Self-Driving Car with PPO (CarRacing-v2)

This project implements a Reinforcement Learning (RL) agent using the **PPO (Proximal Policy Optimization)** algorithm to solve the `CarRacing-v2` environment from Gymnasium.

The project features two main logic branches:

1. **Standard Training:** The agent learns to maximize the default environment reward.
2. **"Grass Penalty" Training (Files ending in `*2.py`):** An advanced implementation that actively penalizes the agent if it drives on the grass, forcing it to stay on the asphalt track.

Features

- **Algorithm:** PPO with Actor-Critic architecture.
- **Vision:** Convolutional Neural Network (CNN) processing 4 stacked grayscale frames.
- **Grass Penalty Wrapper (v2.0):** A custom *wrapper* that detects if the car goes off-track by analyzing the Region of Interest (ROI) and applies penalties or "sudden death".
- **Evaluation Pipeline:** Scripts to evaluate models, generate comparative plots, correlation heatmaps, and detailed reports.

Project Structure

The source code is located in the `src/` folder. The key distinction is the `2` suffix in the filenames:

Core Files

- **`agent.py`**: Defines the Neural Network (CNN) and the `Agent` class (Actor-Critic).
- **`utils.py`**: Basic utilities to create the standard environment.
- **`utils2.py`**: **IMPORTANT.** Contains the `GrassPenaltyWrapper`. Used in all version `2` scripts.
- **`compare_models.py`**: Class to generate comparative plots and statistical reports across multiple models.

Training

- **`train.py`**: Standard PPO training. Saves to `../Models/models_A`.
- **`train2.py`**: **Grass Penalty Training.** Uses `utils2.py` to penalize off-track driving. Saves to `../Models/models_B`.

Evaluation

- **`evaluate_pro.py`**: Detailed evaluation for standard models. Generates videos and JSON.
- **`evaluate_pro2.py`**: **Evaluation for "Grass Penalty" models.** Activates grass detection during evaluation.
- **`run_evaluation_pipeline.py`**: Master script. Evaluates a list of models sequentially and generates a final comparison.

Usage Guide

1. Train the Agent

To train a new model from scratch, run one of the training scripts.

Option A: Strict Training (Recommended) This method penalizes the agent for touching the grass, resulting in more precise driving. `python src/train2.py` *(Models will be automatically saved in the `Models/models_B` folder)*.

Option B: Standard Training `python src/train.py`

2. Evaluate an Individual Model

To test a specific model (.pth), view its metrics, and generate a video:

1. Open `src/evaluate_pro2.py`.
2. Go to the end of the file (`if __name__ == "__main__":`).
3. Modify the variable (e.g., `model_3m`) with the **absolute or relative path** to your `.pth` file.
4. Run: `python src/evaluate_pro2.py` *(Results will be saved in `evaluation_results/<model_name>/`)*.

3. Run Comparative Pipeline

If you have multiple checkpoints (e.g., 200k, 500k, 1M, 2M steps) and want to visualize the progress:

1. Open `src/run_evaluation_pipeline.py`.
2. In the `models_to_evaluate` dictionary, update the paths to your `.pth` files.
3. Run the script: `python src/run_evaluation_pipeline.py`
4. The script will evaluate each model one by one and generate reports and plots in the `comparison_analysis/` folder.

Results and Plots

The evaluation system automatically generates:

- **Text Reports:** Detailed statistics on reward, episode length, and control usage (brake/throttle).
- **Plots:** Learning curves, reward distributions, driving profile radar charts, and correlation matrices.
- **Videos:** Recordings of the evaluation episodes. **These videos will be created in the `videos` folder, which is automatically created inside the `src` directory.**