SmartPacks Power-Pack Generator: Systemdokumentation

Überblick

Dieses System analysiert mathematische Schülerfehler im Zahlenraum bis 20 und generiert didaktisch fundierte "SmartPacks Power-Packs" zur gezielten Förderung.

1. Fehler-Kategorisierung (math-pedagogy.ts)

1.1 Erkannte Fehlerkategorien

Das System unterscheidet 9 Hauptfehlerkategorien:

Arithmetische Fehler:

1. **`zehnuebergang_addition`** - Zehnerübergang bei Addition
   - Erkennungskriterium: Summe > 10, beide Summanden ≤ 10
   - Beispiel: 8 + 5 = 12 (statt 13)

2. **`zehnuebergang_subtraction`** - Zehnerübergang bei Subtraktion
   - Erkennungskriterium: Minuend > 10, Differenz < 10
   - Beispiel: 13 - 9 = 5 (statt 4)

3. **`complementary_pairs`** - Partnerzahlen nicht erkannt
   - Erkennungskriterium: Summe = 10 oder 20, aber falsch gerechnet
   - Beispiel: 7 + 3 = 11 (statt 10)

4. **`calculation_facts`** - Lücken bei Grundrechenarten
   - Allgemeine Rechenfehler ohne spezifisches Muster

5. **`subtraction_borrowing`** - Allgemeine Subtraktionsschwierigkeiten
   - Fehler bei Subtraktionsaufgaben außerhalb der anderen Kategorien

Konzeptuelle Fehler:

6. **`number_reversal`** - Zahlenreihenfolge vertauscht
   - Erkennungskriterium: Bei Subtraktion wird gerechnet: num2 - num1 statt num1 - num2
   - Beispiel: 13 - 5 → Kind rechnet 5 - 13

7. **`digit_reversal`** - Ziffern vertauscht (Zahlendreher)
   - Erkennungskriterium: Ergebnis ist Spiegelung der korrekten Zahl
   - Beispiel: 12 statt 21, 92 statt 29

8. **`operation_confusion`** - Operationsvertauschung
   - Erkennungskriterium: Ergebnis passt zur anderen Operation
   - Beispiel: 8 + 5 = 3 (Kind subtrahiert statt addiert)

9. **`pattern_break`** - Muster nicht erkannt
   - Fehler in strukturierten Aufgabenreihen

1.2 Klassifikationsalgorithmus

```typescript
function classifyErrors(errors: StudentError[]): ClassifiedError[]
```

Ablauf:

1. Trennung nach Operationen (Addition/Subtraktion)
2. Sequentielle Prüfung jeder Kategorie mit spezifischen Kriterien
3. Sammlung aller betroffenen Zahlen (targetNumbers)
4. Rückgabe klassifizierter Fehler mit didaktischer Beschreibung

Mathematische Kriterien-Beispiele:

- Zehnerübergang Addition: `(num1 + num2 > 10 && num1 ≤ 10 && num2 ≤ 10)`
- Zehnerübergang Subtraktion: `(num1 > 10 && (num1 % 10) < num2)`
- Zahlendreher: Stringvergleich nach Ziffernumkehrung

2. Päckchen-Bibliothek (paeckchen-library.ts)

2.1 Päckchen-Typen und didaktische Ziele

Das System enthält **12 verschiedene Päckchen-Typen**, jeweils mit spezifischem didaktischen Fokus:

Strukturmuster-Päckchen:

1. **`constant_sum`** - Konstanz der Summe
   - Mathematisches Prinzip: Gegensinnige Veränderung (a↑+1, b↓-1, Summe=)
   - Beispiel: 5+8=13, 6+7=13, 7+6=13
   - Didaktischer Wert: Einsicht in Zahlbeziehungen, flexible Zerlegungsstrategien

