

Title

Einführung des Singleton-Patterns für die Klasse Mars im Mars Rover Projekt

Status

Accepted

Kontext

Im aktuellen Design des Mars Rover Projekts interagieren verschiedene Komponenten mit einer zentralen Mars-Instanz. Das Klassendiagramm zeigt klar, dass es im gesamten System nur einen Mars geben kann und soll. Die Rover-Klasse ruft die Methode `drive(cmd: str)` auf Mars auf, und `MissionControl` sendet Befehle über eine REST-API.

Um sicherzustellen, dass alle Teile des Systems mit exakt derselben Mars-Instanz arbeiten – und somit Konsistenz der Zustände (z. B. Positionen, Hindernisse) gewährleistet ist – wurde überlegt, ob das **Singleton-Pattern** verwendet werden soll.

Zur Implementation wurde sich an den Diskussionen und Best Practices auf StackOverflow orientiert:

➔ [How to implement a singleton in Python? – StackOverflow](#)

Die Diskussion über das Singleton-Muster wurde auch auf die `Plateau`-Klasse ausgeweitet, da sie die interne Darstellung des Mars-Geländes (`grid`) kapselt.

Entscheidung

Die Klasse `Mars` wird als **Singleton** implementiert. Die Instanziierung erfolgt kontrolliert, sodass im gesamten Lebenszyklus der Anwendung genau eine Instanz existiert. Dadurch wird sichergestellt, dass alle `Rover`, `MissionControl` und Beobachtungsinstrumente denselben Mars-Zustand sehen und manipulieren.

Die Klasse `Plateau` **wird *nicht*** als Singleton implementiert. Begründung:

Argumente für Singleton `Plateau`:

- Es gibt nur ein `Plateau` pro Mars-Instanz.
- Verhindert ungewollte Mehrfachinstanzen und Inkonsistenzen im Grid.
- Könnte initial ebenfalls zentral verwaltet werden.

Argumente gegen Singleton `Plateau`:

- `Plateau` ist eine Implementierungsdetails von `Mars` und wird nicht direkt von außen angesprochen.
- Durch die Kapselung innerhalb von `Mars` kann die Lebensdauer und Instanziierung ohnehin intern kontrolliert werden.
- Eine zweite Singleton-Instanz erhöht Komplexität ohne zusätzlichen Nutzen.
- Testbarkeit leidet durch globale Zustände.

Daher bleibt `Plateau` eine reguläre Klasse, die innerhalb von `Mars` verwaltet wird.

Konsequenzen

Vorteile:

- Konsistenter Zustand über alle Systemkomponenten hinweg (nur ein `Mars`-Zustand).
- Geringere Fehleranfälligkeit durch ungewollte Mehrfachinstanzen.
- Die Architektur entspricht der realen Modellierung: es gibt nur einen Mars.

Nachteile:

- Singleton-Pattern erschwert teilweise das Testen (z. B. parallele Tests mit unterschiedlichen Zuständen).
- Globale Zustände können langfristig zu Designverengungen führen, wenn Flexibilität benötigt wird.
- Bei falscher Anwendung kann Singleton zu unerwünschten Seiteneffekten führen (z. B. unerwartete geteilte Zustände).

Maßnahmen:

- Implementierung erfolgt nach bewährtem Python-Ansatz mit Metaklasse oder Modul-Level-Variable (siehe StackOverflow-Link).
- In Tests wird gezielt auf Isolierung des Zustands geachtet, ggf. durch Reset-Mechanismen oder Mocks.

Dieses ADR dokumentiert eine strukturell wichtige Designentscheidung und stellt sicher, dass die Architektur auf reale Gegebenheiten (nur ein Mars) korrekt abbildet.