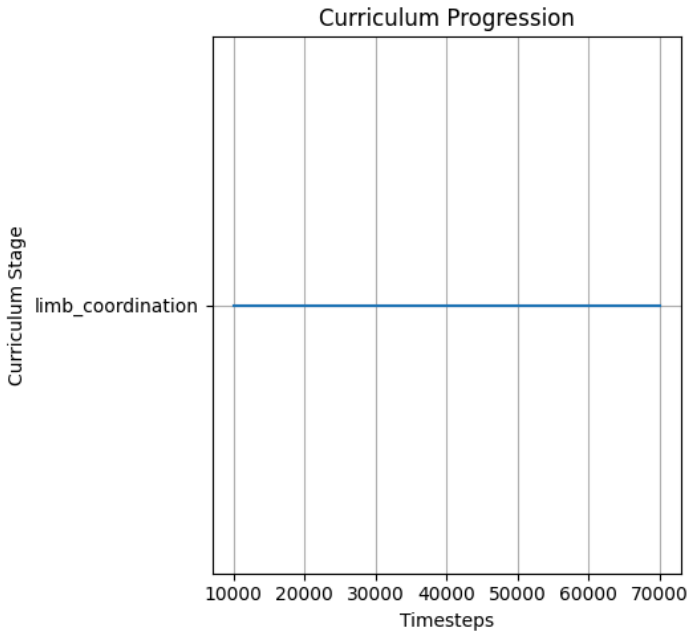



```
/usr/local/lib/python3.11/dist-packages/stable_baselines3/common/evaluation.py:67: UserWarning: Evaluation environment is not wrapped with a ``Monitor`` wrapper. This may result in reporting mo
```

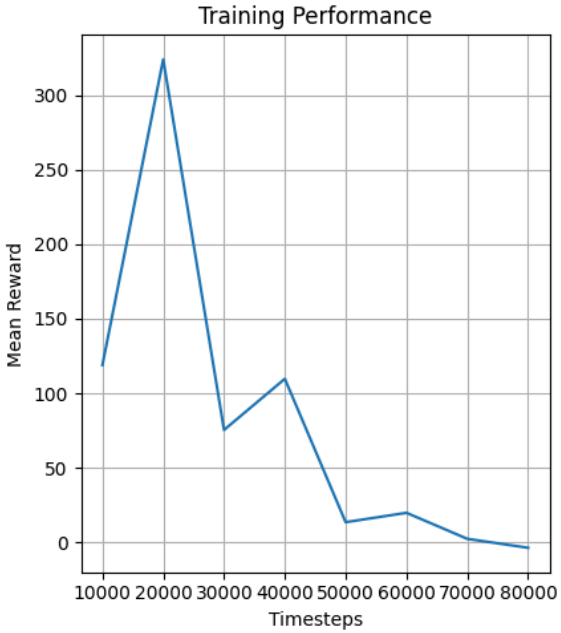
warnings.warn(

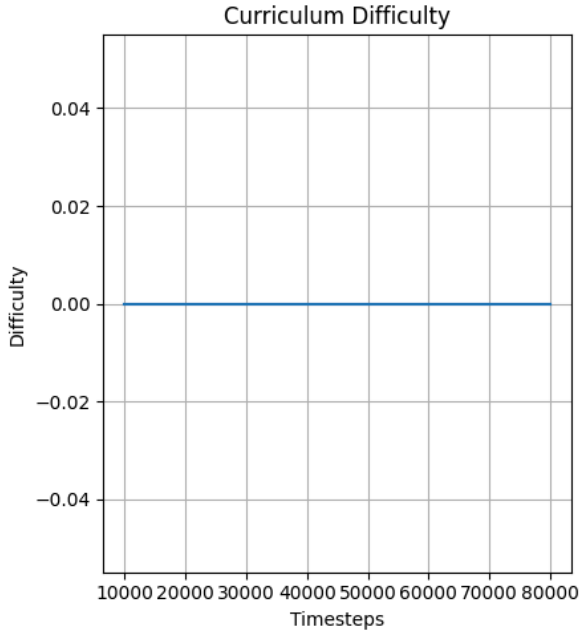


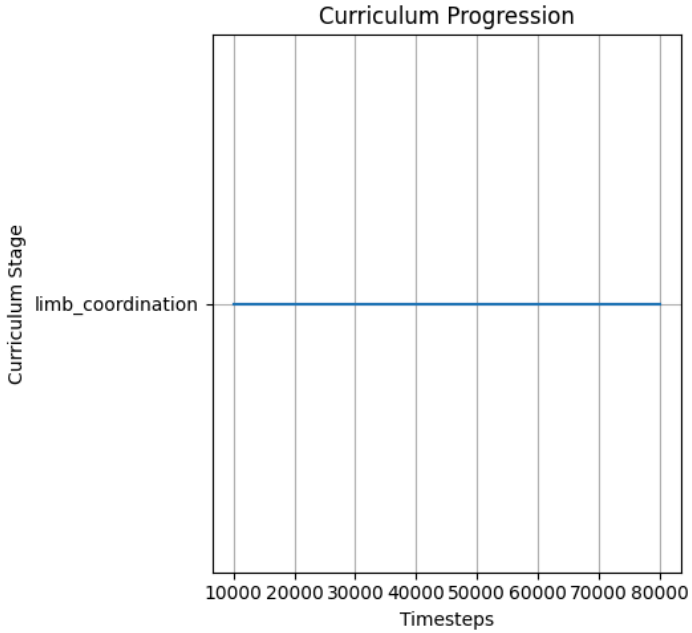




/usr/local/lib/python3.11/dist-packages/stable_baselines3/common/evaluation.py:67: UserWarning: Evaluation environment is not wrapped with a ``Monitor`` wrapper. This may result in reporting mc
warnings.warn(

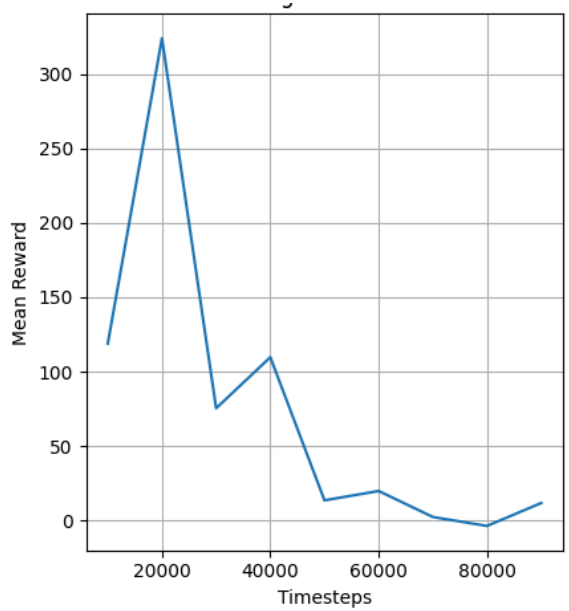


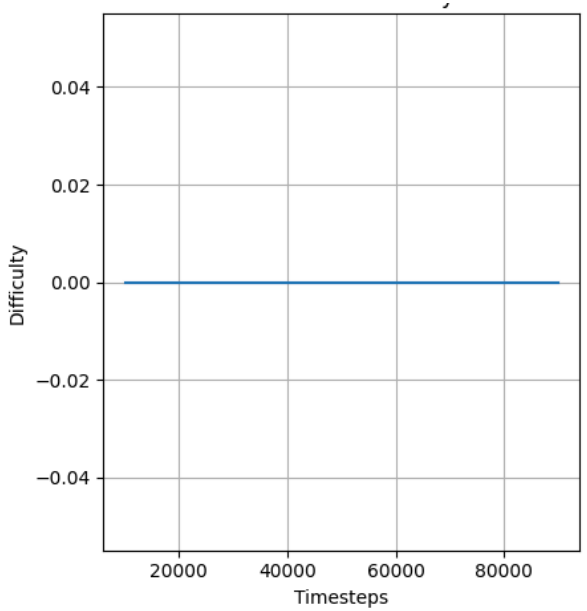


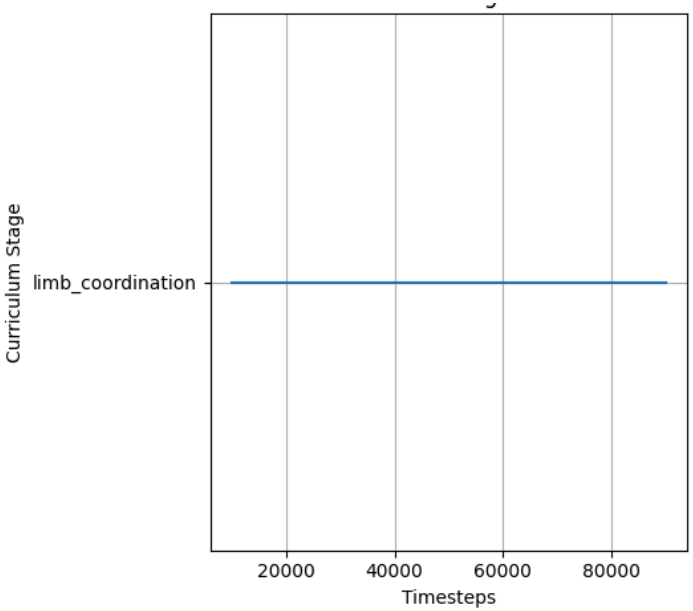


/usr/local/lib/python3.11/dist-packages/stable_baselines3/common/evaluation.py:67: UserWarning: Evaluation environment is not wrapped with a ``Monitor`` wrapper. This may result in reporting mc
warnings.warn(

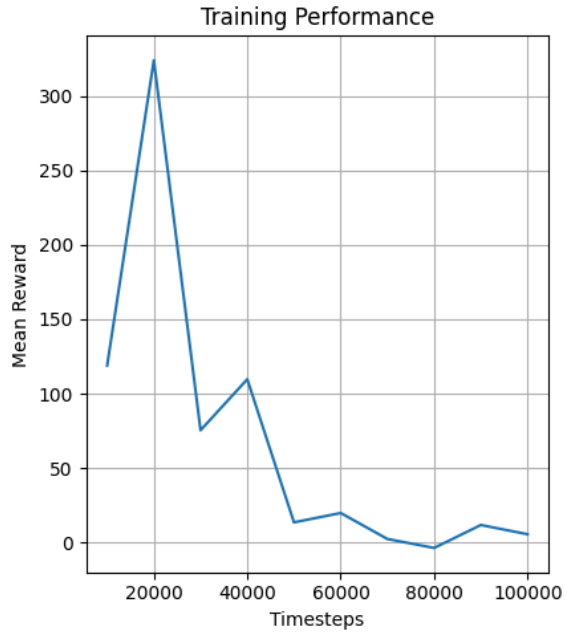
Training Performance Curriculum Difficulty Curriculum Progression

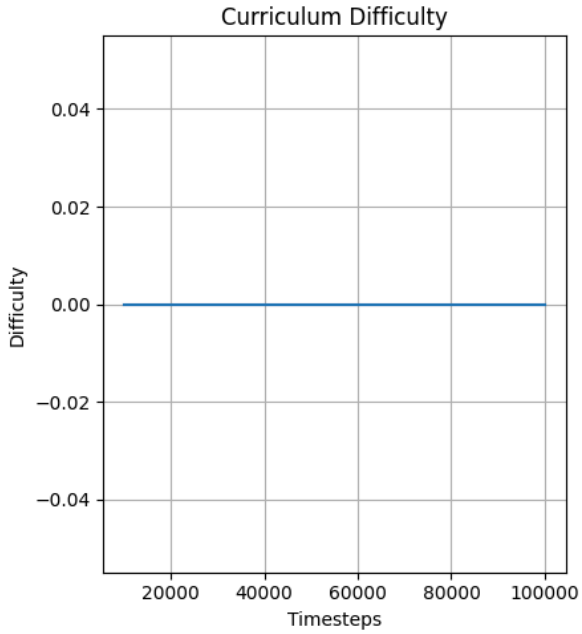


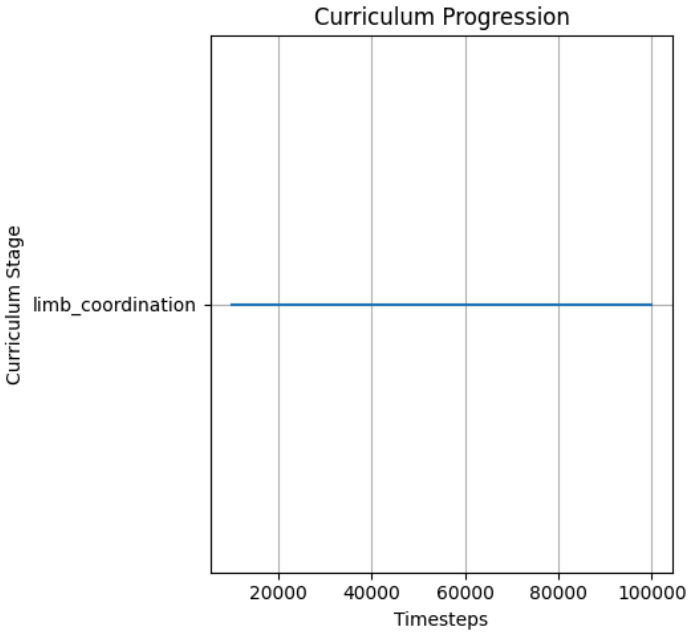




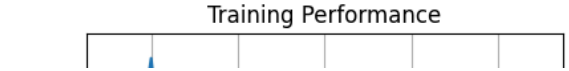

```
/usr/local/lib/python3.11/dist-packages/stable_baselines3/common/evaluation.py:67: UserWarning: Evaluation environment is not wrapped with a ``Monitor`` wrapper. This may result in reporting mo
  warnings.warn(
```

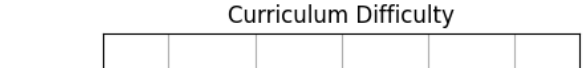


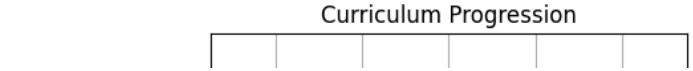


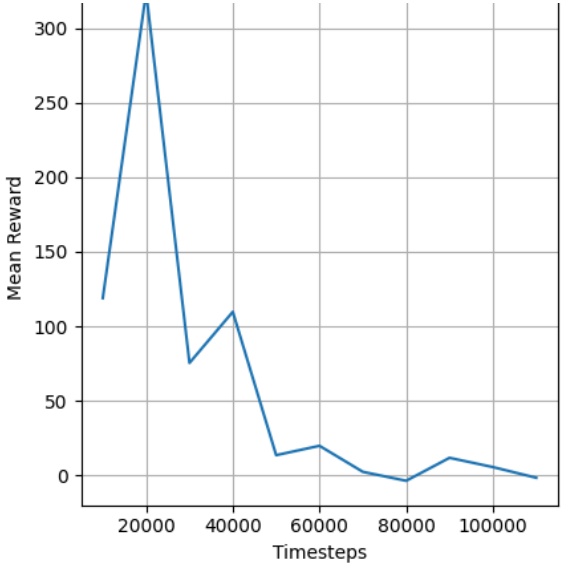


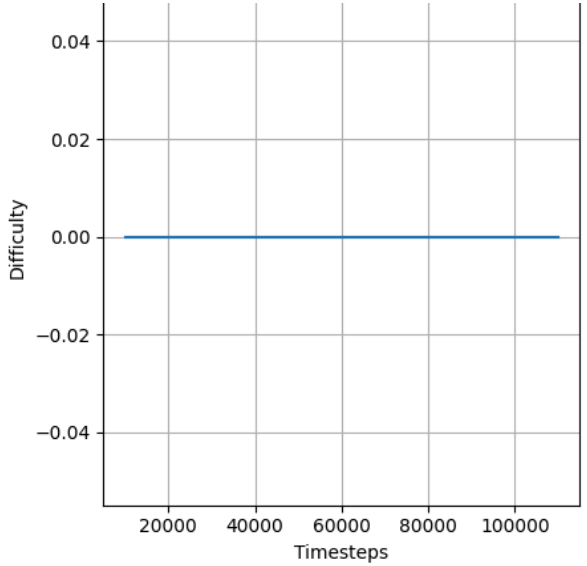

```
/usr/local/lib/python3.11/dist-packages/stable_baselines3/common/evaluation.py:67: UserWarning: Evaluation environment is not wrapped with a ``Monitor`` wrapper. This may result in reporting mo
  warnings.warn(
```

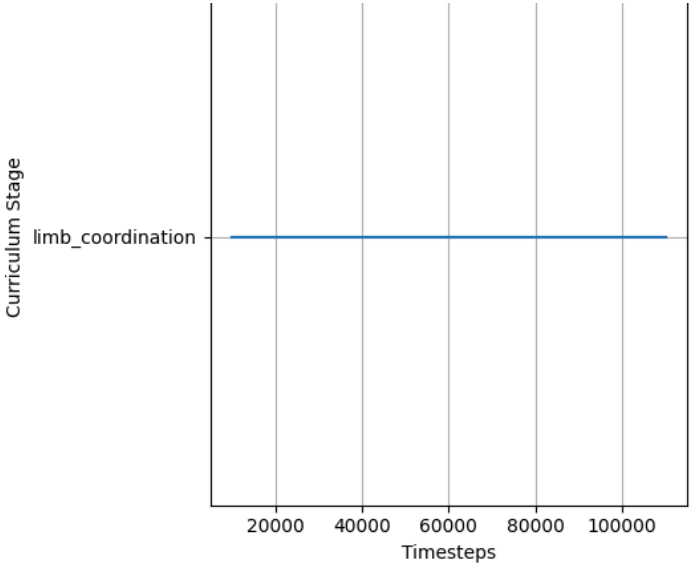




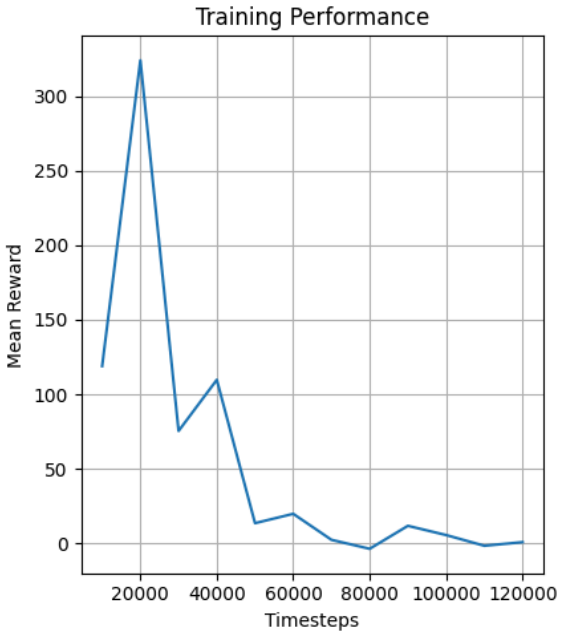


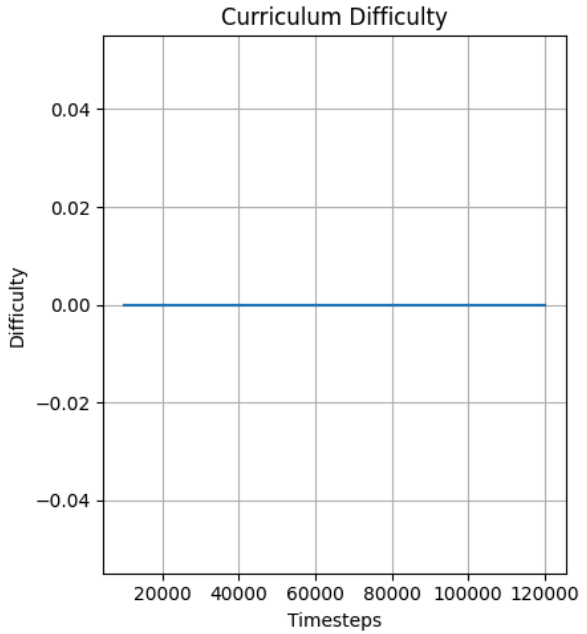


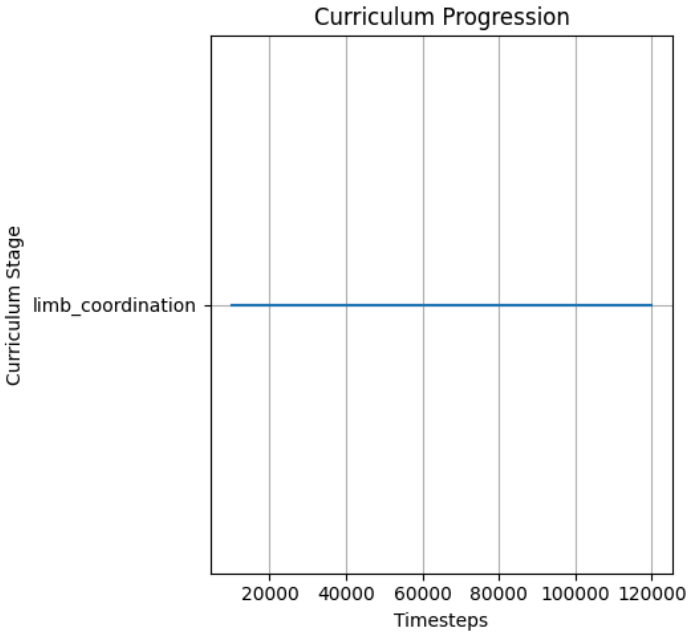




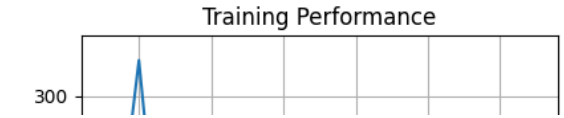

```
/usr/local/lib/python3.11/dist-packages/stable_baselines3/common/evaluation.py:67: UserWarning: Evaluation environment is not wrapped with a ``Monitor`` wrapper. This may result in reporting mo
  warnings.warn(
```