2. **`opposing_change`** - Gegensinnige Veränderung bei Subtraktion
   - Mathematisches Prinzip: Minuend konstant, Subtrahend↑ → Differenz↓
   - Beispiel: 15-3=12, 15-4=11, 15-5=10
   - Didaktischer Wert: Zusammenhang Subtrahend-Differenz verstehen

3. **`same_direction`** - Gleichsinnige Veränderung
   - Mathematisches Prinzip: Beide Summanden↑+1 → Summe↑+2
   - Beispiel: 3+4=7, 4+5=9, 5+6=11
   - Didaktischer Wert: Erkennen kumulativer Effekte

Strategische Päckchen:

4. **`crossing_tens`** - Zehnerübergang meistern
   - Systematisches Üben des Zehnerübergangs
   - Beispiel: 8+2=10, 8+3=11, 8+4=12
   - Didaktischer Wert: Routine und Sicherheit beim Übergang

5. **`decomposition_steps`** - Zerlegungspäckchen
   - Mehrstufige Darstellung (z.B. 13-9 = 13-3-6 = 10-6 = 4)
   - Didaktischer Wert: Explizite Strategie-Visualisierung

Konzeptuelle Päckchen:

6. **`inverse_tasks`** - Umkehraufgaben
   - Prinzip: a+b=c ⇒ c-a=b, c-b=a
   - Didaktischer Wert: Operationsverständnis durch Umkehrbeziehung

7. **`exchange_tasks`** - Tauschaufgaben
   - Kommutativgesetz: a+b = b+a
   - Didaktischer Wert: Flexibilität, Aufgaben vereinfachen

8. **`number_reversal_demo`** - Reihenfolge beachten
   - Kontrastierung: 15-8 ≠ 8-15
   - Didaktischer Wert: Nicht-Kommutativität der Subtraktion verstehen

9. **`operation_contrast`** - Addition vs. Subtraktion
   - Direkter Vergleich: 12+7=19 vs. 12-7=5
   - Didaktischer Wert: Operationsverständnis schärfen

Metakognitive Päckchen:

10. **`error_research`** - Fehlerforschungs-Päckchen
    - Absichtlicher Fehler im Päckchen versteckt
    - Didaktischer Wert: Musterverständnis durch Fehleranalyse

11. **`pattern_analysis`** - Muster-Analyse
    - Frage: "Ist das ein schönes Päckchen?"
    - Didaktischer Wert: Kritisches Denken, mathematisches Argumentieren

12. **`digit_detective`** - Zahlendreher-Detektiv
    - Stellenwertübungen: 12 ≠ 21
    - Didaktischer Wert: Sichere Ziffernfolge, Stellenwertverständnis

13. **`continuation_challenge`** - Fortsetzungs-Challenge
    - Kind setzt Muster selbstständig fort
    - Didaktischer Wert: Transfer, kreatives mathematisches Denken

2.2 Zuordnung Fehler → Päckchen

```typescript
function selectPaeckchenTypeForError(errorType: string): PaeckchenType[]
```

Mapping-Strategie:
- Jeder Fehlertyp → 2-3 passende Päckchen (aufsteigend nach Schwierigkeit)
- Beispiel Zehnerübergang Addition:
  1. `crossing_tens` (üben)
  2. `constant_sum` (verstehen)
  3. `decomposition_steps` (Strategie)

2.3 Generator-Funktionen

Jeder Päckchen-Typ hat eine spezialisierte Generator-Funktion:

Beispiel: `generateConstantSumPaeckchen(targetSum, startNum, operator)`

```typescript
// Generiert: 5+8=13, 6+7=13, 7+6=13, 8+5=13, 9+4=13
for (let i = 0; i < 5; i++) {
  const num1 = startNum + i;
  const num2 = targetSum - num1;
  problems.push(`${num1} + ${num2} = ___`);
}
```

Mathematische Intelligenz:
- Automatische Bereichsprüfung (Zahlenraum 0-20)
- Kontextanpassung (Addition/Subtraktion)
- Musterkonsistenz durch algorithmische Generierung

