Projekt Präsentation - Polygon sketching

Artner Patrick, Dedinak Sebastian

Funktionale Programmierung, 15.01.2025

Reflexion der Baseline:
Identifizierung der verwendeten Funktionen

Der Code ist größtenteils nicht funktional programmiert.

Funktionale Elemente im Code:

- **Array.forEach**
  - Nutzung einer Higher-Order-Funktion. Deklarativer als eine for-Schleife.
- **Kapselung von Logik in Funktionen**
  - Formal eine Funktion
- **JSON.stringify({ polygons, currentPolygon, color });**
  - Zustand wird als Wert behandelt. Snapshot-Idee ähnelt funktionalen State-Modellen

Reflexion der Baseline: Was war schwierig?

- “Doppelklick beendet das Polygon” Logik
- Sonst nicht viel

Reflexion der Baseline: Was könnte verbessert werden?

- Farbauswahl zurzeit begrenzt, könnte erweitert werden
- Fachlogik von UI trennen
- Nutzung funktionaler Array-Operationen zB: map
- Undo/Redo durch State-Historie ersetzen zB: past: [...history.past, history.present], present: newState, future: []
- Ein einziges State-Objekt, das nicht mutiert, sondern ersetzt wird zB: let state = {polygons: [],currentPolygon: []}

Reflexion des rein funktionalen Ansatzes: Identifizierung der verwendeten Funktionen

Der Code ist soweit möglich funktional programmiert.

Funktionale Elemente im Code:

- Immutable Datenstrukturen (das Model wird nie verändert)
- Reine Funktionen (immer dasselbe Ergebnis für dieselben Eingaben)
- Explizite Zustandsübergänge mittels messages (Msg)
- Pattern Matching statt IFs
- Keine Seiteneffekte in der Kernlogik (Seiteneffekte nur in der UI-Schicht)

Reflexion des rein funktionalen Ansatzes: Was war schwierig?

- “Doppelklick beendet das Polygon” Logik
- Umdenken und hineindenken in funktionale Programmierung und deren Syntax

Reflexion des rein funktionalen Ansatzes: Was könnte verbessert werden?

- Weitere Aufteilung der Logik in kleinere Funktionen
- Strengere Trennung von Logik und UI
- Anzeigen von Verwendungsmöglichkeiten des Programms
- Farbauswahl zurzeit begrenzt, könnte erweitert werden

Reflexion der Ansätze im Vergleich: Funktional vs Imperativ

Erweiterbarkeit:

Funktionale Programmierung ist besser erweiterbar, weil neue Funktionen einfacher ergänzt werden können, ohne bestehenden Code stark zu verändern

Prüfbarkeit:

Funktionaler Code ist leichter zu testen, da Funktionen meist ohne Seiteneffekte und versteckte Abhängigkeiten arbeiten und einzeln überprüft werden können

Wartbarkeit:

Imperativer Code ist leichter zu lesen und der Programmablauf ist direkt nachvollziehbar, da Zustandsänderungen sofort sichtbar sind

Komplexität:

Funktionale Programmierung ist am Anfang schwer zu verstehen aber führt langfristig wahrscheinlich zu weniger kompliziertem Code als der imperative Ansatz