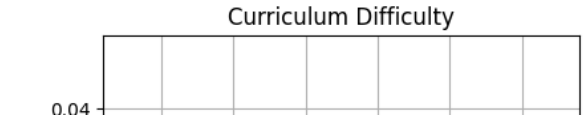


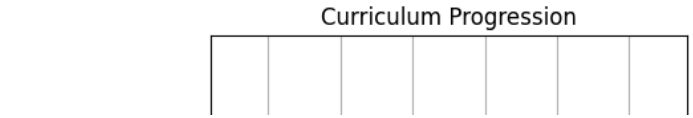


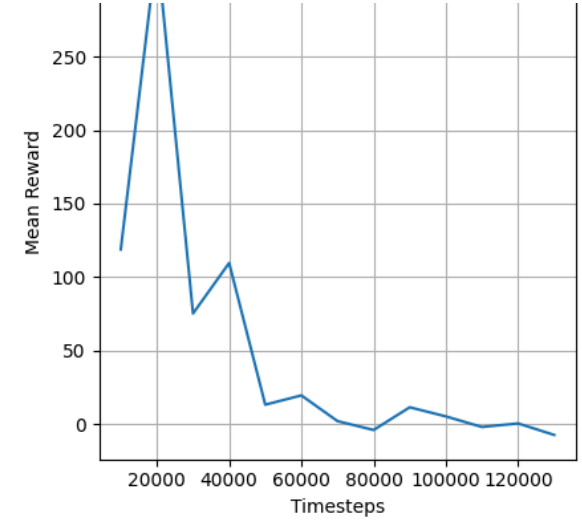


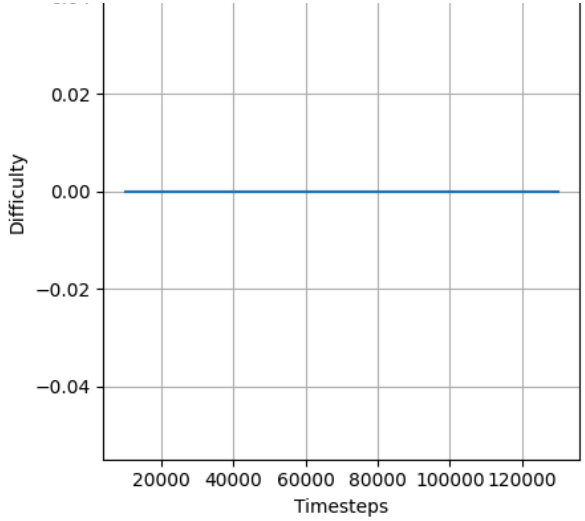

```
/usr/local/lib/python3.11/dist-packages/stable_baselines3/common/evaluation.py:67: UserWarning: Evaluation environment is not wrapped with a ``Monitor`` wrapper. This may result in reporting mo
  warnings.warn(
```

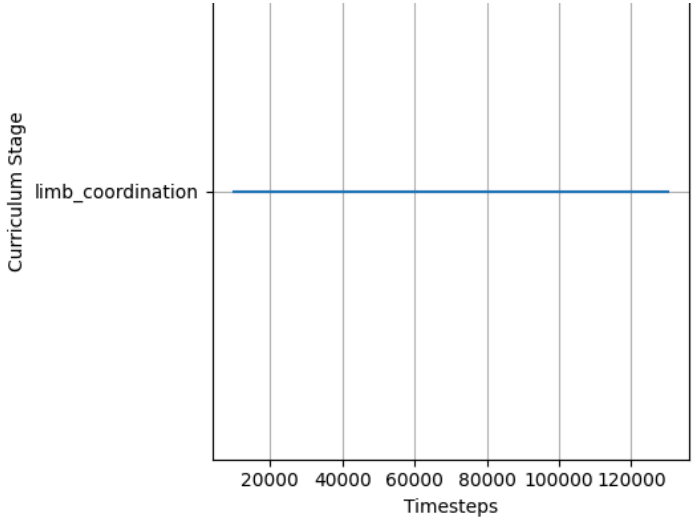




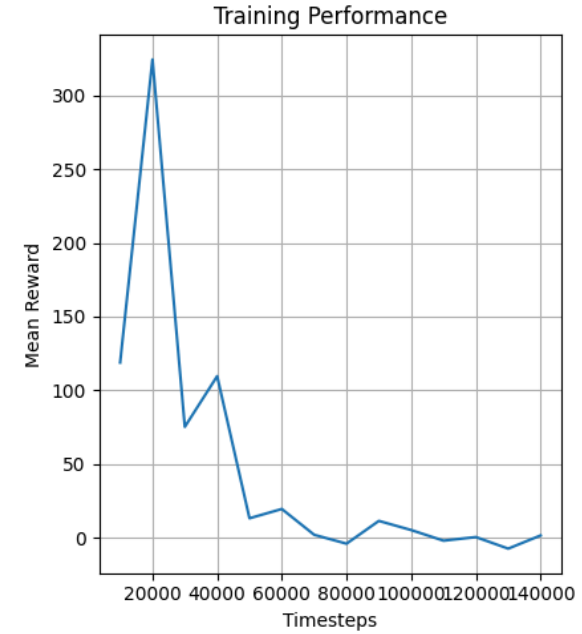


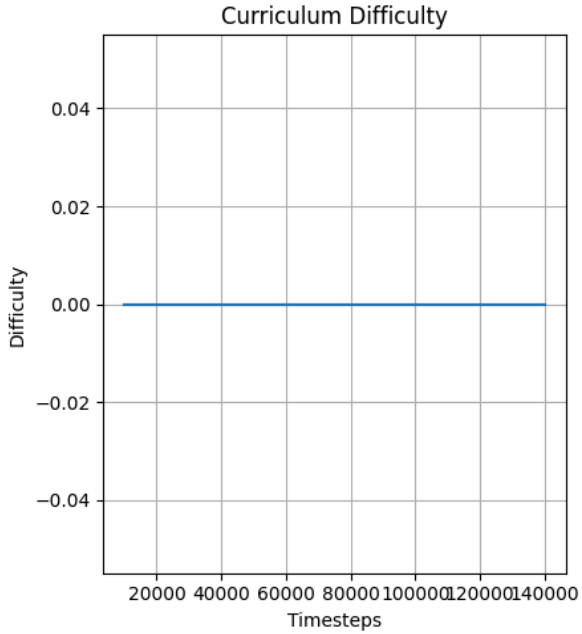


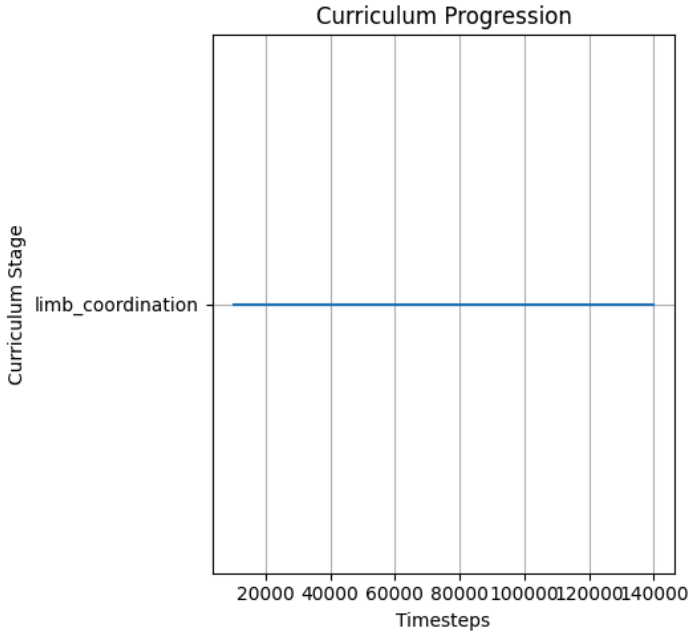




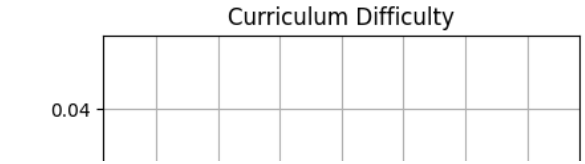

```
/usr/local/lib/python3.11/dist-packages/stable_baselines3/common/evaluation.py:67: UserWarning: Evaluation environment is not wrapped with a ``Monitor`` wrapper. This may result in reporting mo
  warnings.warn(
```

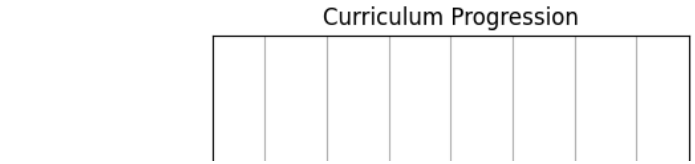


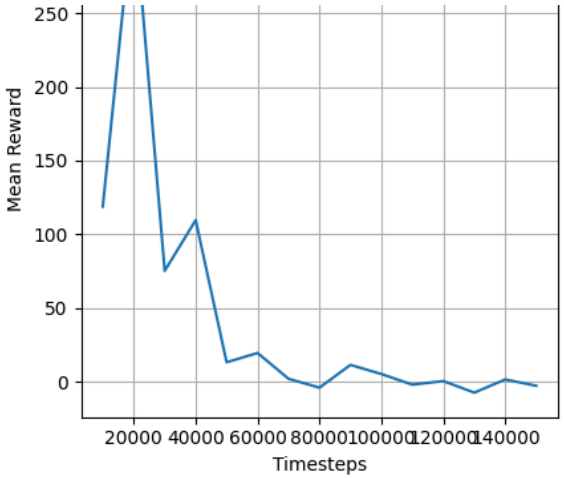


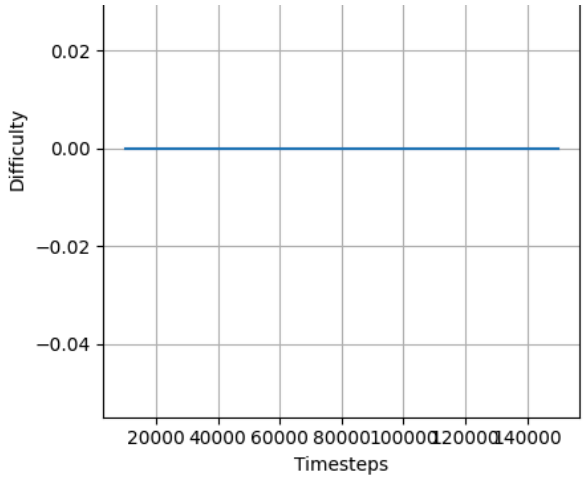


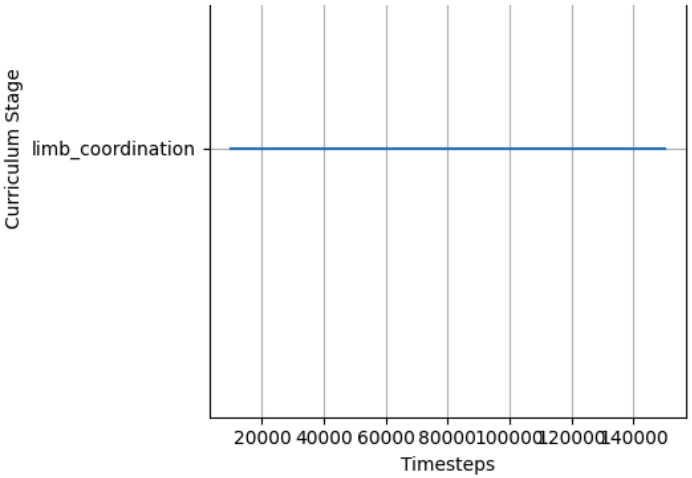

```
/usr/local/lib/python3.11/dist-packages/stable_baselines3/common/evaluation.py:67: UserWarning: Evaluation environment is not wrapped with a ``Monitor`` wrapper. This may result in reporting mo
  warnings.warn(
```



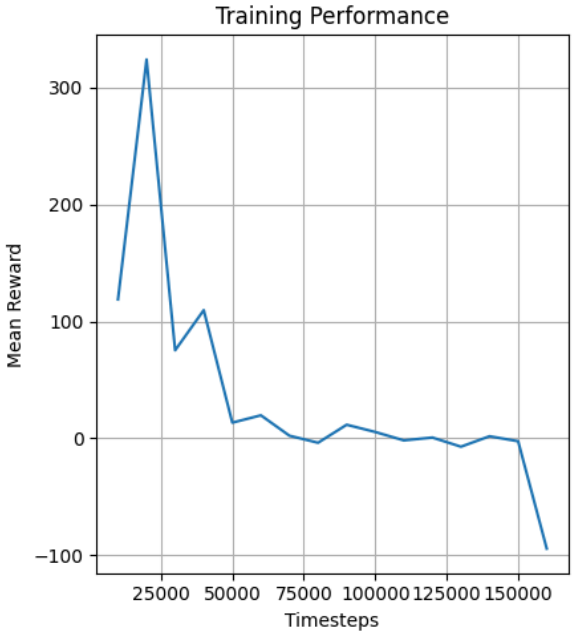


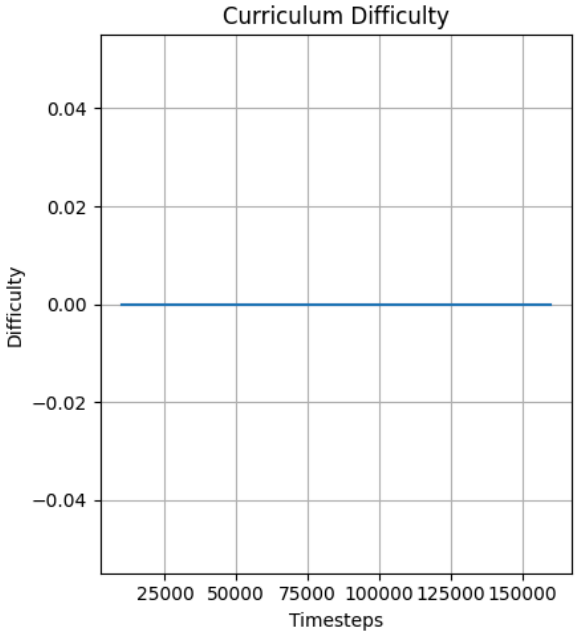


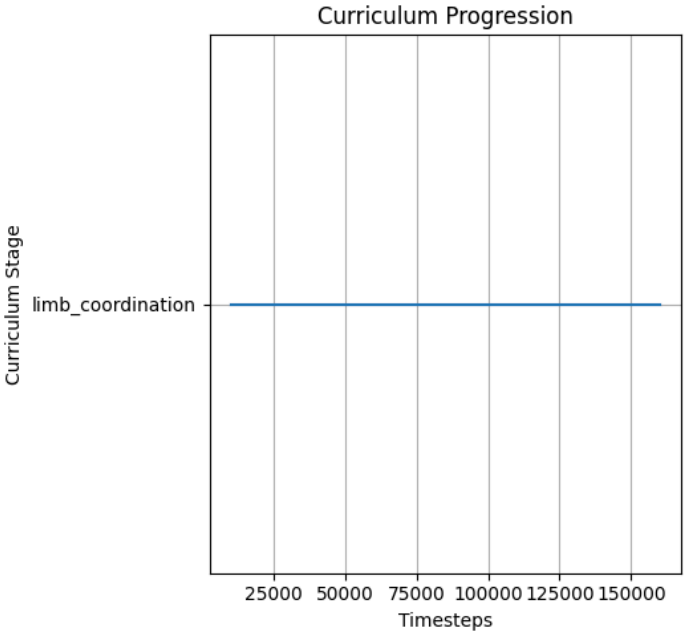




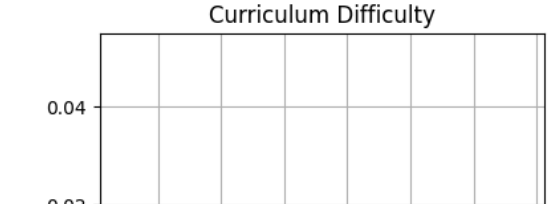

```
/usr/local/lib/python3.11/dist-packages/stable_baselines3/common/evaluation.py:67: UserWarning: Evaluation environment is not wrapped with a ``Monitor`` wrapper. This may result in reporting mo
  warnings.warn(
```

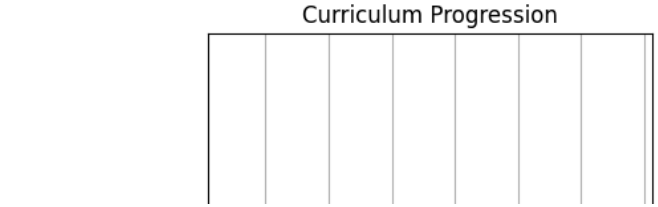


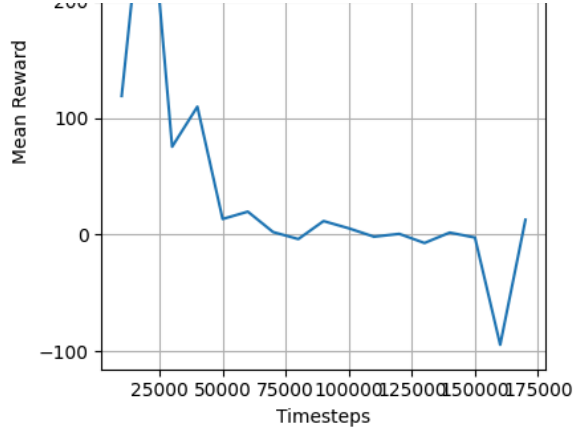


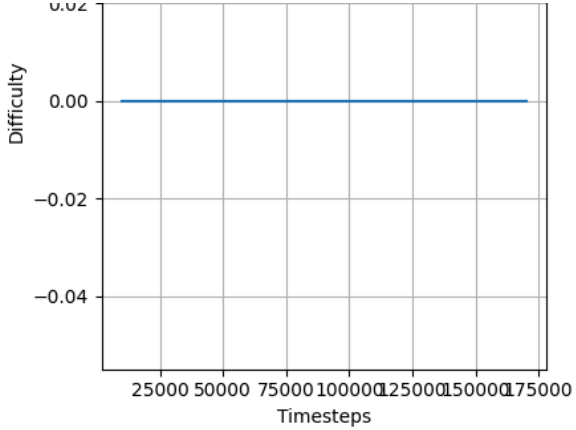


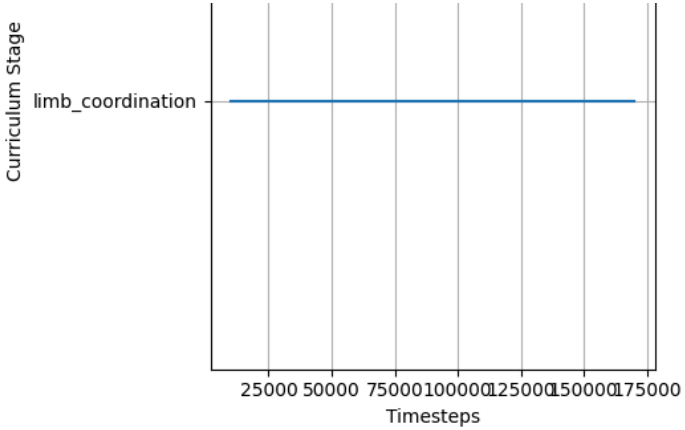

```
/usr/local/lib/python3.11/dist-packages/stable_baselines3/common/evaluation.py:67: UserWarning: Evaluation environment is not wrapped with a ``Monitor`` wrapper. This may result in reporting mo
  warnings.warn(
```