3. Päckchen-Demonstrator (paeckchen-demonstrator.ts)

3.1 Fehleranalyse-Pipeline

```typescript
function analyzeMathError(input: DemonstrationInput): ErrorAnalysis
```

Schritte:
1. Task-Parsing: Extraktion von num1, operator, num2
2. Einzelfehler-Klassifikation via `classifySingleError()`

3. Didaktische Erklärung aus Wissensbasis abrufen

4. Rückgabe vollständiger ErrorAnalysis

Wissensbasis-Struktur:

```typescript
{
  errorType: string,
  errorDescription: string,        // Was ist passiert?
  didacticExplanation: string,     // Warum passiert das?
  commonCause: string,             // Typische Ursache
  learningGap: string              // Was muss gelernt werden?
}
```

3.2 Päckchen-Empfehlung

```typescript
function generatePaeckchenRecommendations(
  errorAnalysis: ErrorAnalysis,
  input: DemonstrationInput
): PaeckchenRecommendation[]
```

Intelligente Kontextanpassung:

- Bei Addition: Summe als targetNumber
- Bei Subtraktion: Minuend als targetNumber
- Dynamische Beispielgenerierung mit Original-Zahlen
- Begründung "whyThisHelps" für jedes Päckchen

Beispiel-Output:

```javascript
{
  paeckchenType: 'crossing_tens',
  template: { name: 'Zehnerübergang meistern', ... },
  example: {
    title: 'Über den Zehner springen',
    problems: ['8+2=___', '8+3=___', '8+4=___', ...]
  },
  whyThisHelps: 'Das Päckchen übt systematisch den Zehnerübergang...'
}
```

3.3 Visualisierungs-Vorschläge

```typescript
function getVisualizationSuggestions(errorType: string): string[]
```

Kategorisierte Vorschläge:
- **Material:** Plättchen, Zehnerstangen, Einerwürfel
- **Darstellung:** Pfeile, Zahlenstrahl, Zerlegungsbaum
- **Markierungen:** Farben für Stellenwerte, Operationszeichen hervorheben

Mathematikdidaktische Prinzipien:
- Enaktiv → Ikonisch → Symbolisch (Bruner)
- Handelndes Lernen mit Material
- Visualisierung von Denkprozessen

3.4 Lehrpersonen-Guidance

```typescript
function generateTeacherGuidance(
  errorAnalysis: ErrorAnalysis,
  recommendations: PaeckchenRecommendation[]
): string
```

Strukturierte Anleitung:
1. **Didaktische Einordnung:** Fehler + Lernlücke
2. **Empfohlenes Vorgehen:** 4-Schritte-Plan
3. **Differenzierung:** Leicht → Mittel → Schwer

Pädagogische Prinzipien:
- Nicht nur rechnen, sondern begründen lassen
- Muster mit eigenen Worten beschreiben
- Visualisierung nutzen

4. Mathematikdidaktische Fundierung

4.1 Theoretische Basis

Zentrale Konzepte:
1. **Operative Durchdringung** (Wittmann)
   - Beziehungen zwischen Aufgaben erkennen
   - Muster und Strukturen nutzen

2. **Produktives Üben**
   - Nicht stures Wiederholen
   - Entdeckendes Lernen durch Päckchen

3. **Sprachförderung**
   - Sentence Stems für mathematische Argumentation
   - Fachsprache durch Reflexionsfragen

4.2 Fehlertypen und Ursachen

Tiefenstruktur der Fehler:
- **Zehnerübergang:** Fehlende Zerlegungskompetenz
- **Partnerzahlen:** Keine Automatisierung
- **Zahlendreher:** Stellenwertverständnis unsicher
- **Operationsverwechslung:** Konzeptuelles Defizit

