Dokumentation für das Modul Aufbaumodul Programmieren
Von Josepha Nangmo
Immatrikulationsnummer 301162

Konzeptbeschreibung des Projektes

Das Sudoku-Projekt ist eine praktische Umsetzung grundlegender Programmierkonzepte, die im Rahmen der objektorientierten Softwareentwicklung angewendet werden. Ziel dieses Projekts ist es gewesen, ein voll funktionsfähiges 4x4-Sudoku-Spiel zu entwickeln, das auf Prinzipien wie Vererbung, Modularität und Wiederverwendbarkeit aufbaut. Dieses Projekt ist ein Projekt, welches vorher schon für das Modul „Grundlagen der Programmierung“ benutzt und für dieses Modul wiederverwendet wurde.

Objektorientierte Entwicklung

Das Sudoku-Projekt wurde unter Berücksichtigung der Prinzipien der objektorientierten Programmierung (OOP) entworfen. Die Basislogik für die Darstellung und Validierung des Sudoku-Boards wird in der Klasse *SudokuBase* definiert, während die spezifische Spiellogik in der abgeleiteten Klasse *SudokuGame* implementiert ist. Dieses Design fördert Wiederverwendbarkeit und Modularität.

- **Klassenstruktur:**
    - *SudokuBase:* Enthält grundlegende Methoden zur Validierung von Reihen, Spalten und Blöcken sowie zum Drucken des Boards.
    - *SudokuGame:* Erbt von *SodukuBase* und erweitert die Funktionalität durch Spiellogik wie das Maskieren von Zellen und das interaktive Spielen.

Vererbung

Die Vererbung wird genutzt, um Code-Duplizierung zu vermeiden und die Spiellogik klar von der Grundlogik zu trennen. Die Klasse *SudokuGame* ruft Funktionen der Basisklasse auf, um Validierungen und Board-Operationen auszuführen.

Beispiel:

- *self.is_valid_row(...)* wird sowohl in der Lösungserstellung als auch bei der Spielerinteraktion verwendet.

Pakete und Modularisierung

Das Projekt ist in zwei Module aufgeteilt:

- **Sudoku_base.py:** Enthält grundlegende Methoden und Attribute, die von anderen Klassen verwendet werden.
- **Sudoku_game.py:** Enthält die Spiellogik und die Benutzerinteraktion.

Diese Struktur erleichtert die Wartung und Wiederverwendung der Komponenten. Zum Beispiel könnte base.py in anderen Sudoku-Varianten verwendet werden.

Technische Implementierung

1. Lösungserstellung

Die Methode fill_board füllt das Sudoku-Board mit einer gültigen Lösung. Dabei wird Backtracking verwendet, um sicherzustellen, dass alle Sudoku-Regeln eingehalten werden.

- **Algorithmus:**
    - Zahlen von 1 bis 4 werden zufällig in einer Zelle platziert.
    - Validierungen (Reihen, Spalten, Blöcke) werden durchgeführt.
    - Backtracking korrigiert Fehler bei ungültigen Platzierungen.

2. Maskierung der Zellen

Die Methode mask_cells entfernt Zahlen vom gelösten Board, um ein spielbares Puzzle zu erstellen. Hierbei werden zufällige Felder auf 0 gesetzt.

- **Parameter:**
    - count: Anzahl der zu maskierenden Felder (Standard: 8).

3. Interaktive Spiellogik

Die Methode play_game erlaubt dem Spieler, interaktiv Züge einzugeben. Die Eingaben werden geprüft, und der Spieler erhält Feedback, ob der Zug gültig war.

- **Eingabeformat:** row column number (z. B. 0 1 3).
- **Beendigung:** Der Spieler kann das Spiel mit quit verlassen.

Änderungen und Erweiterungen

Die folgenden Änderungen wurden gegenüber der ursprünglichen Version vorgenommen:

1. **Vererbung hinzugefügt:**
    - Die ursprüngliche Version verwendete keine Basisklasse. Die aktuelle Version führt SudokuBase ein, um Grundfunktionen zu kapseln.
2. **Modularisierung:**
    - Der Code wurde in zwei Module (base.py, game.py) aufgeteilt.
3. **Detailliertere Validierung:**
    - Die Methoden is_valid_row, is_valid_column und is_valid_box wurden erweitert, um zwischen Spieler- und Lösungsprüfungen zu unterscheiden.
4. **Speicher- und Ladefunktion:**
    - Spieler können ihren Fortschritt speichern (save_to_file) und laden (load_from_file).

Funktionsweise und Kommentare

Der gesamte Code ist kommentiert, um die Funktionsweise der wichtigsten Methoden zu erläutern. Hier ein Beispiel:

```
class SudokuBase:
    def print_board(self, board):
        # Print the Sudoku board with borders
        print("+---+---+---+---+")
        for row in board:
            print("| " + " | ".join(str(num) if num != 0 else "_" for num in row) + " |")
            print("+---+---+---+---+")
```

Die Kommentare sind präzise und erklären den Zweck jeder Methode.