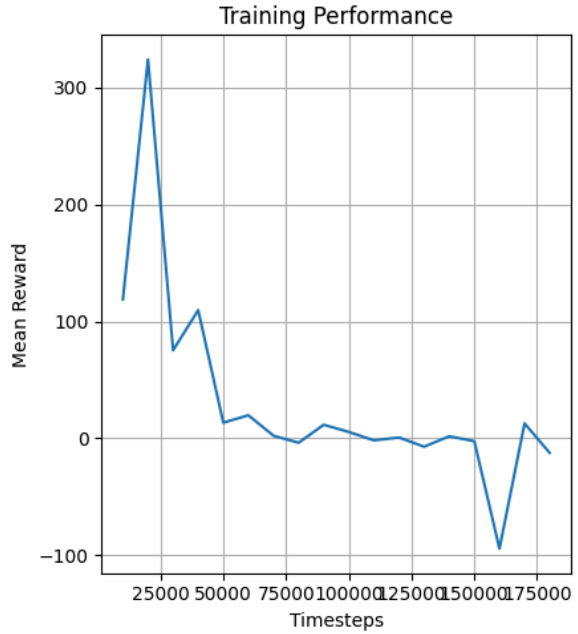


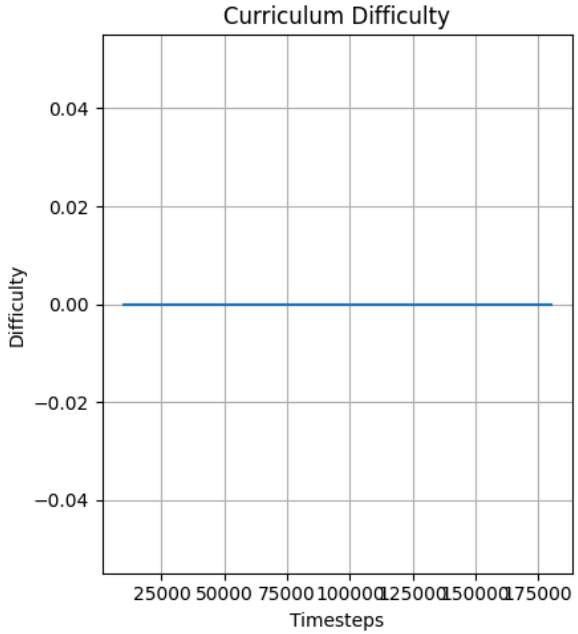


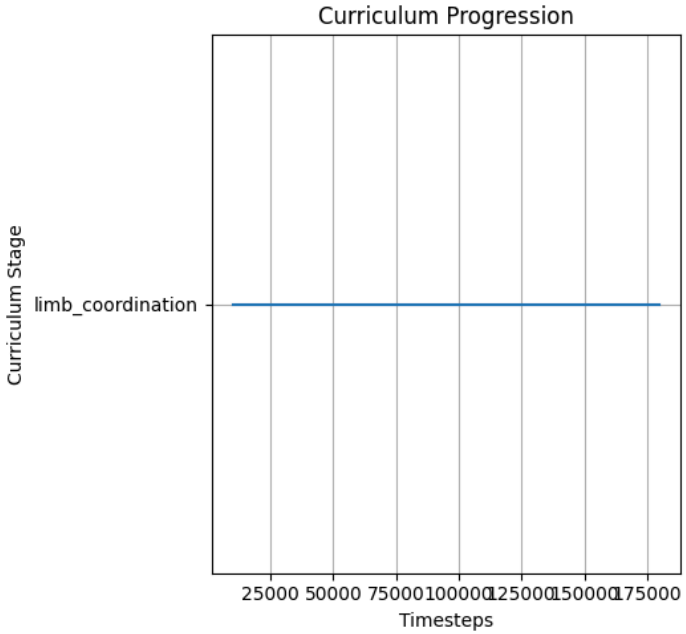




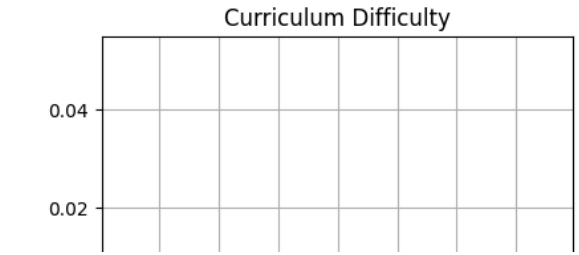

```
/usr/local/lib/python3.11/dist-packages/stable_baselines3/common/evaluation.py:67: UserWarning: Evaluation environment is not wrapped with a ``Monitor`` wrapper. This may result in reporting mo
  warnings.warn(
```

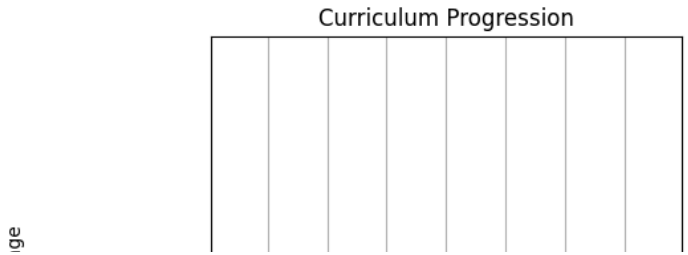


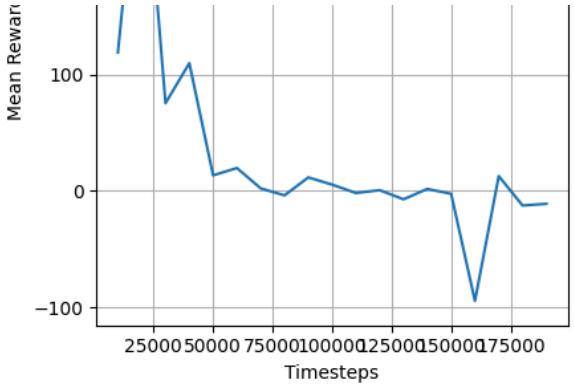


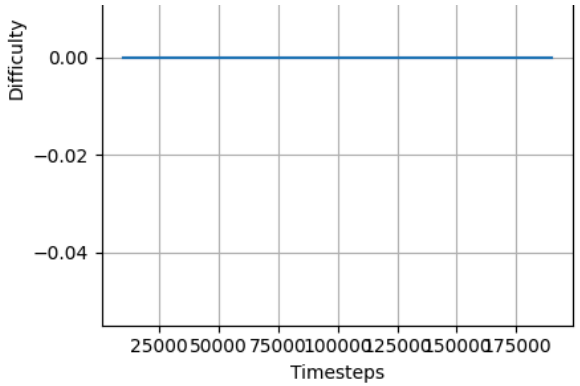


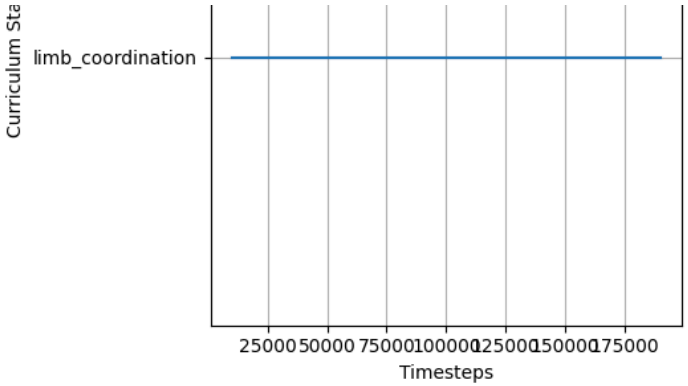

```
/usr/local/lib/python3.11/dist-packages/stable_baselines3/common/evaluation.py:67: UserWarning: Evaluation environment is not wrapped with a ``Monitor`` wrapper. This may result in reporting mo
  warnings.warn(
```



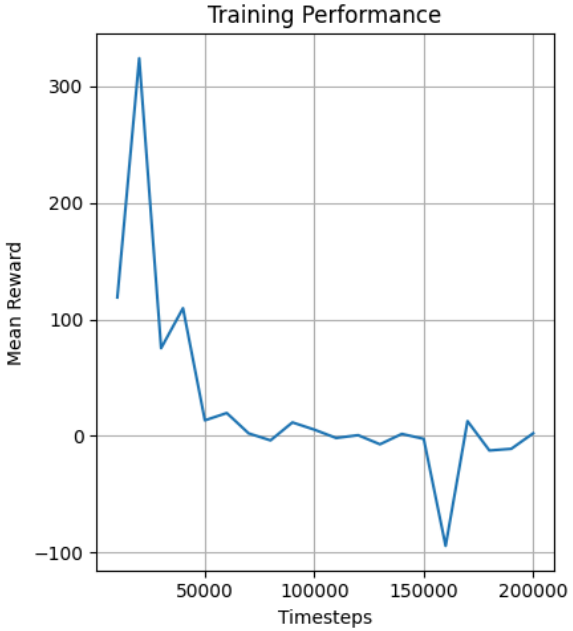


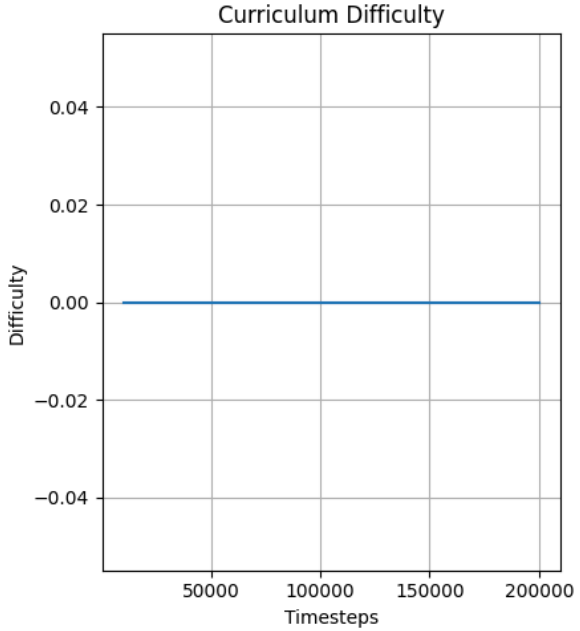


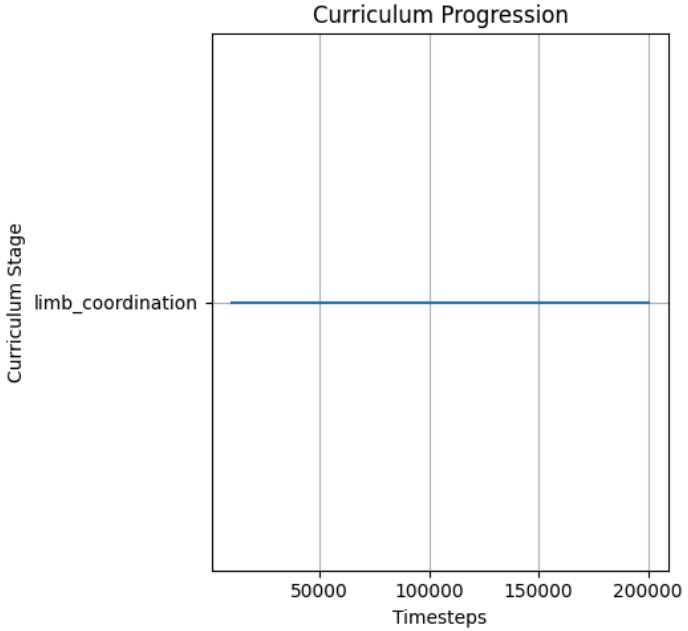




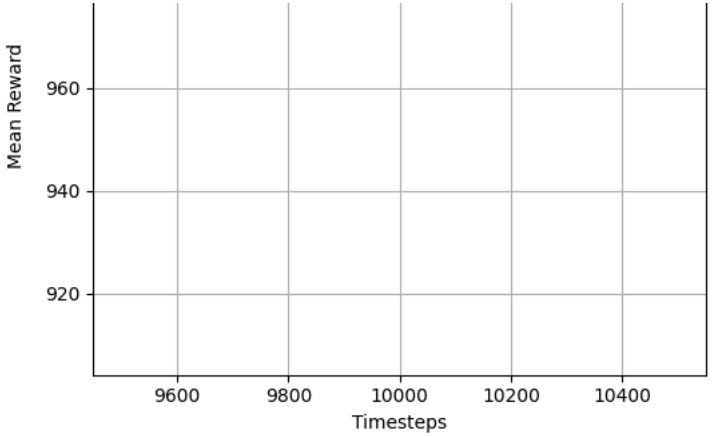

```
/usr/local/lib/python3.11/dist-packages/stable_baselines3/common/evaluation.py:67: UserWarning: Evaluation environment is not wrapped with a ``Monitor`` wrapper. This may result in reporting mc
  warnings.warn(
```

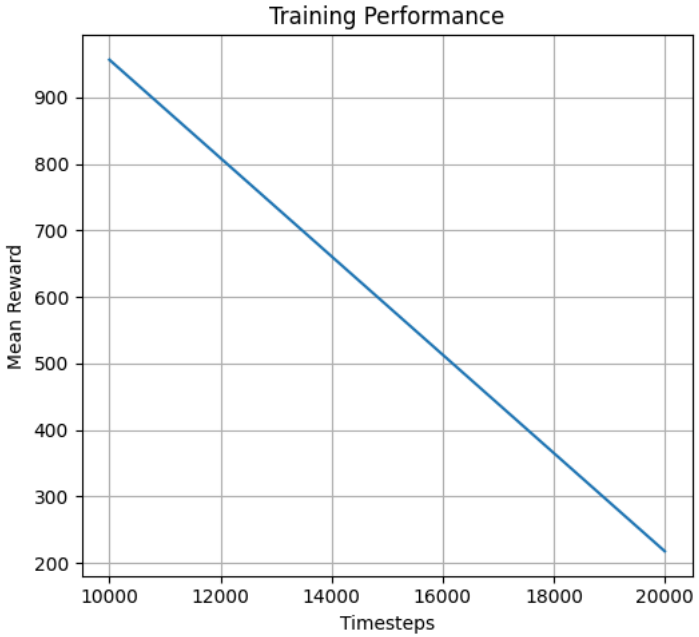




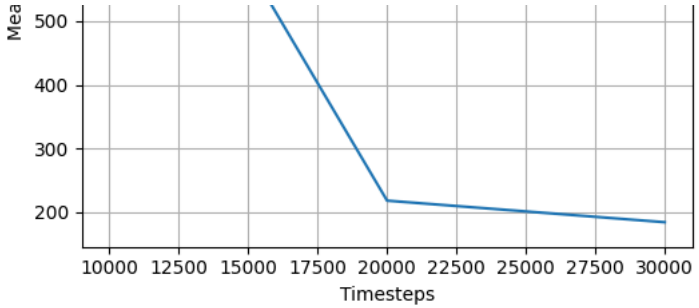


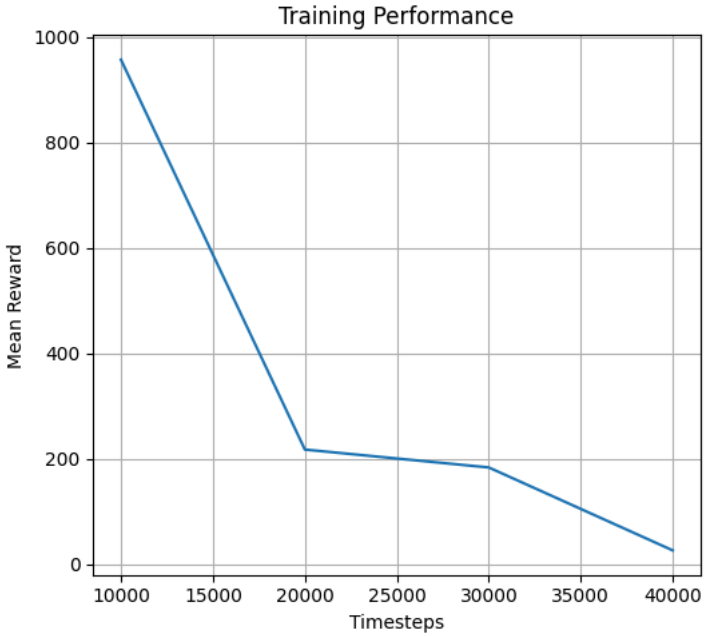

```
Training completed in 6.3 minutes

=== TRAINING STANDARD MODEL ===
Training standard for 200000 steps...
/usr/local/lib/python3.11/dist-packages/stable_baselines3/common/evaluation.py:67: UserWarning: Evaluation environment is not wrapped with a ``Monitor`` wrapper. This may result in reporting mc
  warnings.warn(
```

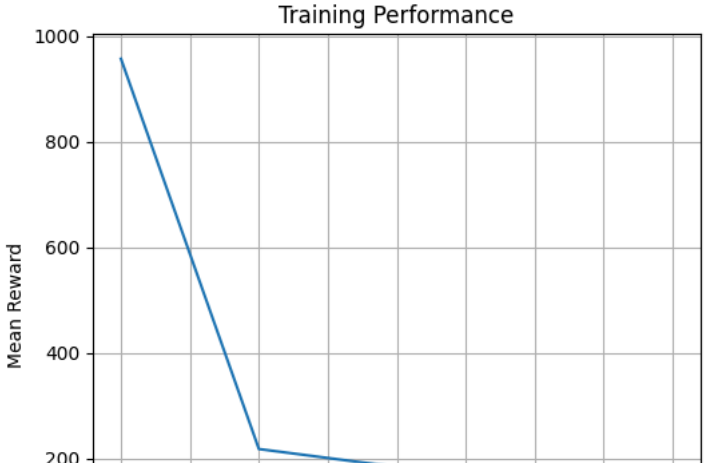

Mean Reward
960
940
920
9600
9800
10000
10200
10400
Timesteps

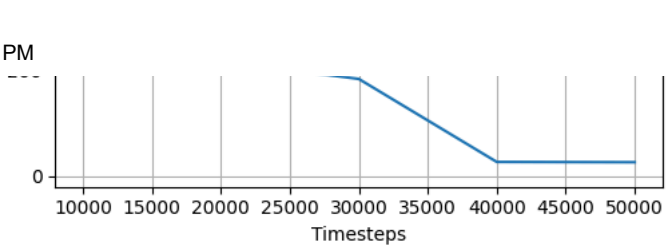

Training Performance
Mean Reward
900
800
700
600
500
400
300
200
10000
12000
14000
16000
18000
20000
Timesteps

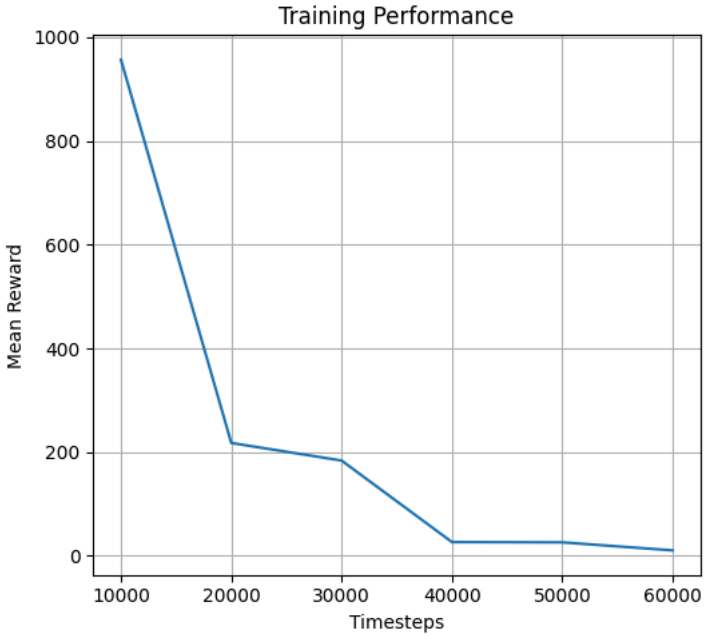

Training Performance
900
800
700
600


Mea
500
400
300
200
10000 12500 15000 17500 20000 22500 25000 27500 30000
Timesteps

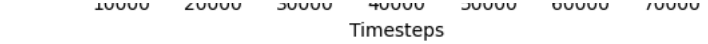

Training Performance
1000
800
600
400
200
0
Mean Reward
10000 15000 20000 25000 30000 35000 40000
Timesteps

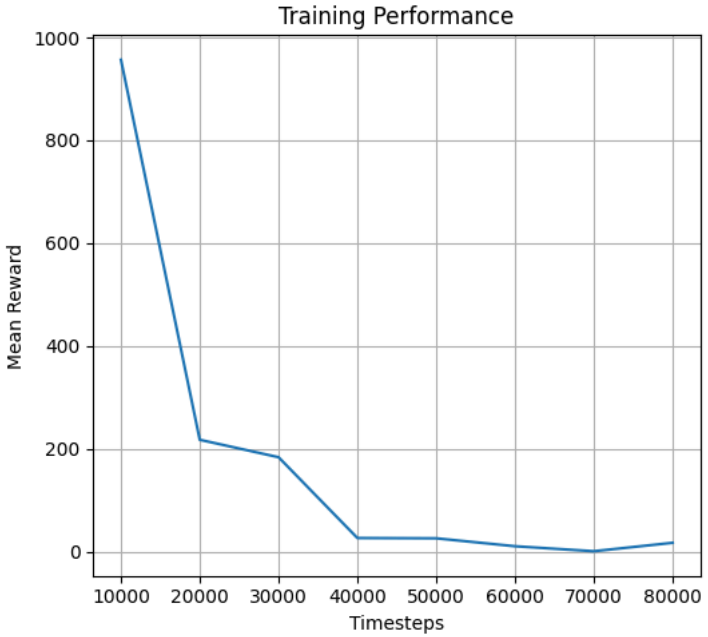

Training Performance
1000
800
600
400
200
Mean Reward

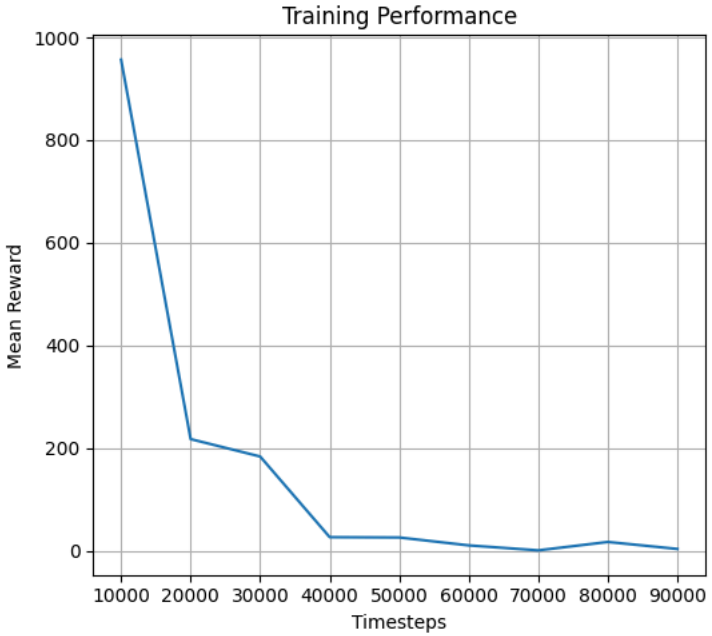

0
10000 15000 20000 25000 30000 35000 40000 45000 50000
Timesteps

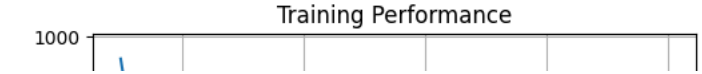

Training Performance
1000
800
600
400
200
0
Mean Reward
10000 20000 30000 40000 50000 60000
Timesteps