4.3 Päckchen als didaktisches Werkzeug

Warum "Power-Packs" (Entdecker-Päckchen)?
1. **Musterorientierung:** Strukturen sichtbar machen
2. **Entlastung:** Nicht jede Aufgabe neu rechnen
3. **Transferförderung:** Prinzipien auf neue Aufgaben anwenden
4. **Selbstkontrolle:** Muster erlaubt Plausibilitätsprüfung

Differenzierungsmöglichkeiten:
- Schwierigkeitsstufen (easy/medium/hard)
- Verschiedene Päckchen-Typen für denselben Fehler
- Scaffolding durch Fortsetzungs-Challenges

5. Technische Implementation

5.1 Datenfluss

```
```

Fehler-Input → Klassifikation → Päckchen-Auswahl → Generierung → Begründung

```

### 5.2 Kernfunktionen

**Klassifikation:**

- `classifyErrors()`: Batch-Verarbeitung
- `classifySingleError()`: Einzelfehler

**Generierung:**

- 12 spezialisierte Generator-Funktionen
- Einheitliches `GeneratedPaeckchen`-Interface

**Demonstrator:**

- `demonstratePaeckchenGeneration()`: Hauptfunktion
- Vollständige Analyse-Pipeline

### 5.3 Erweiterbarkeit

**Neue Fehlertypen hinzufügen:**

1. ErrorCategory erweitern
2. Klassifikationskriterium in `classifyErrors()` ergänzen
3. Neuen Päckchen-Typ entwickeln
4. Mapping in `selectPaeckchenTypeForError()` eintragen

**Neue Päckchen-Typen:**

1. Template in `PAECKCHEN_TEMPLATES` definieren
2. Generator-Funktion implementieren
3. In Zuordnung aufnehmen
```

6. Forschungs- und Optimierungspotential

6.1 Aktuelle Stärken

- Präzise Fehlerklassifikation
- Didaktisch fundierte Päckchen-Auswahl
- Kontextspezifische Begründungen
- Umfassende Lehrpersonen-Unterstützung

6.2 Optimierungsbereiche

Mathematisch:

- Erweiterte Muster-Erkennung (z.B. geometrische Reihen)
- Komplexere Zerlegungsstrategien
- Zahlenraumerweiterung (bis 100, 1000)

Didaktisch:

- Adaptive Schwierigkeitsanpassung basierend auf Lernfortschritt
- Langzeit-Tracking von Fehlermustern
- Personalisierte Päckchen-Sequenzen

Technisch:

- Machine Learning für Fehlerprognose
- Automatische Visualisierungsgenerierung
- Integration von Lernstandserhebungen

6.3 Forschungsfragen

1. **Welche Päckchen-Typen sind am effektivsten?**
   - A/B-Testing verschiedener Ansätze
   - Lernerfolg messen

2. **Wie viele Päckchen pro Fehlertyp sind optimal?**
   - Übungsumfang vs. Lernerfolg
   - Differenzierung nach Leistungsniveau

3. **Kann das System Fehlerursachen vorhersagen?**
   - Musteranalyse über Zeit
   - Präventive Förderung

7. Fazit

Das System verbindet tiefe mathematikdidaktische Expertise mit intelligenter Automatisierung:

Innovation:
- Automatische Fehlerdiagnose auf konzeptueller Ebene
- Generierung kontextspezifischer, didaktisch wertvoller Übungen
- Transparente Begründungen für Lehrpersonen

Mathematikdidaktischer Mehrwert:
- Fokus auf Verstehen statt Auswendiglernen
- Operative Durchdringung durch Musterarbeit
- Sprachförderung durch Reflexionsimpulse

Praktischer Nutzen:
- Zeitersparnis für Lehrpersonen
- Individuelle Förderung skalierbar
- Evidenzbasierte Päckchen-Auswahl

Das System bildet eine solide Basis für weitere Forschung zur automatisierten, adaptiven Mathematikförderung im Grundschulbereich.