Technische Dokumentation: Pokémon Emerald RL-Agent

IT-Architektur und Systembeschreibung

Projektübersicht für Studenten

27. Dezember 2025

Inhaltsverzeichnis

1 Einführung

Dieses Projekt realisiert einen autonomen Agenten für das Spiel *Pokémon Emerald* (Gameboy Advance). Die technische Basis bildet das **Reinforcement Learning** (Bestärkendes Lernen), spezifisch der **PPO-Algorithmus** (Proximal Policy Optimization). Der Agent lernt durch Interaktion mit der Spielumgebung, Belohnungen zu maximieren.

2 IT-Architektur

Das System folgt einer klassischen Agent-Environment-Struktur, die jedoch für die Emulation optimiert wurde.

2.1 Schichtmodell

1. **Umgebungsschicht (Emulator):** Der *VisualBoyAdvance-M* Emulator führt die ROM (`temp_emerald.gba`) aus. Er dient als Blackbox, die Zustände (Frames/RAM) liefert.
2. **Schnittstellenschicht (Bridge):** Python-Skripte lesen den Speicherzustand (RAM) und den Grafikpuffer aus, um dem Agenten Informationen über KP, Position und Spielmenüs zu geben.
3. **Entscheidungsschicht (KI-Modell):** Ein tiefes neuronales Netzwerk (Deep Neural Network), das basierend auf dem aktuellen Zustand die Wahrscheinlichkeit für Aktionen (Steuerkreuz, Buttons) berechnet.
4. **Persistenzschicht (Datenbanken & Checkpoints):** Speicherung von Fortschritt, räumlichen Daten und Trainingsmetriken in SQLite-Datenbanken und ZIP-Dateien.

3 Modulbeschreibung: Wer macht was?

Das Projektverzeichnis ist modular aufgebaut. Jede Datei hat eine spezifische Aufgabe im Trainingszyklus:

4 Funktionsweise des Codes

4.1 Der Trainingszyklus

Beim Ausführen von `train.py` wird ein kontinuierlicher Loop gestartet:

- **Beobachtung (Observation):** Der Agent extrahiert Daten aus dem Spiel.
- **Aktion (Action):** Das Modell wählt einen Tastendruck (z.B. „RECHTS“).
- **Belohnung (Reward):** Das System evaluiert den Erfolg (z.B. Erfahrungspunkte gewonnen = positive Belohnung).
- **Optimierung:** Das Modell passt seine internen Gewichte an, um zukünftige Belohnungen zu erhöhen.

Kategorie	Datei	Beschreibung
Core	`train.py`	Das Hauptskript. Initialisiert die KI-Umgebung und startet die Trainingsschleife.
	`pokemon_agend.py`	Enthält die Definition des Agenten und der neuronalen Netzwerkarchitektur.
Memory	`new_memory.py`	Verwaltet das räumliche Gedächtnis (Speicherung von Gras, Wasser und POIs).
	`ai_ppo_memory.json`	Persistenter Kurzzeitspeicher für den PPO-Algorithmus.
Analyse	`roi_test.py`	Testet die „Region of Interest“-Erkennung (Screen-Scanning).
	`analyze_battles.py`	Wertet Kampflogs aus, um die Strategie-Effizienz zu prüfen.
Storage	`pokemon_ai.db`	Zentrale SQLite-Datenbank für Forschungs- und Agentendaten.
	`cfg.json`	Zentrale Konfiguration für Hyperparameter und Pfade.

Tabelle 1: Übersicht der funktionalen Komponenten

5 Anleitung: System starten

Um das Projekt für eine Demonstration oder weiteres Training zu starten, folgen Sie diesen Schritten im Terminal:

5.1 Schritt 1: Vorbereitung

Stellen Sie sicher, dass die virtuelle Python-Umgebung aktiviert ist, um alle Abhängigkeiten (PyTorch, Stable-Baselines3) zu laden:

```
cd path/to/Gameboy
.\.venv\Scripts\activate
```

5.2 Schritt 2: Training/Ausführung starten

Führen Sie das Trainingsskript aus. Dies lädt automatisch den letzten Checkpoint aus dem Ordner `checkpoints/`, falls vorhanden (*History Restored*):

```
python train.py
```

5.3 Schritt 3: Überwachung

Um die Lernkurve (FPS, Loss, Reward) in Echtzeit zu sehen, kann das Dashboard genutzt werden:

```
python dashboard.py
```

6 Zusammenfassung

Dieses System ist ein Beispiel für ein **End-to-End Reinforcement Learning Framework**. Durch die Trennung von Spiellogik (Emulator) und Entscheidungslosik (Python/-PyTorch) ist es hochgradig modular und ermöglicht es der KI, komplexe Aufgaben wie das Navigieren zu speziellen Orten (Gras/Wasser) durch Erfahrung zu lernen.