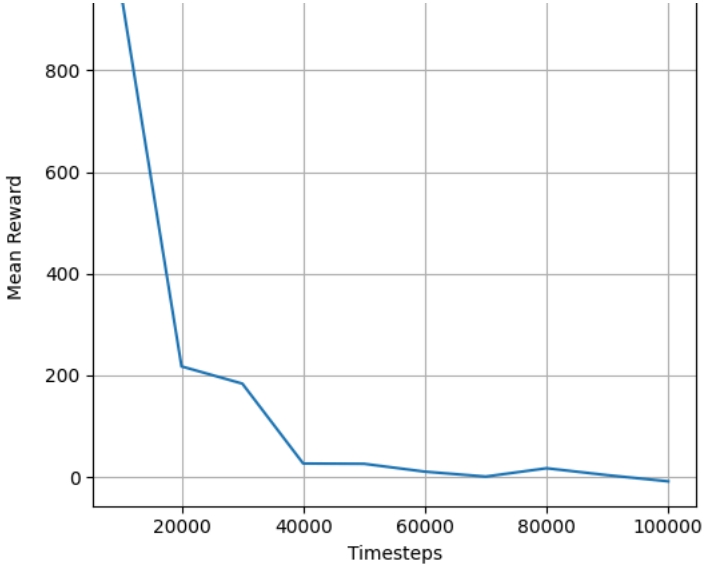

Training Performance
1000
800
600
400
200
0
Mean Reward

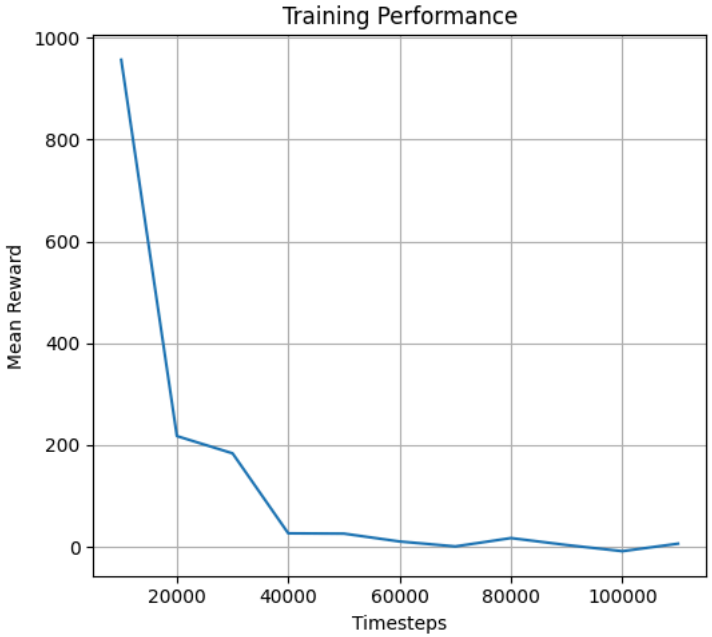

Timesteps


Training Performance
Mean Reward
1000
800
600
400
200
0
10000 20000 30000 40000 50000 60000 70000 80000
Timesteps

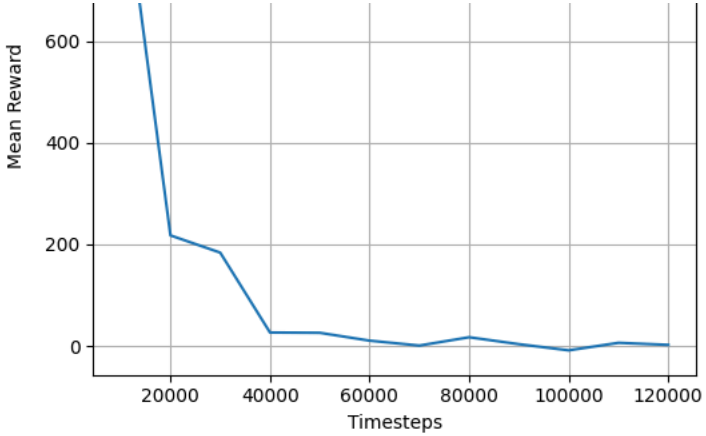

Training Performance
Mean Reward
1000
800
600
400
200
0
10000 20000 30000 40000 50000 60000 70000 80000 90000
Timesteps

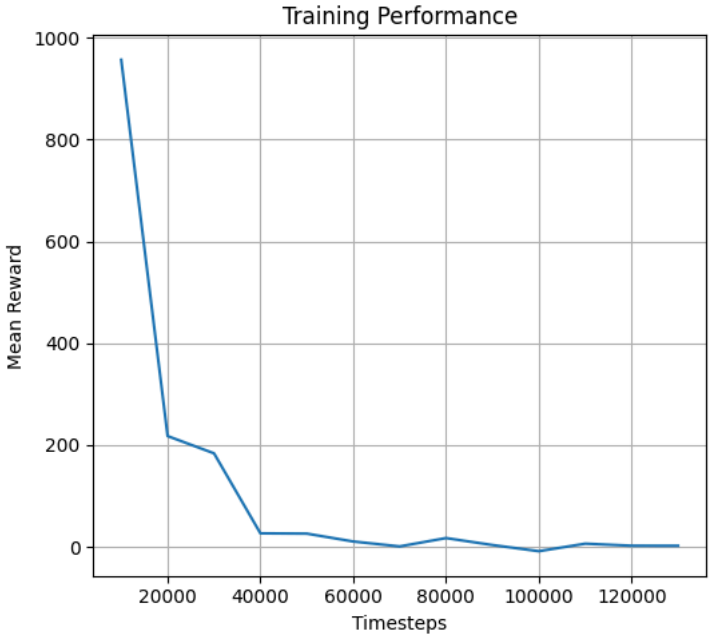

Training Performance
1000


800
600
400
200
0
Mean Reward
20000
40000
60000
80000
100000
Timesteps

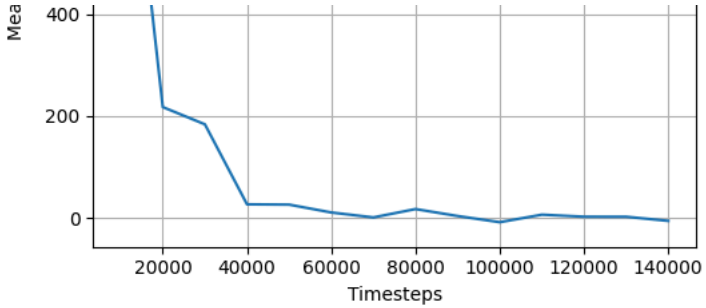

Training Performance
1000
800
600
400
200
0
Mean Reward
20000
40000
60000
80000
100000
Timesteps

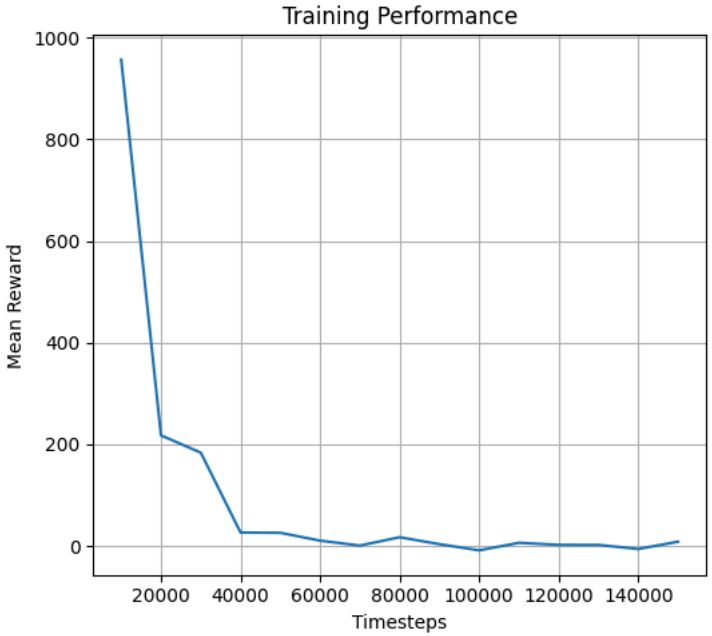

Training Performance
1000
800

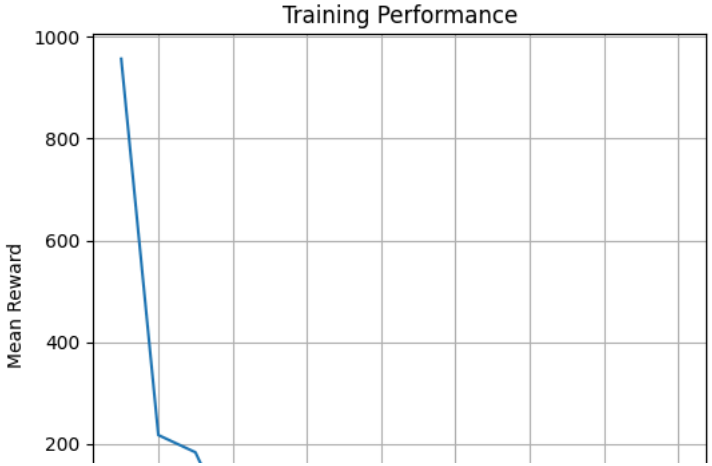

Mean Reward
600
400
200
0
20000
40000
60000
80000
100000
120000
Timesteps

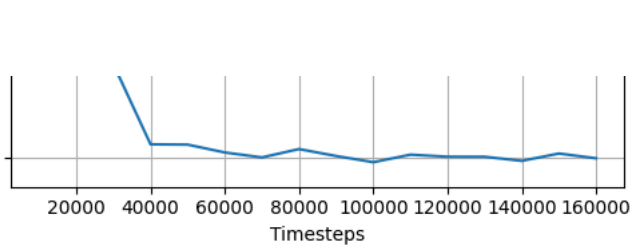

Training Performance
Mean Reward
1000
800
600
400
200
0
20000
40000
60000
80000
100000
120000
Timesteps

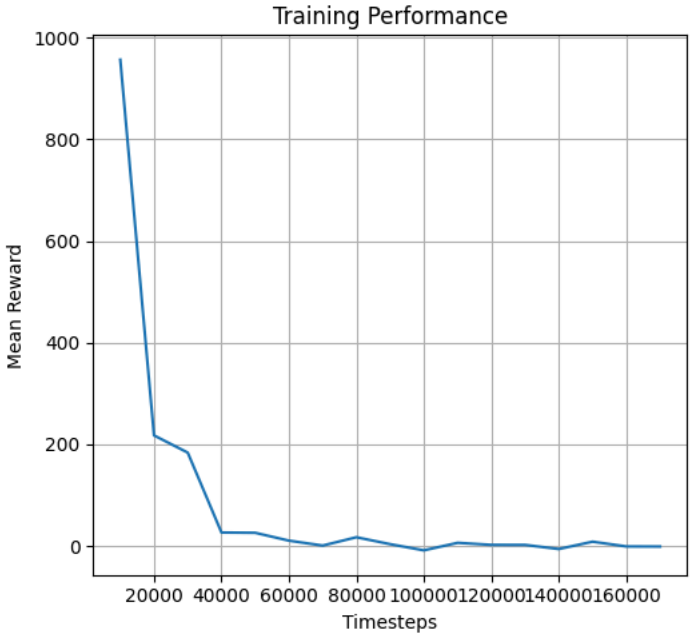

Training Performance
1000
800
600

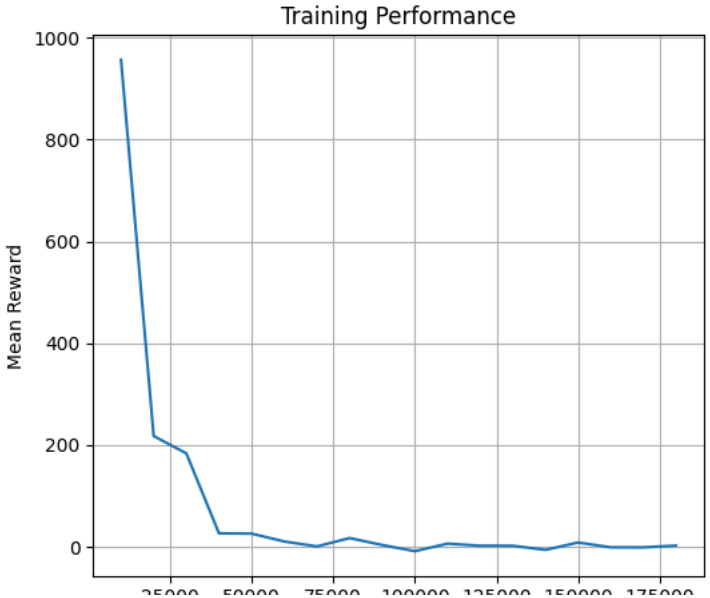

Mea
400
200
0
20000
40000
60000
80000
100000
120000
140000
Timesteps

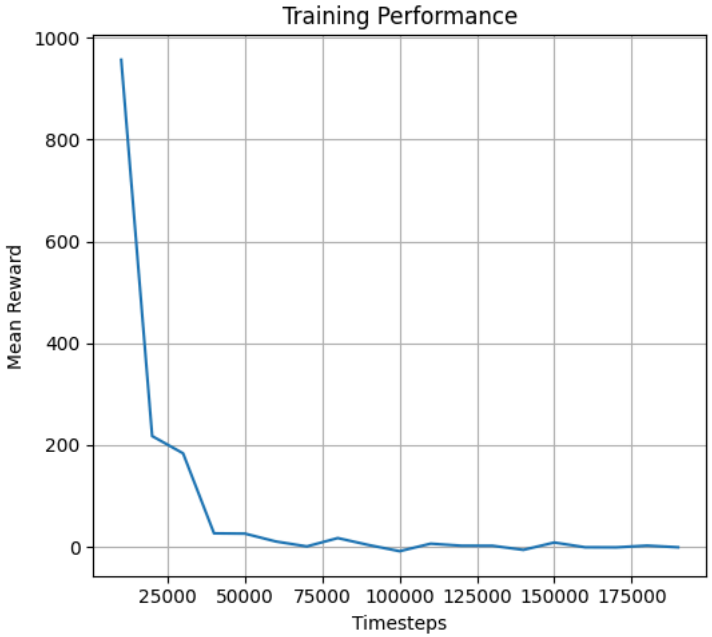

Training Performance
Mean Reward
1000
800
600
400
200
0
20000
40000
60000
80000
100000
120000
140000
Timesteps

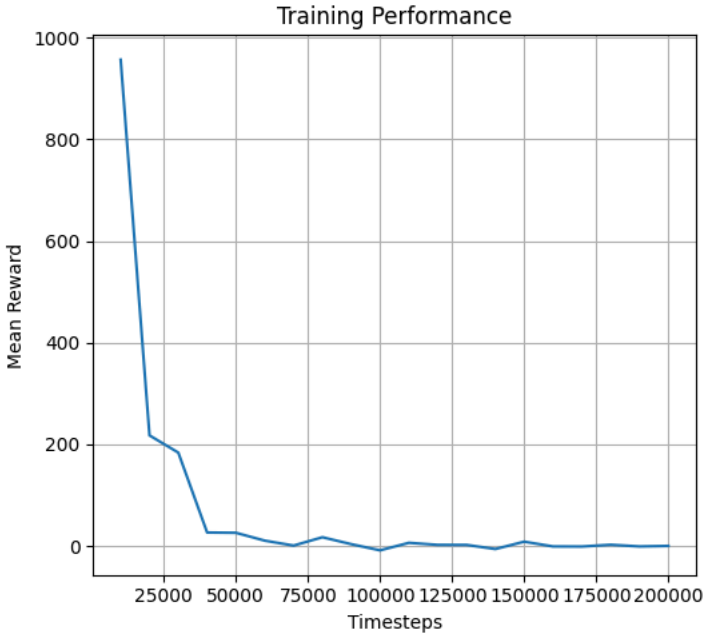

Training Performance
Mean Reward
1000
800
600
400
200


0
20000 40000 60000 80000 100000 120000 140000 160000
Timesteps


Training Performance
1000
800
600
400
200
0
Mean Reward
20000 40000 60000 80000 100000120000140000160000
Timesteps

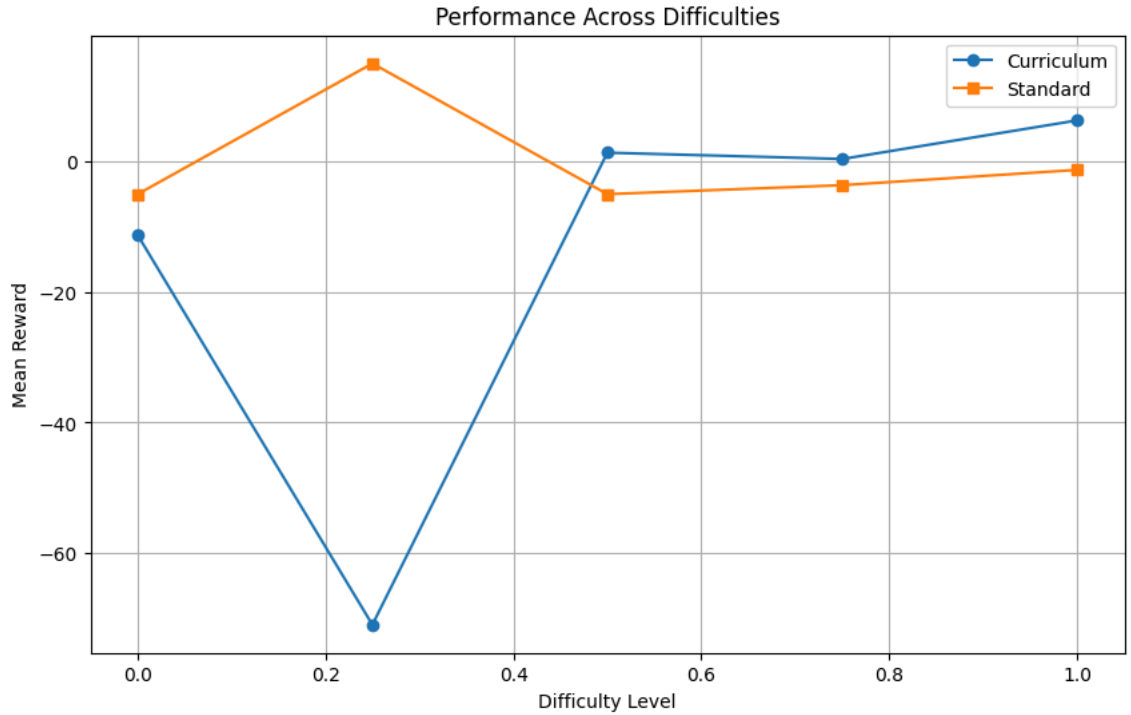

Training Performance
1000
800
600
400
200
0
Mean Reward






```
Training completed in 6.2 minutes

=== COMPARING MODELS ===
```

```
Evaluating models at different difficulties...
Initialized ProgressiveAntCurriculum with stage: limb_coordination
Initialized ProgressiveAntCurriculum with stage: limb_coordination
Difficulty 0.00: Curriculum=-11.3, Standard=-5.1
Initialized ProgressiveAntCurriculum with stage: limb_coordination
Initialized ProgressiveAntCurriculum with stage: limb_coordination
Difficulty 0.25: Curriculum=-70.9, Standard=14.9
Initialized ProgressiveAntCurriculum with stage: limb_coordination
Initialized ProgressiveAntCurriculum with stage: limb_coordination
Difficulty 0.50: Curriculum=1.3, Standard=-5.1
Initialized ProgressiveAntCurriculum with stage: limb_coordination
Initialized ProgressiveAntCurriculum with stage: limb_coordination
Difficulty 0.75: Curriculum=0.3, Standard=-3.7
Initialized ProgressiveAntCurriculum with stage: limb_coordination
Initialized ProgressiveAntCurriculum with stage: limb_coordination
Difficulty 1.00: Curriculum=6.2, Standard=-1.4
```




```
Results:
- Curriculum average reward: -14.9
- Standard average reward: -0.1
- Improvement: -25473.0%

Robustness:
- Curriculum performance drop: -17.5
- Standard performance drop: -3.7
- Robustness improvement: -367.2%

=== VISUALIZING CURRICULUM MODEL ===
Initialized ProgressiveAntCurriculum with stage: limb_coordination
```

0:00 / 0:12

```
=== VISUALIZING STANDARD MODEL ===
Initialized ProgressiveAntCurriculum with stage: limb_coordination
```

0:00 / 0:12

```
# Enhanced Curriculum Learning for Ant-v5 with Progressive Task Decomposition
# For Google Colab with visualization

# 1. Install dependencies
!apt-get install -y xvfb python-opengl ffmpeg > /dev/null 2>&1
!pip install gymnasium[mujoco] stable-baselines3 matplotlib pyvirtualdisplay > /dev/null 2>&1

# 2. Set up virtual display
from pyvirtualdisplay import Display
display = Display(visible=0, size=(1400, 900))
display.start()

# 3. Import required libraries
import os
import time
import numpy as np
import matplotlib.pyplot as plt
from IPython.display import display, clear_output, HTML
from base64 import b64encode
import gymnasium as gym
from stable_baselines3 import PPO
from stable_baselines3.common.callbacks import BaseCallback
from stable_baselines3.common.vec_env import DummyVecEnv, VecNormalize
from stable_baselines3.common.evaluation import evaluate_policy
from typing import List, Tuple, Optional

# 4. Create directories for saving
os.makedirs("./models", exist_ok=True)
os.makedirs("./plots", exist_ok=True)
os.makedirs("./vec_normalize", exist_ok=True)

# 5. Learning rate schedule function
def linear_schedule(initial_value):
    """Linear learning rate schedule."""
    def func(progress_remaining):
        return progress_remaining * initial_value
    return func

# 6. Enhanced Ant Curriculum Wrapper
class ProgressiveAntCurriculum(gym.Wrapper):
    def __init__(self,
                 env: gym.Env,
                 initial_difficulty: float = 0.1,  # Start with a slightly higher initial difficulty
                 max_difficulty: float = 1.0,
                 adaptation_speed: float = 0.005,  # Slower adaptation for stability
                 success_threshold: float = 0.7,   # More achievable threshold
                 window_size: int = 50,            # Larger window for stable assessment
                 curriculum_stages: List[Tuple[str, float]] = None):
        """
        Enhanced curriculum wrapper with progressive task decomposition.

        Args:
            env: Gym environment to wrap
            initial_difficulty: Starting difficulty (0-1)
            max_difficulty: Maximum difficulty level
            adaptation_speed: How quickly to adjust difficulty
```

```
            success_threshold: Performance threshold for increasing difficulty
            window_size: Number of episodes to consider for performance evaluation
            curriculum_stages: List of (stage_name, target_difficulty) pairs
        """
        super().__init__(env)

        # Curriculum parameters
        self.difficulty = initial_difficulty
        self.max_difficulty = max_difficulty
        self.adaptation_speed = adaptation_speed
        self.success_threshold = success_threshold
        self.window_size = window_size

        # Curriculum stages with more gradual progression
        self.curriculum_stages = curriculum_stages or [
            ("basic_balance", 0.15),       # Just learn to balance
            ("limb_coordination", 0.35),    # Coordinate limbs for basic movement
            ("forward_movement", 0.55),     # Focus on consistent forward movement
            ("robust_movement", 0.75),      # Handle mild perturbations
            ("full_task", 1.0)              # Handle full task complexity
        ]
        self.current_stage = 0

        # Tracking
        self.recent_rewards = []
        self.episode_count = 0
        self.difficulty_history = []
        self.stage_history = []
        self.episode_rewards = 0

        # Environment modifications
        self.original_init_params = self._get_env_params()
        self._apply_curriculum_modifications()

        print(f"Initialized ProgressiveAntCurriculum with stage: {self.curriculum_stages[self.current_stage][0]} (difficulty: {self.difficulty:.2f})")

    def _get_env_params(self) -> dict:
        """Get original environment parameters for reference"""
        return {
            'gravity': self.env.unwrapped.model.opt.gravity.copy(),
            'torso_mass': self.env.unwrapped.model.body_mass[1].copy(),
            'joint_damping': self.env.unwrapped.model.dof_damping[:8].copy(),
            'ctrl_range': self.env.unwrapped.model.actuator_ctrlrange[:8].copy(),
            'friction': self.env.unwrapped.model.geom_friction.copy()
        }

    def _apply_curriculum_modifications(self):
        """Apply modifications based on current curriculum stage"""
        stage_name, target_diff = self.curriculum_stages[self.current_stage]

        # Reset to original parameters first
        self._reset_to_original()

        # Apply stage-specific modifications
        if stage_name == "basic_balance":
            # Very easy: higher damping, stronger actions, lower gravity
```

```
            self.env.unwrapped.model.dof_damping[:8] *= 2.0  # Higher damping for stability
            self.env.unwrapped.model.opt.gravity[2] *= 0.8   # Lower gravity
            # Increase friction for better ground contact
            self.env.unwrapped.model.geom_friction[:, 0] *= 1.5

        elif stage_name == "limb_coordination":
            # Focus on limb coordination with moderate physics
            self.env.unwrapped.model.dof_damping[:8] *= 1.5  # Still higher damping
            self.env.unwrapped.model.opt.gravity[2] *= 0.9   # Slightly lower gravity
            self.env.unwrapped.model.geom_friction[:, 0] *= 1.2  # More friction

        elif stage_name == "forward_movement":
            # Normal physics with slightly easier control
            self.env.unwrapped.model.dof_damping[:8] *= 1.1  # Bit more damping
            self.env.unwrapped.model.geom_friction[:, 0] *= 1.1  # Bit more friction

        elif stage_name == "robust_movement":
            # Slightly harder physics
            self.env.unwrapped.model.opt.gravity[2] *= 1.05  # Small gravity increase
            self.env.unwrapped.model.body_mass[1] *= 1.05    # Small mass increase

        elif stage_name == "full_task":
            # Original task parameters with slight challenge
            self.env.unwrapped.model.opt.gravity[2] *= 1.1   # Increased gravity

        # Apply difficulty scaling within stage
        self._scale_difficulty(target_diff)

    def _reset_to_original(self):
        """Reset environment parameters to original values"""
        params = self.original_init_params
        self.env.unwrapped.model.opt.gravity[:] = params['gravity']
        self.env.unwrapped.model.body_mass[1] = params['torso_mass']
        self.env.unwrapped.model.dof_damping[:8] = params['joint_damping']
        self.env.unwrapped.model.actuator_ctrlrange[:8] = params['ctrl_range']
        self.env.unwrapped.model.geom_friction[:] = params['friction']

    def _scale_difficulty(self, target_diff: float):
        """Scale parameters based on difficulty within stage"""
        # Linearly interpolate between stage parameters, with smoother transitions
        if target_diff == 0:  # Avoid division by zero
            interp = 0
        else:
            interp = min(self.difficulty / target_diff, 1.0)

        stage_name = self.curriculum_stages[self.current_stage][0]

        # Adjust parameters based on stage and difficulty
        if stage_name == "basic_balance":
            # Gradually reduce the damping assistance as difficulty increases
            damping_factor = 2.0 - interp * 0.5
            self.env.unwrapped.model.dof_damping[:8] = (
                self.original_init_params['joint_damping'] * damping_factor)

            # Gradually normalize gravity
            gravity_factor = 0.8 + interp * 0.2
```

```
                self.env.unwrapped.model.opt.gravity[2] = (
                    self.original_init_params['gravity'][2] * gravity_factor)

            elif stage_name == "limb_coordination":
                # Gradually normalize parameters
                damping_factor = 1.5 - interp * 0.5
                self.env.unwrapped.model.dof_damping[:8] = (
                    self.original_init_params['joint_damping'] * damping_factor)

                gravity_factor = 0.9 + interp * 0.1
                self.env.unwrapped.model.opt.gravity[2] = (
                    self.original_init_params['gravity'][2] * gravity_factor)

            elif stage_name == "forward_movement":
                # Approach normal physics
                damping_factor = 1.1 - interp * 0.1
                self.env.unwrapped.model.dof_damping[:8] = (
                    self.original_init_params['joint_damping'] * damping_factor)

            elif stage_name == "robust_movement":
                # Gradually increase challenge
                gravity_factor = 1.0 + interp * 0.05
                self.env.unwrapped.model.opt.gravity[2] = (
                    self.original_init_params['gravity'][2] * gravity_factor)

                mass_factor = 1.0 + interp * 0.05
                self.env.unwrapped.model.body_mass[1] = (
                    self.original_init_params['torso_mass'] * mass_factor)

            elif stage_name == "full_task":
                # Gradually introduce more challenge
                gravity_factor = 1.0 + interp * 0.1
                self.env.unwrapped.model.opt.gravity[2] = (
                    self.original_init_params['gravity'][2] * gravity_factor)

        def reset(self, **kwargs):
            self.episode_rewards = 0
            return self.env.reset(**kwargs)

        def step(self, action):
            # Apply action
            obs, reward, terminated, truncated, info = self.env.step(action)

            # Track rewards
            self.episode_rewards += reward

            # Add reward shaping based on curriculum stage
            stage_name = self.curriculum_stages[self.current_stage][0]
            shaped_reward = reward

            # Apply reward shaping based on stage
            if stage_name == "basic_balance":
                # Reward staying upright and penalize excessive motion
                upright_reward = 1.0 if obs[2] > 0.2 else 0.0  # Height of torso
                velocity_penalty = -0.01 * np.sum(np.square(obs[8:14]))  # Control velocity
                shaped_reward += upright_reward + velocity_penalty
```

```
            elif stage_name == "limb_coordination":
                # Reward smooth movement
                forward_velocity = obs[13]  # Forward velocity component
                velocity_reward = 0.1 * forward_velocity if forward_velocity > 0 else 0
                shaped_reward += velocity_reward

            elif stage_name == "forward_movement":
                # Extra reward for moving forward
                forward_velocity = obs[13]
                forward_reward = 0.2 * forward_velocity if forward_velocity > 0 else 0
                shaped_reward += forward_reward

            # Apply perturbations if in later stages
            if stage_name in ["robust_movement", "full_task"]:
                if np.random.random() < 0.05 * self.difficulty:  # Lower chance of perturbation
                    self._apply_perturbation()

            # Update curriculum if episode done
            if terminated or truncated:
                self._update_curriculum()

            return obs, shaped_reward, terminated, truncated, info

        def _apply_perturbation(self):
            """Apply controlled perturbation to torso"""
            try:
                # Scale perturbation force by stage and difficulty
                stage_idx = self.current_stage
                force_scale = 5 + 10 * (stage_idx / (len(self.curriculum_stages) - 1))
                force = np.random.uniform(-1, 1, 3) * force_scale * self.difficulty
                self.env.unwrapped.data.xfrc_applied[1, :3] = force
            except Exception as e:
                print(f"Could not apply perturbation: {e}")

        def _update_curriculum(self):
            """Update curriculum stage and difficulty based on performance"""
            # Update tracking
            self.recent_rewards.append(self.episode_rewards)
            if len(self.recent_rewards) > self.window_size:
                self.recent_rewards.pop(0)

            self.episode_count += 1
            self.difficulty_history.append(self.difficulty)
            self.stage_history.append(self.current_stage)

            # Only update every window_size/5 episodes for more frequent feedback
            if self.episode_count % (self.window_size // 5) == 0 and len(self.recent_rewards) >= min(10, self.window_size):
                # Calculate normalized performance (0-1)
                avg_reward = np.mean(self.recent_rewards[-min(10, len(self.recent_rewards)):])
                max_expected = 1000  # More conservative estimate for Ant
                normalized_perf = np.clip((avg_reward + 500) / (max_expected + 500), 0, 1)

                # Get current stage parameters
                stage_name, target_diff = self.curriculum_stages[self.current_stage]
```

```
            # More conservative stage advancement
            if (self.current_stage < len(self.curriculum_stages) - 1 and
                self.difficulty >= target_diff * 0.95 and
                normalized_perf > self.success_threshold and
                len(self.recent_rewards) >= self.window_size // 2):

                self.current_stage += 1
                new_stage = self.curriculum_stages[self.current_stage][0]
                print(f"\nAdvancing to stage: {new_stage} (Performance: {avg_reward:.1f}, Normalized: {normalized_perf:.2f})")

                # Keep some difficulty progress when advancing stages (smoother transition)
                stage_diff_gap = (self.curriculum_stages[self.current_stage][1] -
                                  (self.curriculum_stages[self.current_stage-1][1] if self.current_stage > 0 else 0))
                self.difficulty = max(self.curriculum_stages[self.current_stage-1][1] if self.current_stage > 0 else 0,
                                      self.curriculum_stages[self.current_stage][1] * 0.3)

                self._apply_curriculum_modifications()
                return  # Skip difficulty adjustment in stage transition

            # Update difficulty within stage with momentum and hysteresis
            if normalized_perf > self.success_threshold:
                # Increase difficulty, faster if performance is much better than threshold
                increase_rate = self.adaptation_speed * (1 + (normalized_perf - self.success_threshold) * 2)
                new_diff = min(self.difficulty + increase_rate, target_diff)
                if new_diff > self.difficulty + 0.01:  # Only report significant changes
                    print(f"Increasing difficulty: {self.difficulty:.2f}→{new_diff:.2f} (Performance: {avg_reward:.1f}, Normalized: {normalized_perf:.2f})")
                    self.difficulty = new_diff
            else:
                # Decrease difficulty, faster if performance is much worse than threshold
                decrease_rate = self.adaptation_speed * 0.5 * (1 + (self.success_threshold - normalized_perf) * 2)
                new_diff = max(self.difficulty - decrease_rate,
                               self.curriculum_stages[self.current_stage-1][1] if self.current_stage > 0 else 0)
                if new_diff < self.difficulty - 0.01:  # Only report significant changes
                    print(f"Decreasing difficulty: {self.difficulty:.2f}→{new_diff:.2f} (Performance: {avg_reward:.1f}, Normalized: {normalized_perf:.2f})")
                    self.difficulty = new_diff

            # Apply current difficulty
            self._scale_difficulty(target_diff)

# 7. Training and Evaluation Functions
def make_ant_env(env_id='Ant-v5', curriculum=False, seed=0):
    """Create environment factory"""
    def _init():
        env = gym.make(env_id)
        env.reset(seed=seed)
        if curriculum:
            env = ProgressiveAntCurriculum(env)
        return env
    return _init

def train_ant_model(total_timesteps=300000, curriculum=True):  # Increased timesteps
    """Train model with enhanced curriculum"""
    # Create environments
    env_fn = make_ant_env(curriculum=curriculum)
    env = DummyVecEnv([env_fn])
    env = VecNormalize(env, norm_obs=True, norm_reward=True, clip_reward=10.0)  # Added reward clipping
```

```
    eval_env_fn = make_ant_env(curriculum=curriculum)
    eval_env = DummyVecEnv([eval_env_fn])

    # Create model with improved hyperparameters
    model = PPO(
        "MlpPolicy",
        env,
        verbose=0,
        learning_rate=linear_schedule(3e-4),  # Learning rate decay
        n_steps=2048,
        batch_size=64,
        n_epochs=10,
        gamma=0.99,
        gae_lambda=0.95,
        clip_range=0.2,
        ent_coef=0.01,
        max_grad_norm=0.5,  # Gradient clipping
        device='cpu'
    )

    # Callback for tracking
    callback = CurriculumTrackingCallback(
        eval_env,
        eval_freq=10000,
        n_eval_episodes=10  # More evaluation episodes for stability
    )

    # Train
    print(f"Training {'with curriculum' if curriculum else 'standard'} for {total_timesteps} steps...")
    start_time = time.time()
    model.learn(total_timesteps=total_timesteps, callback=callback)
    print(f"Training completed in {(time.time()-start_time)/60:.1f} minutes")

    # Save
    model.save(f"./models/ant_{'curriculum' if curriculum else 'standard'}")
    env.save(f"./vec_normalize/ant_{'curriculum' if curriculum else 'standard'}.pkl")

    return model, callback

class CurriculumTrackingCallback(BaseCallback):
    """Enhanced tracking callback with curriculum visualization"""
    def __init__(self, eval_env, eval_freq=10000, n_eval_episodes=5):
        super().__init__()
        self.eval_env = eval_env
        self.eval_freq = eval_freq
        self.n_eval_episodes = n_eval_episodes
        self.rewards = []
        self.timesteps = []
        self.difficulties = []
        self.stages = []

    def _on_step(self):
        if self.n_calls % self.eval_freq == 0:
            # Evaluate
            mean_reward, _ = evaluate_policy(
```

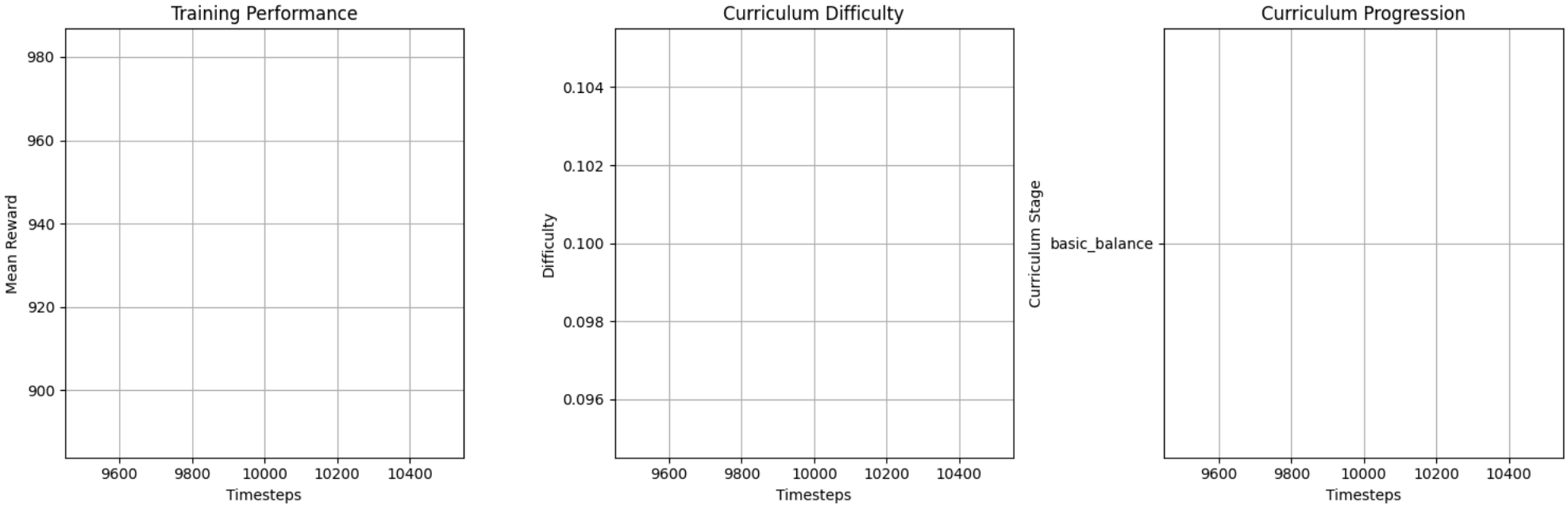
```
                self.model, self.eval_env, n_eval_episodes=self.n_eval_episodes)

            # Track metrics
            self.rewards.append(mean_reward)
            self.timesteps.append(self.n_calls)

            # Get curriculum info if available
            if hasattr(self.eval_env, 'envs'):
                env = self.eval_env.envs[0]
                if hasattr(env, 'difficulty_history') and env.difficulty_history:
                    self.difficulties.append(env.difficulty_history[-1])
                if hasattr(env, 'current_stage') and hasattr(env, 'curriculum_stages'):
                    self.stages.append(env.curriculum_stages[env.current_stage][0])

            # Plot
            self._plot_progress()

        return True

    def _plot_progress(self):
        """Plot training progress with curriculum info"""
        plt.figure(figsize=(15, 5))

        # Reward plot
        plt.subplot(1, 3, 1)
        plt.plot(self.timesteps, self.rewards)
        plt.xlabel('Timesteps')
        plt.ylabel('Mean Reward')
        plt.title('Training Performance')
        plt.grid(True)

        # Difficulty plot
        if self.difficulties:
            plt.subplot(1, 3, 2)
            plt.plot(self.timesteps[:len(self.difficulties)], self.difficulties)
            plt.xlabel('Timesteps')
            plt.ylabel('Difficulty')
            plt.title('Curriculum Difficulty')
            plt.grid(True)

        # Stage plot
        if self.stages:
            plt.subplot(1, 3, 3)
            unique_stages = list(dict.fromkeys(self.stages))  # Preserve order
            stage_nums = [unique_stages.index(s) for s in self.stages]
            plt.plot(self.timesteps[:len(stage_nums)], stage_nums)
            plt.yticks(range(len(unique_stages)), unique_stages)
            plt.xlabel('Timesteps')
            plt.ylabel('Curriculum Stage')
            plt.title('Curriculum Progression')
            plt.grid(True)

        plt.tight_layout()
        display(plt.gcf())
        plt.close()
```

```
    def compare_models(curriculum_model, standard_model):
        """Compare curriculum vs standard models with more meaningful metrics"""
        difficulties = [0.0, 0.25, 0.5, 0.75, 1.0]
        n_trials = 10  # More trials for better statistics
        curr_rewards = {diff: [] for diff in difficulties}
        std_rewards = {diff: [] for diff in difficulties}

        print("Evaluating models at different difficulties...")

        for diff in difficulties:
            for _ in range(n_trials):
                # Curriculum model
                env = gym.make('Ant-v5')
                env = ProgressiveAntCurriculum(env)
                env.difficulty = diff
                mean_reward, _ = evaluate_policy(curriculum_model, env, n_eval_episodes=3)
                curr_rewards[diff].append(mean_reward)
                env.close()

                # Standard model
                env = gym.make('Ant-v5')
                env = ProgressiveAntCurriculum(env)
                env.difficulty = diff
                mean_reward, _ = evaluate_policy(standard_model, env, n_eval_episodes=3)
                std_rewards[diff].append(mean_reward)
                env.close()

            curr_mean = np.mean(curr_rewards[diff])
            std_mean = np.mean(std_rewards[diff])
            curr_std = np.std(curr_rewards[diff])
            std_std = np.std(std_rewards[diff])

            print(f"Difficulty {diff:.2f}: Curriculum={curr_mean:.1f}±{curr_std:.1f}, Standard={std_mean:.1f}±{std_std:.1f}")

        # Plot comparison
        plt.figure(figsize=(10, 6))

        # Calculate means and standard deviations
        curr_means = [np.mean(curr_rewards[d]) for d in difficulties]
        std_means = [np.mean(std_rewards[d]) for d in difficulties]
        curr_stds = [np.std(curr_rewards[d]) for d in difficulties]
        std_stds = [np.std(std_rewards[d]) for d in difficulties]

        plt.errorbar(difficulties, curr_means, yerr=curr_stds, fmt='o-', capsize=5, label='Curriculum')
        plt.errorbar(difficulties, std_means, yerr=std_stds, fmt='s-', capsize=5, label='Standard')
        plt.xlabel('Difficulty Level')
        plt.ylabel('Mean Reward')
        plt.title('Performance Across Difficulties')
        plt.legend()
        plt.grid(True)
        plt.show()

        # Calculate metrics
        curr_avg = np.mean(curr_means)
        std_avg = np.mean(std_means)
        improvement = (curr_avg - std_avg) / abs(max(std_avg, 1)) * 100
```

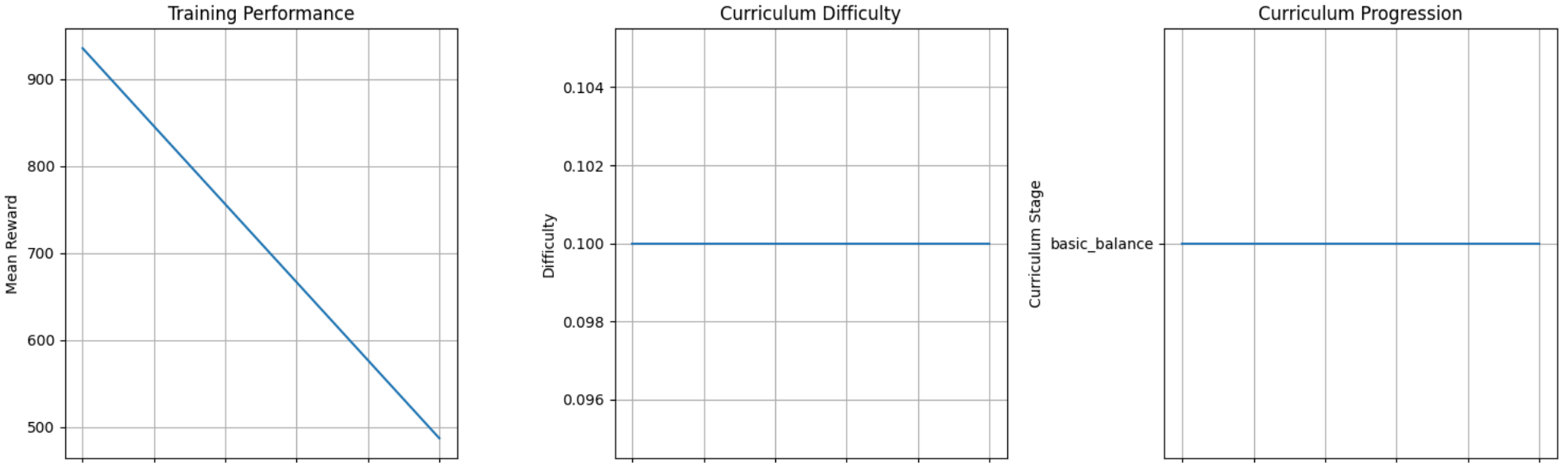
```
        print(f"\nResults:")
        print(f"- Curriculum average reward: {curr_avg:.1f}")
        print(f"- Standard average reward: {std_avg:.1f}")
        print(f"- Overall improvement: {improvement:.1f}%")

        # Robustness (performance retention in difficult scenarios)
        easy_perf_curr = curr_means[0]
        hard_perf_curr = curr_means[-1]
        curr_retention = (hard_perf_curr / max(easy_perf_curr, 1)) * 100

        easy_perf_std = std_means[0]
        hard_perf_std = std_means[-1]
        std_retention = (hard_perf_std / max(easy_perf_std, 1)) * 100

        print(f"\nRobustness:")
        print(f"- Curriculum performance retention: {curr_retention:.1f}%")
        print(f"- Standard performance retention: {std_retention:.1f}%")
        print(f"- Retention difference: {curr_retention - std_retention:.1f}%")

    # 8. Visualization
    def visualize_ant(model, difficulty=0.5, steps=300):
        """Render ant behavior as video"""
        env = gym.make('Ant-v5', render_mode='rgb_array')
        env = ProgressiveAntCurriculum(env)
        env.difficulty = difficulty

        frames = []
        obs, _ = env.reset()
        for _ in range(steps):
            action, _ = model.predict(obs, deterministic=True)
            obs, _, terminated, truncated, _ = env.step(action)
            frames.append(env.render())
            if terminated or truncated:
                obs, _ = env.reset()

        env.close()

        # Save as video
        height, width, _ = frames[0].shape
        video_path = '/tmp/ant_curriculum.mp4'

        import subprocess
        cmd = [
            'ffmpeg', '-y', '-f', 'rawvideo',
            '-vcodec', 'rawvideo', '-s', f'{width}x{height}',
            '-pix_fmt', 'rgb24', '-r', '25', '-i', '-',
            '-c:v', 'libx264', '-pix_fmt', 'yuv420p',
            video_path
        ]
        process = subprocess.Popen(cmd, stdin=subprocess.PIPE)
        for frame in frames:
            process.stdin.write(frame.tobytes())
        process.stdin.close()
        process.wait()
```

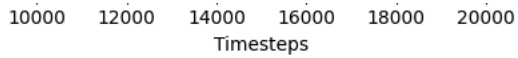
```
        # Display
        mp4 = open(video_path,'rb').read()
        data_url = "data:video/mp4;base64," + b64encode(mp4).decode()
        return HTML(f"""
        <video width=600 controls>
            <source src="{data_url}" type="video/mp4">
        </video>
        """)

    # 9. Main Experiment
    def run_experiment(fast_mode=False):
        """Run full curriculum learning experiment

        Args:
            fast_mode: If True, use reduced timesteps and evaluations for faster testing
        """
        timesteps = 100000 if fast_mode else 300000

        # Train curriculum model
        print("=== TRAINING CURRICULUM MODEL ===")
        curriculum_model, _ = train_ant_model(total_timesteps=timesteps, curriculum=True)

        # Train standard model
        print("\n=== TRAINING STANDARD MODEL ===")
        standard_model, _ = train_ant_model(total_timesteps=timesteps, curriculum=False)

        # Compare models
        print("\n=== COMPARING MODELS ===")
        compare_models(curriculum_model, standard_model)

        # Visualize
        print("\n=== VISUALIZING CURRICULUM MODEL ===")
        display(visualize_ant(curriculum_model, difficulty=0.5))

        print("\n=== VISUALIZING STANDARD MODEL ===")
        display(visualize_ant(standard_model, difficulty=0.5))

    # Run the experiment
    if __name__ == "__main__":
        # Set fast_mode=True for quicker testing (reduces training time)
        # Set fast_mode=False for full experiment with better results
        run_experiment(fast_mode=True)  # Change to False for complete training
```

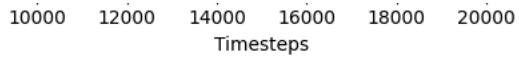
```
=== TRAINING CURRICULUM MODEL ===
Initialized ProgressiveAntCurriculum with stage: basic_balance (difficulty: 0.10)
Initialized ProgressiveAntCurriculum with stage: basic_balance (difficulty: 0.10)
Training with curriculum for 100000 steps...
/usr/local/lib/python3.11/dist-packages/stable_baselines3/common/evaluation.py:67: UserWarning: Evaluation environment is not wrapped with a ``Monitor`` wrapper. This may result in reporting modif
  warnings.warn(
```

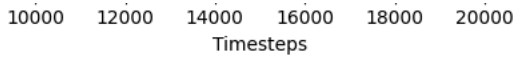



```
/usr/local/lib/python3.11/dist-packages/stable_baselines3/common/evaluation.py:67: UserWarning: Evaluation environment is not wrapped with a ``Monitor`` wrapper. This may result in reporting modif
  warnings.warn(
```

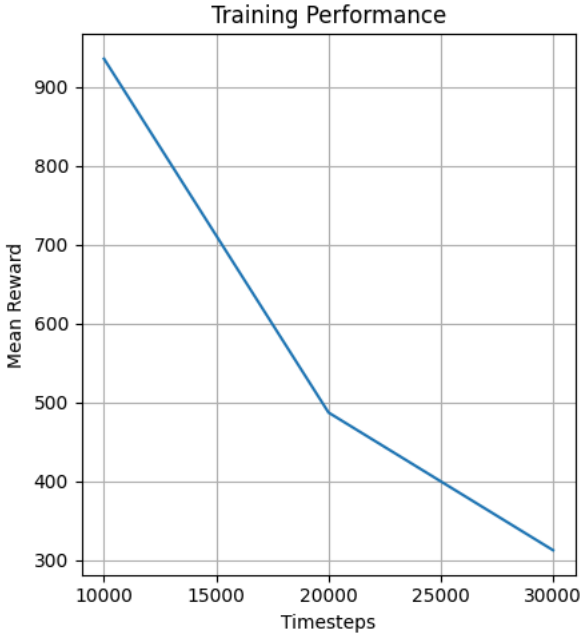


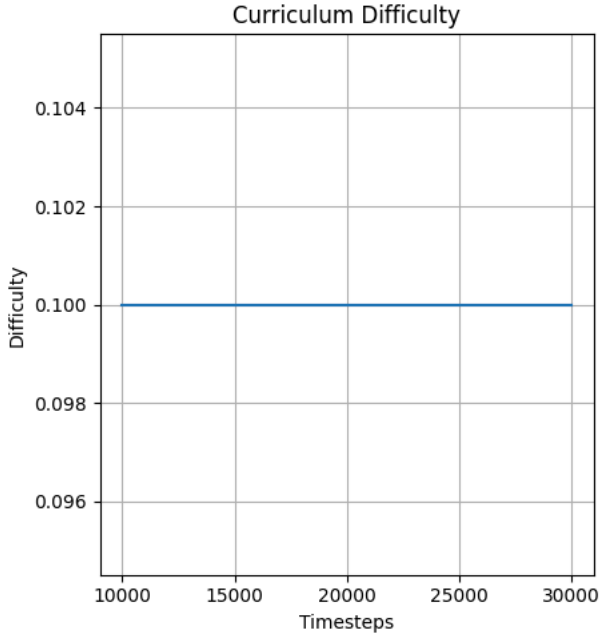


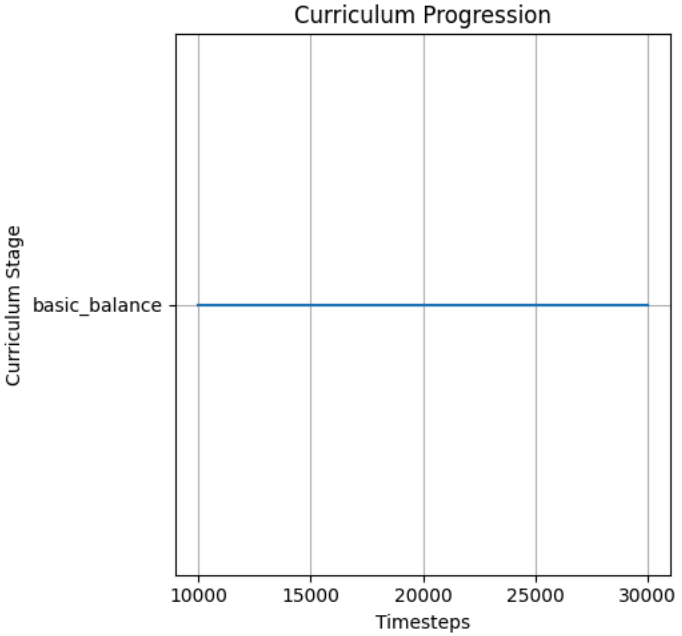




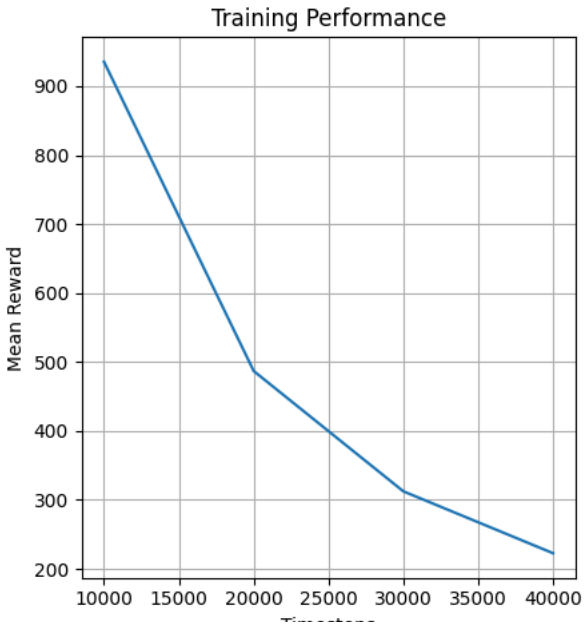

```
/usr/local/lib/python3.11/dist-packages/stable_baselines3/common/evaluation.py:67: UserWarning: Evaluation environment is not wrapped with a ``Monitor`` wrapper. This may result in reporting modif
  warnings.warn(
```

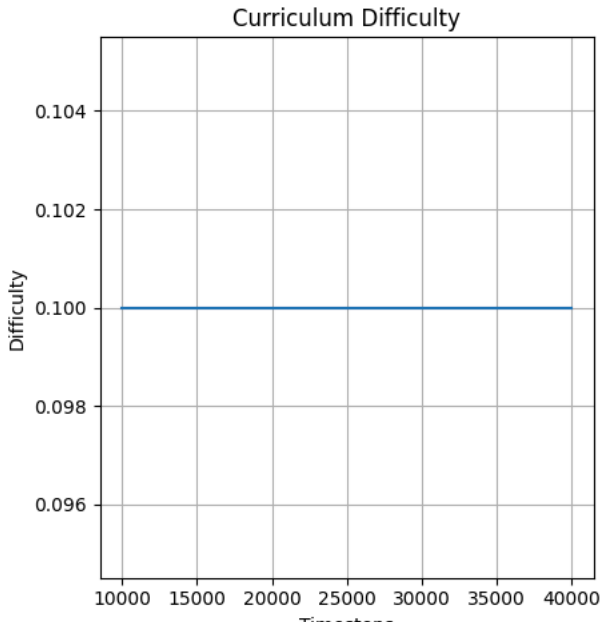


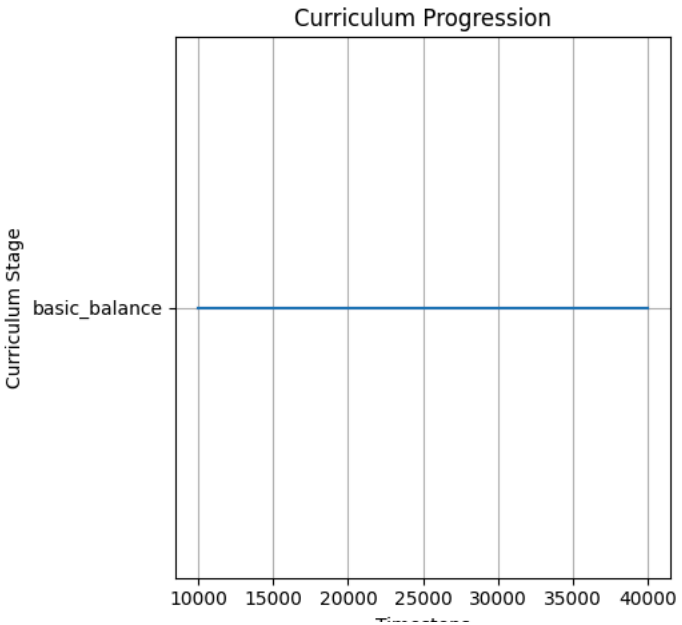





```
/usr/local/lib/python3.11/dist-packages/stable_baselines3/common/evaluation.py:67: UserWarning: Evaluation environment is not wrapped with a ``Monitor`` wrapper. This may result in reporting modif
  warnings.warn(
```

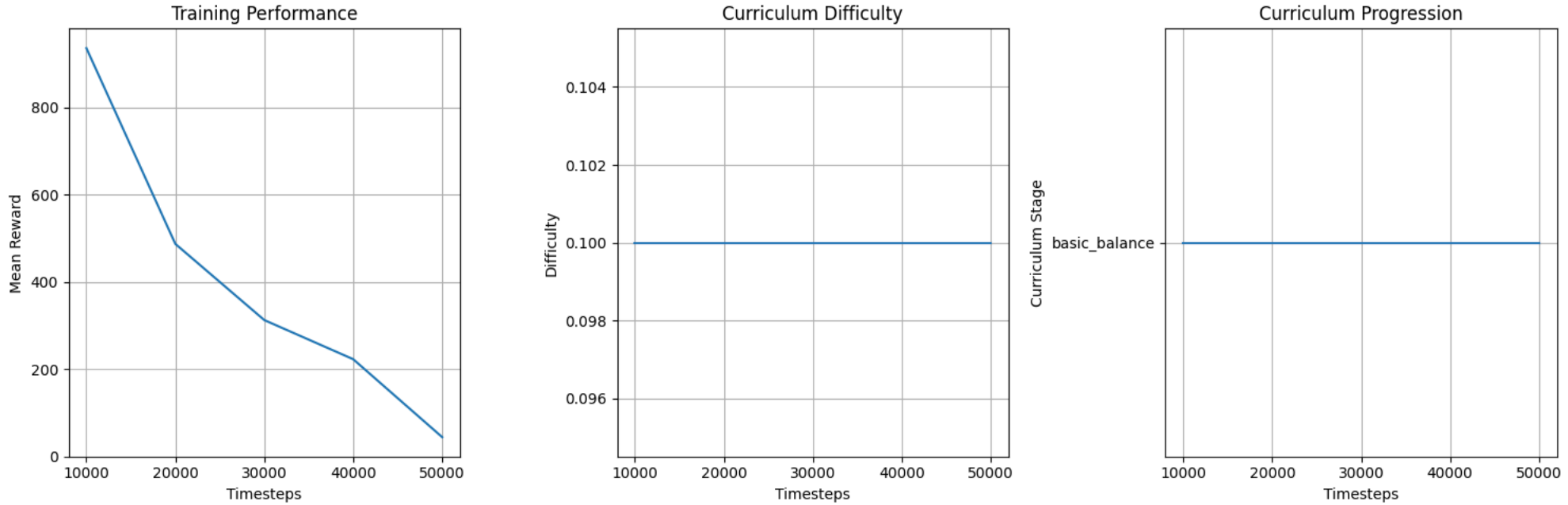




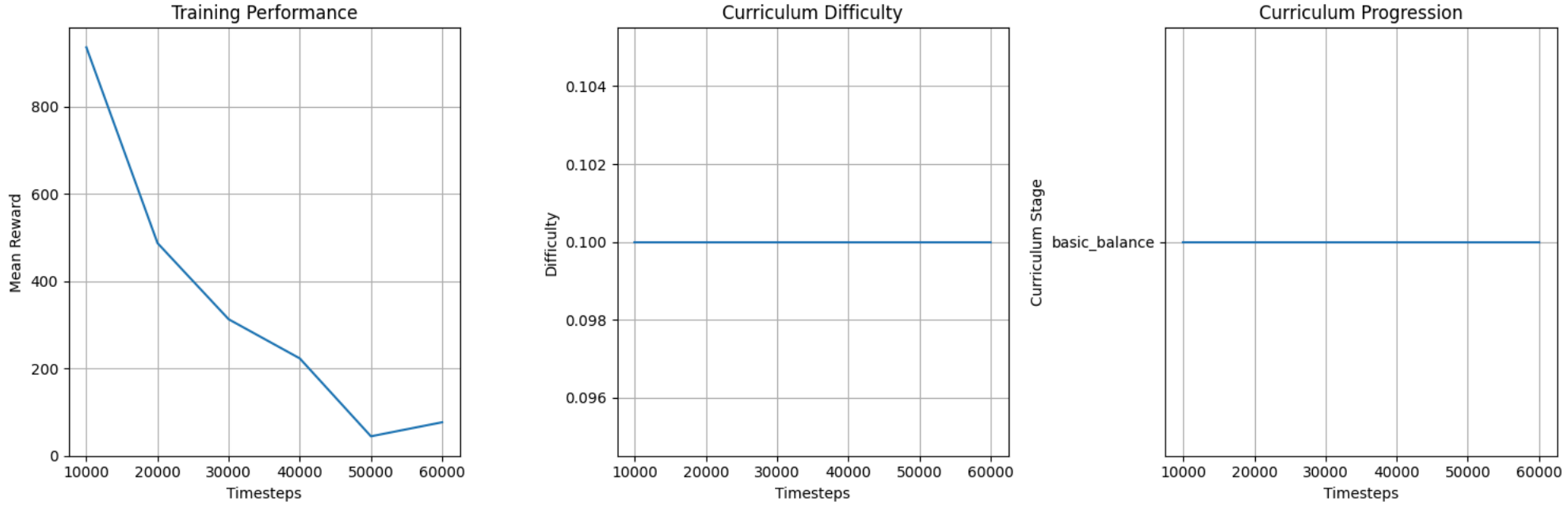


Timesteps Timesteps Timesteps

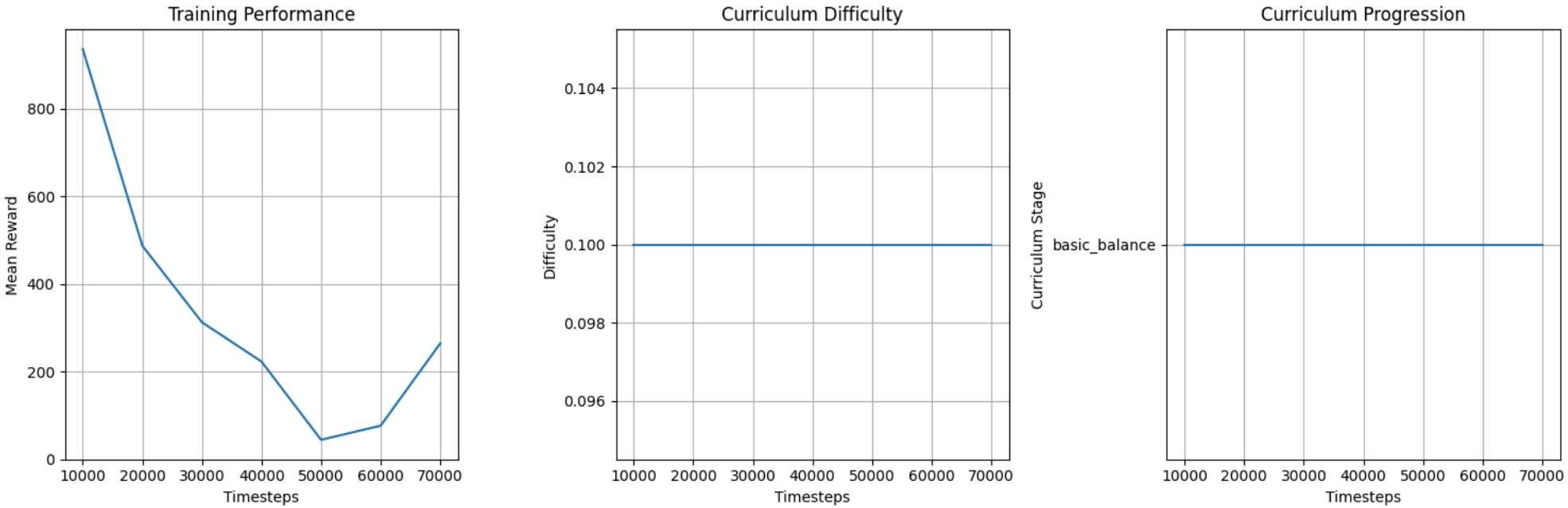
```
/usr/local/lib/python3.11/dist-packages/stable_baselines3/common/evaluation.py:67: UserWarning: Evaluation environment is not wrapped with a ``Monitor`` wrapper. This may result in reporting modif
  warnings.warn(
```



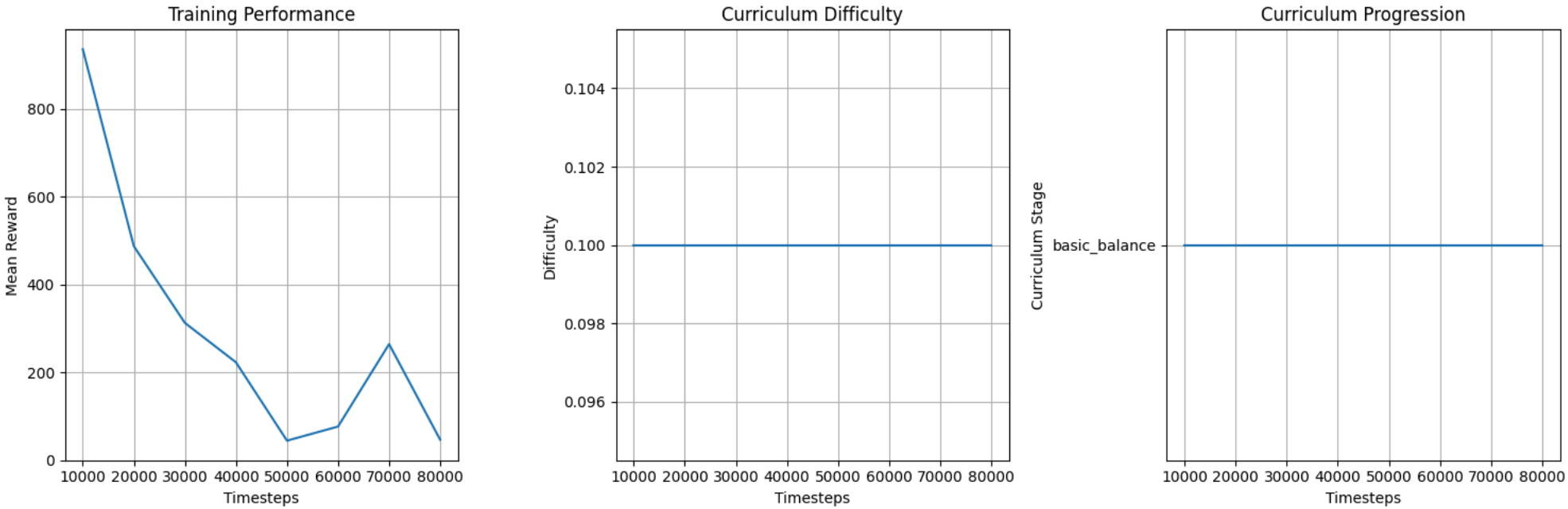

```
/usr/local/lib/python3.11/dist-packages/stable_baselines3/common/evaluation.py:67: UserWarning: Evaluation environment is not wrapped with a ``Monitor`` wrapper. This may result in reporting modif
  warnings.warn(
```



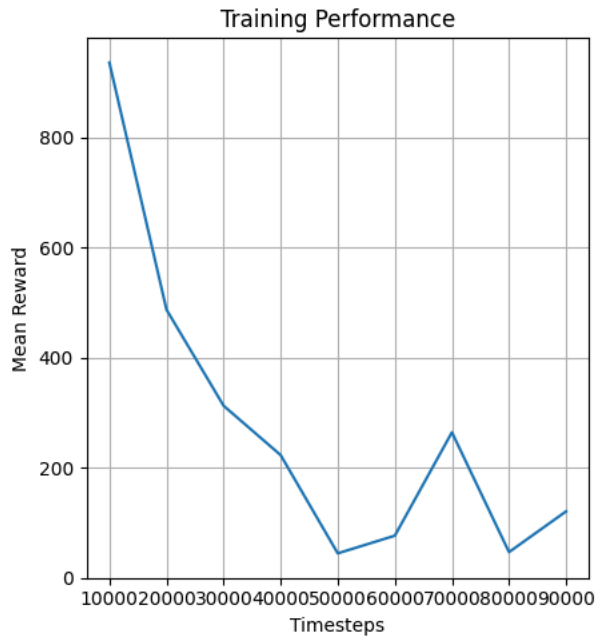
/usr/local/lib/python3.11/dist-packages/stable_baselines3/common/evaluation.py:67: UserWarning: Evaluation environment is not wrapped with a ``Monitor`` wrapper. This may result in reporting modif
  warnings.warn(

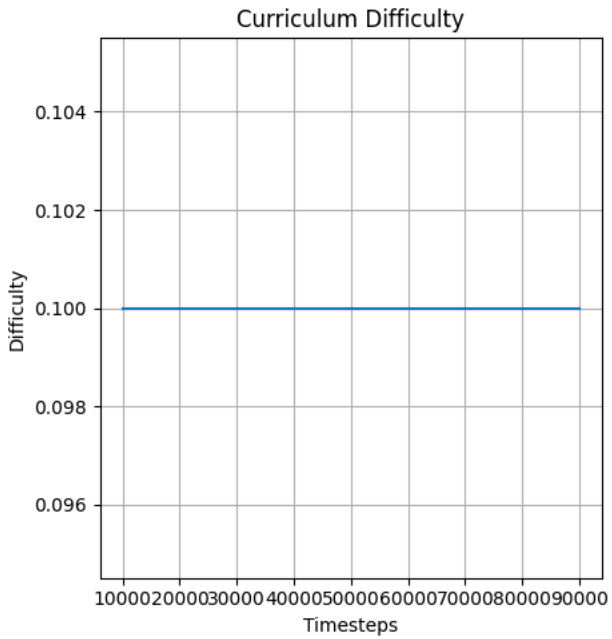


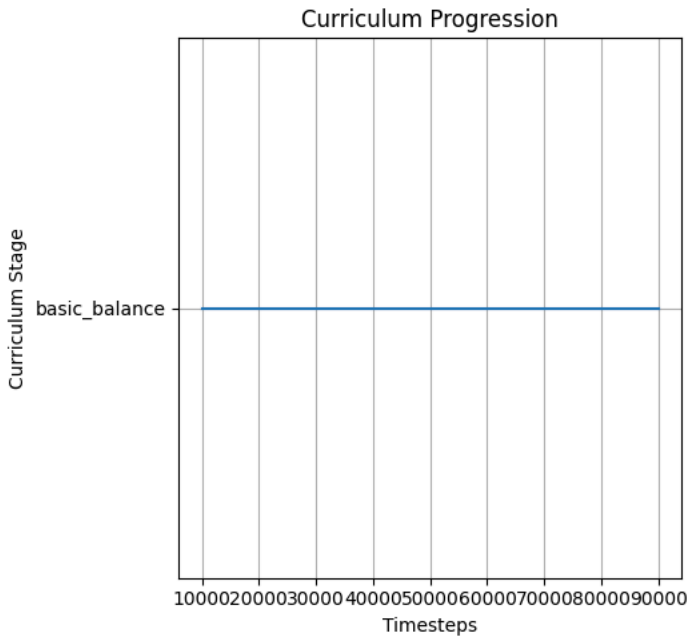
/usr/local/lib/python3.11/dist-packages/stable_baselines3/common/evaluation.py:67: UserWarning: Evaluation environment is not wrapped with a ``Monitor`` wrapper. This may result in reporting modif
  warnings.warn(



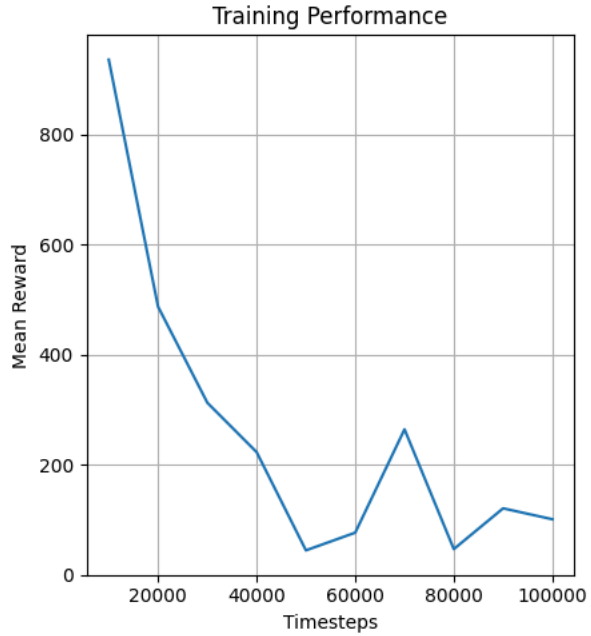
/usr/local/lib/python3.11/dist-packages/stable_baselines3/common/evaluation.py:67: UserWarning: Evaluation environment is not wrapped with a ``Monitor`` wrapper. This may result in reporting modif
  warnings.warn(

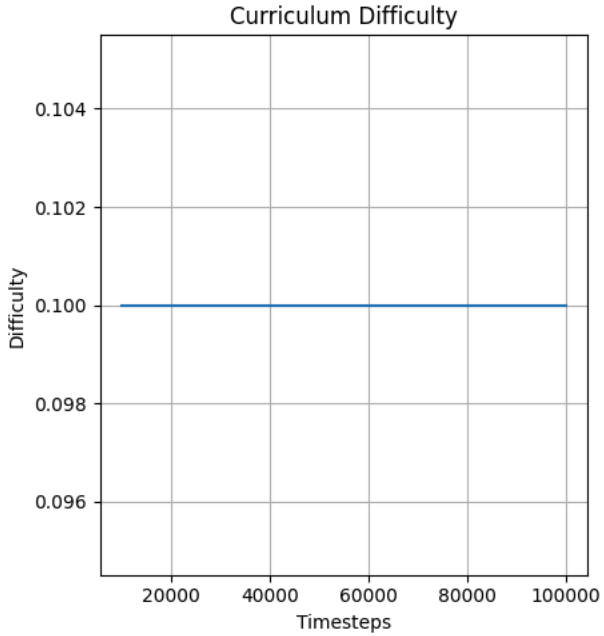


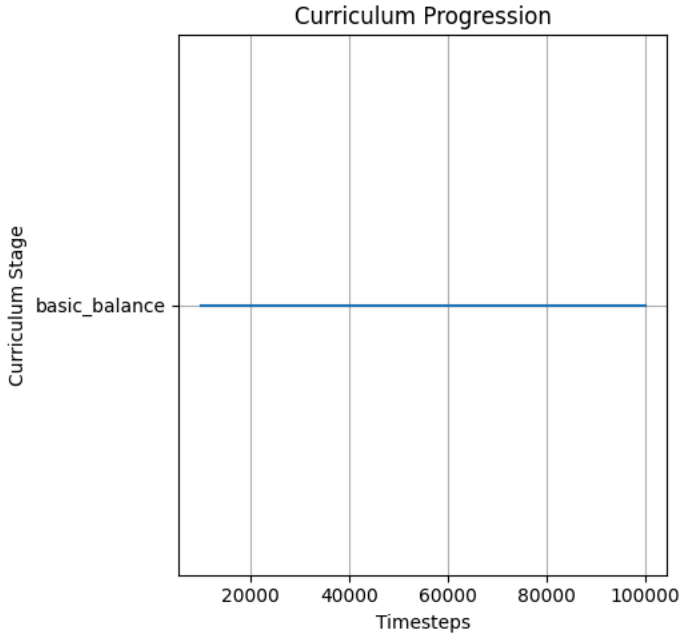





```
/usr/local/lib/python3.11/dist-packages/stable_baselines3/common/evaluation.py:67: UserWarning: Evaluation environment is not wrapped with a ``Monitor`` wrapper. This may result in reporting modi
  warnings.warn(
```

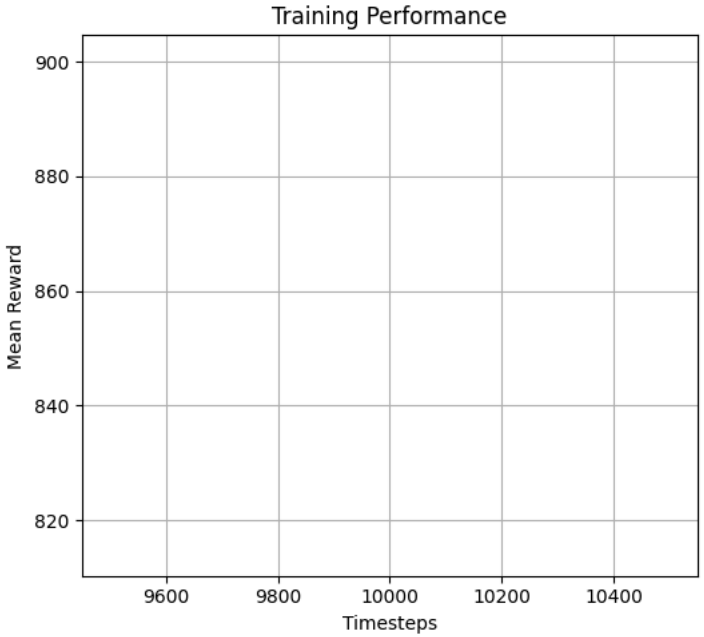


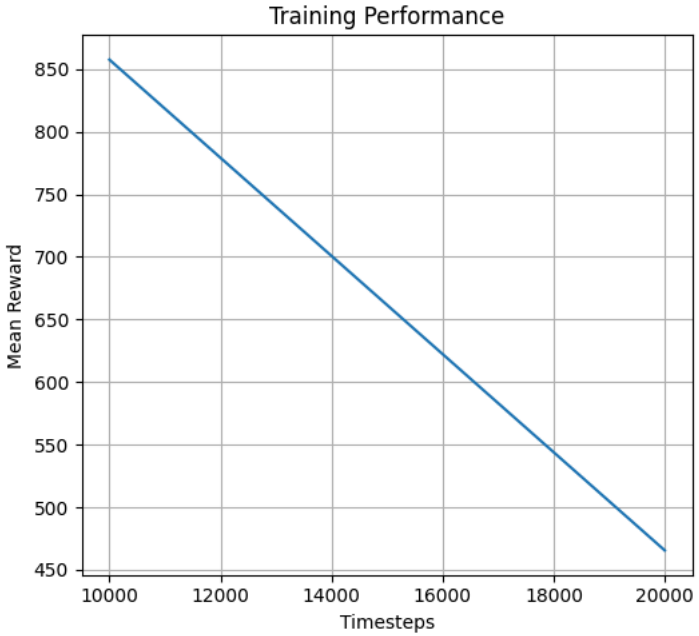


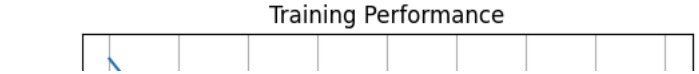


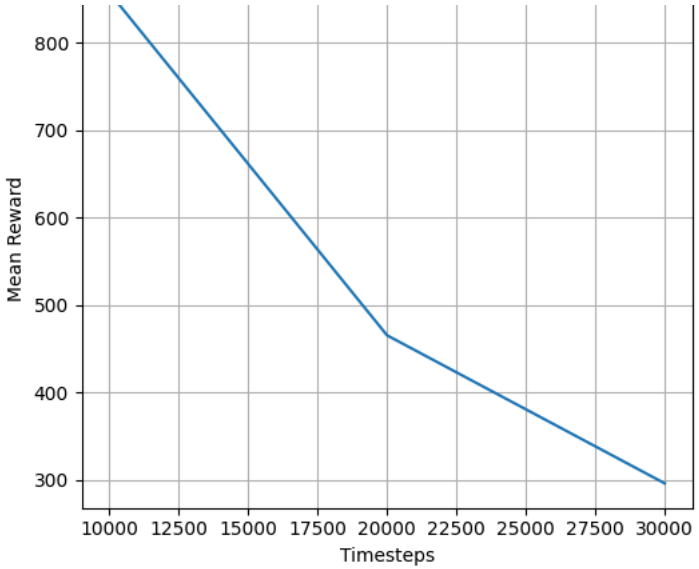

```
Training completed in 3.7 minutes

=== TRAINING STANDARD MODEL ===
Training standard for 100000 steps...
```

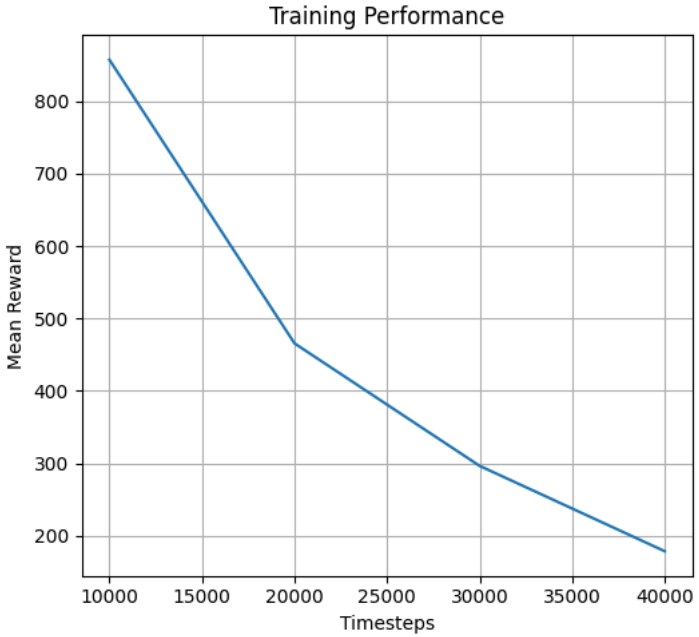
```
/usr/local/lib/python3.11/dist-packages/stable_baselines3/common/evaluation.py:67: UserWarning: Evaluation environment is not wrapped with a ``Monitor`` wrapper. This may result in reporting modif
  warnings.warn(
```



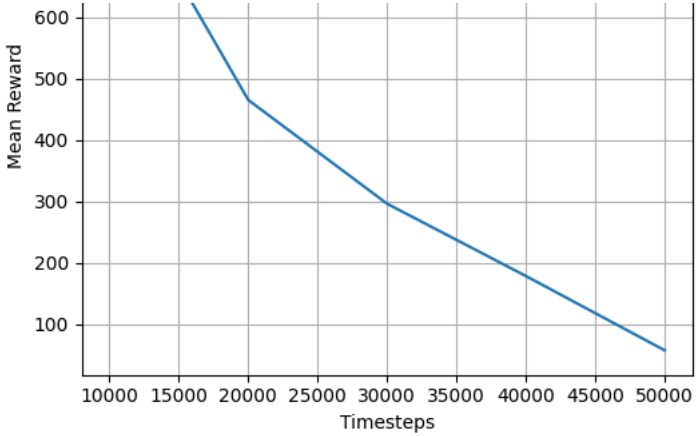


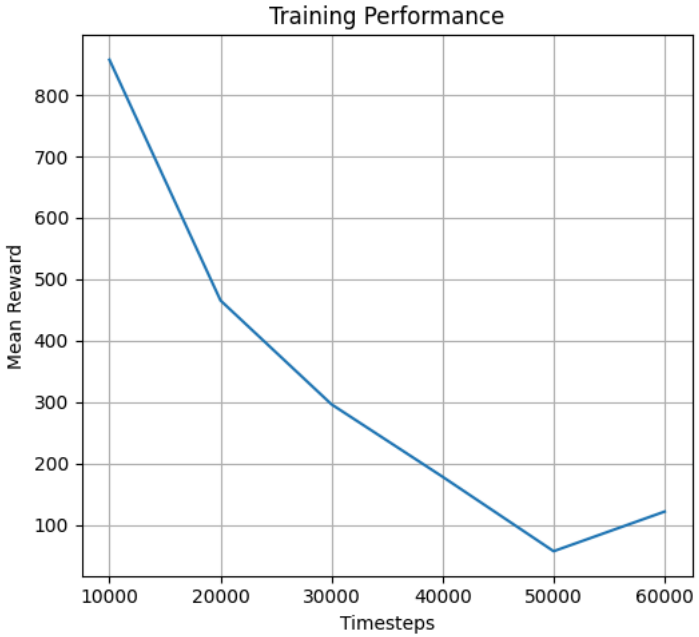



800
700
600
500
400
300
Mean Reward
10000 12500 15000 17500 20000 22500 25000 27500 30000
Timesteps

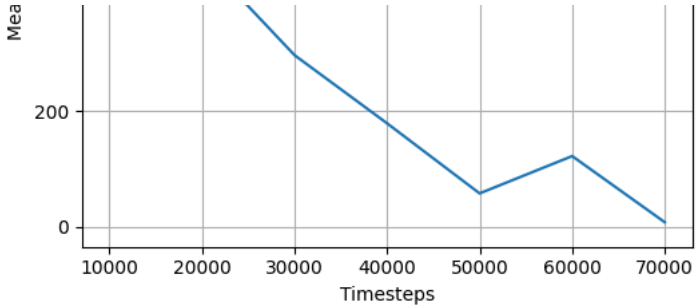

Training Performance
800
700
600
500
400
300
200
Mean Reward
10000 15000 20000 25000 30000 35000 40000
Timesteps

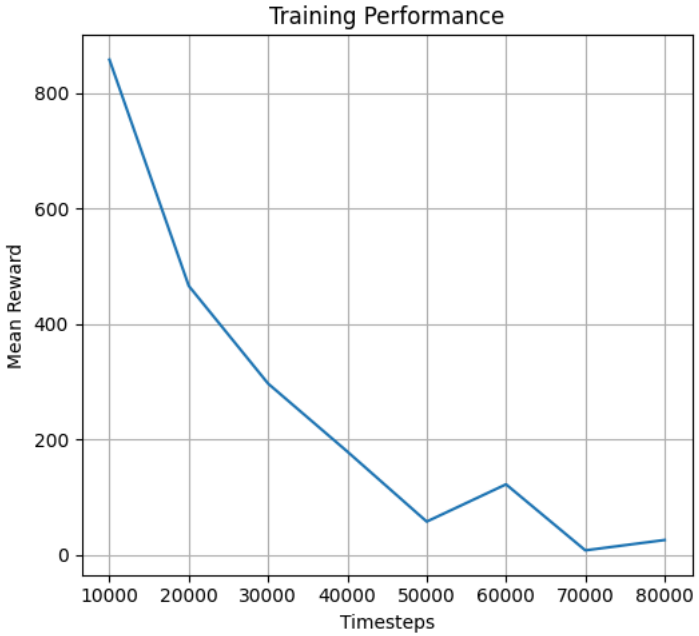

Training Performance
800
700


Mean Reward
600
500
400
300
200
100
10000 15000 20000 25000 30000 35000 40000 45000 50000
Timesteps

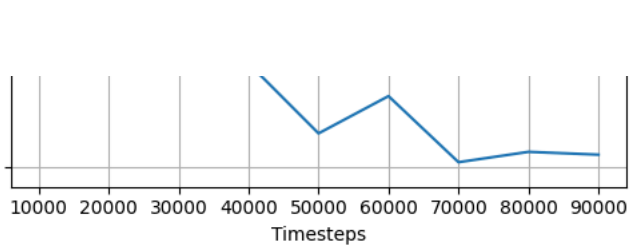

Training Performance
Mean Reward
800
700
600
500
400
300
200
100
10000 20000 30000 40000 50000 60000
Timesteps

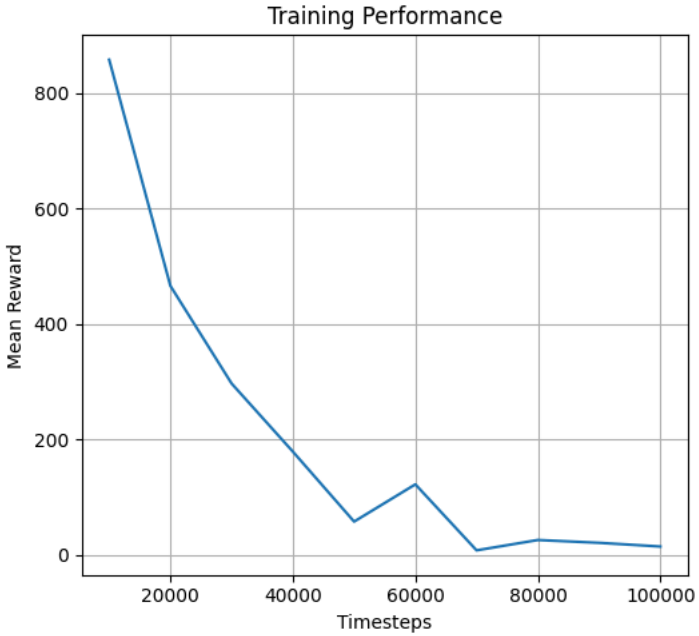

Training Performance
800
600
400


Mea
200
0
10000
20000
30000
40000
50000
60000
70000
Timesteps


Training Performance
800
600
400
200
0
Mean Reward
10000
20000
30000
40000
50000
60000
70000
80000
Timesteps


Training Performance
800
600
400
200
Mean Reward

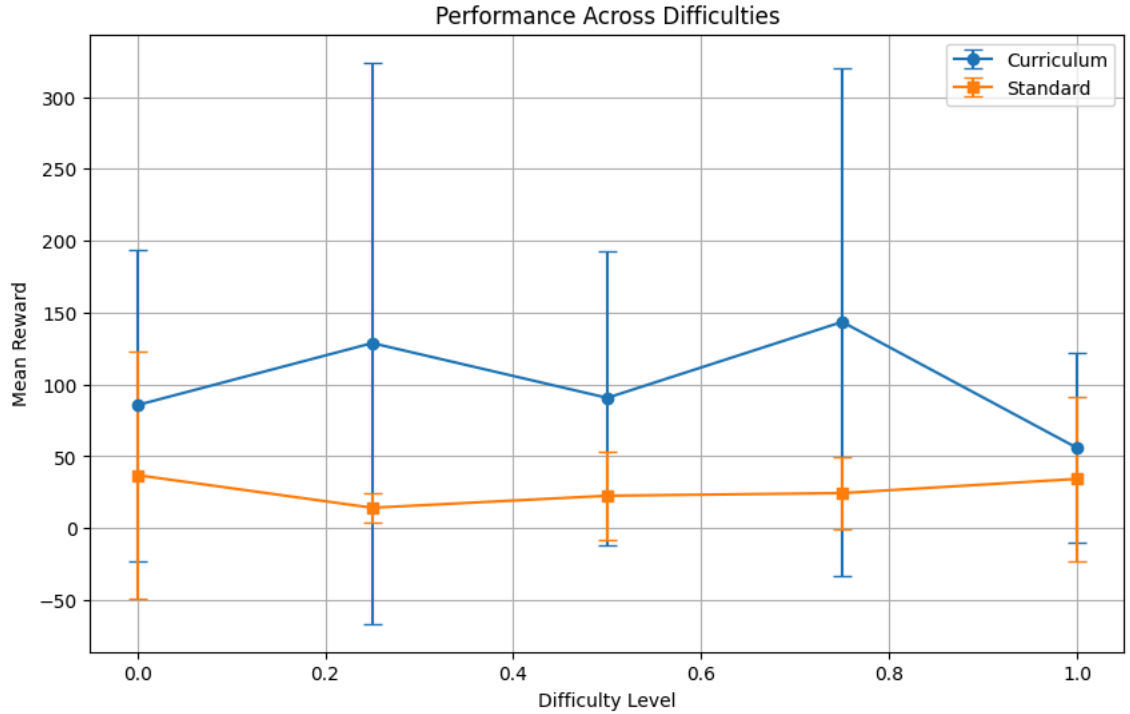





```
Training completed in 3.6 minutes

=== COMPARING MODELS ===
Evaluating models at different difficulties...
Initialized ProgressiveAntCurriculum with stage: basic_balance (difficulty: 0.10)
Initialized ProgressiveAntCurriculum with stage: basic_balance (difficulty: 0.10)
Initialized ProgressiveAntCurriculum with stage: basic_balance (difficulty: 0.10)
Initialized ProgressiveAntCurriculum with stage: basic_balance (difficulty: 0.10)
Initialized ProgressiveAntCurriculum with stage: basic_balance (difficulty: 0.10)
Initialized ProgressiveAntCurriculum with stage: basic_balance (difficulty: 0.10)
Initialized ProgressiveAntCurriculum with stage: basic_balance (difficulty: 0.10)
Initialized ProgressiveAntCurriculum with stage: basic_balance (difficulty: 0.10)
Initialized ProgressiveAntCurriculum with stage: basic_balance (difficulty: 0.10)
Initialized ProgressiveAntCurriculum with stage: basic_balance (difficulty: 0.10)
Initialized ProgressiveAntCurriculum with stage: basic_balance (difficulty: 0.10)
Initialized ProgressiveAntCurriculum with stage: basic_balance (difficulty: 0.10)
Initialized ProgressiveAntCurriculum with stage: basic_balance (difficulty: 0.10)
Initialized ProgressiveAntCurriculum with stage: basic_balance (difficulty: 0.10)
Initialized ProgressiveAntCurriculum with stage: basic_balance (difficulty: 0.10)
Initialized ProgressiveAntCurriculum with stage: basic_balance (difficulty: 0.10)
Initialized ProgressiveAntCurriculum with stage: basic_balance (difficulty: 0.10)
Initialized ProgressiveAntCurriculum with stage: basic_balance (difficulty: 0.10)
Initialized ProgressiveAntCurriculum with stage: basic_balance (difficulty: 0.10)
Initialized ProgressiveAntCurriculum with stage: basic_balance (difficulty: 0.10)
Initialized ProgressiveAntCurriculum with stage: basic_balance (difficulty: 0.10)
Initialized ProgressiveAntCurriculum with stage: basic_balance (difficulty: 0.10)
Initialized ProgressiveAntCurriculum with stage: basic_balance (difficulty: 0.10)
Initialized ProgressiveAntCurriculum with stage: basic_balance (difficulty: 0.10)
Initialized ProgressiveAntCurriculum with stage: basic_balance (difficulty: 0.10)
Initialized ProgressiveAntCurriculum with stage: basic_balance (difficulty: 0.10)
Difficulty 0.00: Curriculum=85.6±108.2, Standard=36.9±86.0
Initialized ProgressiveAntCurriculum with stage: basic_balance (difficulty: 0.10)
```

```
Initialized ProgressiveAntCurriculum with stage: basic_balance (difficulty: 0.10)
Initialized ProgressiveAntCurriculum with stage: basic_balance (difficulty: 0.10)
Initialized ProgressiveAntCurriculum with stage: basic_balance (difficulty: 0.10)
Initialized ProgressiveAntCurriculum with stage: basic_balance (difficulty: 0.10)
Initialized ProgressiveAntCurriculum with stage: basic_balance (difficulty: 0.10)
Initialized ProgressiveAntCurriculum with stage: basic_balance (difficulty: 0.10)
Initialized ProgressiveAntCurriculum with stage: basic_balance (difficulty: 0.10)
Initialized ProgressiveAntCurriculum with stage: basic_balance (difficulty: 0.10)
Initialized ProgressiveAntCurriculum with stage: basic_balance (difficulty: 0.10)
Initialized ProgressiveAntCurriculum with stage: basic_balance (difficulty: 0.10)
Initialized ProgressiveAntCurriculum with stage: basic_balance (difficulty: 0.10)
Initialized ProgressiveAntCurriculum with stage: basic_balance (difficulty: 0.10)
Initialized ProgressiveAntCurriculum with stage: basic_balance (difficulty: 0.10)
Initialized ProgressiveAntCurriculum with stage: basic_balance (difficulty: 0.10)
Initialized ProgressiveAntCurriculum with stage: basic_balance (difficulty: 0.10)
Initialized ProgressiveAntCurriculum with stage: basic_balance (difficulty: 0.10)
Initialized ProgressiveAntCurriculum with stage: basic_balance (difficulty: 0.10)
Initialized ProgressiveAntCurriculum with stage: basic_balance (difficulty: 0.10)
Initialized ProgressiveAntCurriculum with stage: basic_balance (difficulty: 0.10)
Difficulty 0.25: Curriculum=128.8±195.2, Standard=14.2±10.0
Initialized ProgressiveAntCurriculum with stage: basic_balance (difficulty: 0.10)
Initialized ProgressiveAntCurriculum with stage: basic_balance (difficulty: 0.10)
Initialized ProgressiveAntCurriculum with stage: basic_balance (difficulty: 0.10)
Initialized ProgressiveAntCurriculum with stage: basic_balance (difficulty: 0.10)
Initialized ProgressiveAntCurriculum with stage: basic_balance (difficulty: 0.10)
Initialized ProgressiveAntCurriculum with stage: basic_balance (difficulty: 0.10)
Initialized ProgressiveAntCurriculum with stage: basic_balance (difficulty: 0.10)
Initialized ProgressiveAntCurriculum with stage: basic_balance (difficulty: 0.10)
Initialized ProgressiveAntCurriculum with stage: basic_balance (difficulty: 0.10)
Initialized ProgressiveAntCurriculum with stage: basic_balance (difficulty: 0.10)
Initialized ProgressiveAntCurriculum with stage: basic_balance (difficulty: 0.10)
Initialized ProgressiveAntCurriculum with stage: basic_balance (difficulty: 0.10)
Initialized ProgressiveAntCurriculum with stage: basic_balance (difficulty: 0.10)
Initialized ProgressiveAntCurriculum with stage: basic_balance (difficulty: 0.10)
Initialized ProgressiveAntCurriculum with stage: basic_balance (difficulty: 0.10)
Initialized ProgressiveAntCurriculum with stage: basic_balance (difficulty: 0.10)
Initialized ProgressiveAntCurriculum with stage: basic_balance (difficulty: 0.10)
Initialized ProgressiveAntCurriculum with stage: basic_balance (difficulty: 0.10)
Initialized ProgressiveAntCurriculum with stage: basic_balance (difficulty: 0.10)
Initialized ProgressiveAntCurriculum with stage: basic_balance (difficulty: 0.10)
Difficulty 0.50: Curriculum=90.7±102.2, Standard=22.6±31.1
Initialized ProgressiveAntCurriculum with stage: basic_balance (difficulty: 0.10)
Initialized ProgressiveAntCurriculum with stage: basic_balance (difficulty: 0.10)
Initialized ProgressiveAntCurriculum with stage: basic_balance (difficulty: 0.10)
Initialized ProgressiveAntCurriculum with stage: basic_balance (difficulty: 0.10)
Initialized ProgressiveAntCurriculum with stage: basic_balance (difficulty: 0.10)
Initialized ProgressiveAntCurriculum with stage: basic_balance (difficulty: 0.10)
Initialized ProgressiveAntCurriculum with stage: basic_balance (difficulty: 0.10)
Initialized ProgressiveAntCurriculum with stage: basic_balance (difficulty: 0.10)
Initialized ProgressiveAntCurriculum with stage: basic_balance (difficulty: 0.10)
Initialized ProgressiveAntCurriculum with stage: basic_balance (difficulty: 0.10)
Initialized ProgressiveAntCurriculum with stage: basic_balance (difficulty: 0.10)
Initialized ProgressiveAntCurriculum with stage: basic_balance (difficulty: 0.10)
Initialized ProgressiveAntCurriculum with stage: basic_balance (difficulty: 0.10)
Initialized ProgressiveAntCurriculum with stage: basic_balance (difficulty: 0.10)
Initialized ProgressiveAntCurriculum with stage: basic_balance (difficulty: 0.10)
Initialized ProgressiveAntCurriculum with stage: basic_balance (difficulty: 0.10)
Initialized ProgressiveAntCurriculum with stage: basic_balance (difficulty: 0.10)
Initialized ProgressiveAntCurriculum with stage: basic_balance (difficulty: 0.10)
Initialized ProgressiveAntCurriculum with stage: basic_balance (difficulty: 0.10)
Initialized ProgressiveAntCurriculum with stage: basic_balance (difficulty: 0.10)
Difficulty 0.75: Curriculum=143.5±176.5, Standard=24.4±25.4
```

```
Initialized ProgressiveAntCurriculum with stage: basic_balance (difficulty: 0.10)
Initialized ProgressiveAntCurriculum with stage: basic_balance (difficulty: 0.10)
Initialized ProgressiveAntCurriculum with stage: basic_balance (difficulty: 0.10)
Initialized ProgressiveAntCurriculum with stage: basic_balance (difficulty: 0.10)
Initialized ProgressiveAntCurriculum with stage: basic_balance (difficulty: 0.10)
Initialized ProgressiveAntCurriculum with stage: basic_balance (difficulty: 0.10)
Initialized ProgressiveAntCurriculum with stage: basic_balance (difficulty: 0.10)
Initialized ProgressiveAntCurriculum with stage: basic_balance (difficulty: 0.10)
Initialized ProgressiveAntCurriculum with stage: basic_balance (difficulty: 0.10)
Initialized ProgressiveAntCurriculum with stage: basic_balance (difficulty: 0.10)
Initialized ProgressiveAntCurriculum with stage: basic_balance (difficulty: 0.10)
Initialized ProgressiveAntCurriculum with stage: basic_balance (difficulty: 0.10)
Initialized ProgressiveAntCurriculum with stage: basic_balance (difficulty: 0.10)
Initialized ProgressiveAntCurriculum with stage: basic_balance (difficulty: 0.10)
Initialized ProgressiveAntCurriculum with stage: basic_balance (difficulty: 0.10)
Initialized ProgressiveAntCurriculum with stage: basic_balance (difficulty: 0.10)
Initialized ProgressiveAntCurriculum with stage: basic_balance (difficulty: 0.10)
Initialized ProgressiveAntCurriculum with stage: basic_balance (difficulty: 0.10)
Initialized ProgressiveAntCurriculum with stage: basic_balance (difficulty: 0.10)
Initialized ProgressiveAntCurriculum with stage: basic_balance (difficulty: 0.10)
Initialized ProgressiveAntCurriculum with stage: basic_balance (difficulty: 0.10)
Initialized ProgressiveAntCurriculum with stage: basic_balance (difficulty: 0.10)
Initialized ProgressiveAntCurriculum with stage: basic_balance (difficulty: 0.10)
Difficulty 1.00: Curriculum=55.8±66.1, Standard=34.3±57.2
```




```
Results:
- Curriculum average reward: 100.9
- Standard average reward: 26.5
- Overall improvement: 280.9%

Robustness:
- Curriculum performance retention: 65.2%
- Standard performance retention: 92.9%
- Retention difference: -27.7%
```