11 Praxisbeispiel: Tic Tac Toe

Tic Tac Toe ist ein Strategiespiel für zwei Personen. Gespielt wird auf einem Spielfeld mit 3 × 3 Feldern. Die Spieler markieren abwechselnd freie Positionen entweder mit einem Kreuz oder einem Kreis. Derjenige, der zuerst 3 gleiche Formen in einer Reihe horizontal, vertikal oder diagonal erreicht, gewinnt das Spiel.

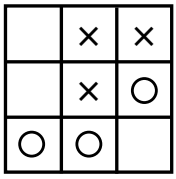

Abbildung 11-1 *Beispiel für ein Tic-Tac-Toe-Spielfeld*

Im Folgenden ist es das Ziel, ein Grundgerüst für ein Tic-Tac-Toe-Spiel zu erstellen, das aber noch genug Raum für eigene Experimente und Erweiterungen bietet.

11.1 Spielfeld initialisieren und darstellen

Zum Einstieg müssen wir eine geeignete Repräsentation bzw. Modellierung des Spielfelds finden. Für das 3 × 3-Spielfeld bietet sich eine verschachtelte Liste mit jeweils 3 Einträgen pro Richtung an. Die Symbole Kreuz und Kreis kann man gut durch die Buchstaben X und O repräsentieren. Für ein leeres Feld könnte man ein Leerzeichen nutzen. Aus Darstellungsgründen wollen wir ein Minuszeichen verwenden.

Spielfeld initialisieren

Nachfolgend schreiben wir eine Funktion, die mithilfe von List Comprehensions ein neues zweidimensionales Gebilde erzeugt und dabei die Zeilen und Spalten je 3-mal durchläuft und jeweils an passender Position ein Minuszeichen speichert:

```
def initialize_board():
   board = [['-' for col in range(3)] for row in range(3)]
   return board
```

Spielfeld darstellen

Machen wir uns an die Darstellung. Diese erfolgt zwar auf der Konsole, soll aber doch ein wenig ansprechend sein und die Felder als Rechtecke visualisieren. Dazu verwenden wir horizontale und vertikale Linien. In Python realisieren wir das, indem wir über alle Zeilen und Spalten laufen und an passender Stelle die Linienbestandteile ergänzen:

```
def print_board(board):
   print("-------------")

   for row in range(3):
       print("| ", end="")
       for col in range(3):
           print(board[row][col] + " | ", end="")
       print()
       print("-------------")
```

Probieren wir das einmal aus:

```
tictactoe_board = initialize_board()
print_board(tictactoe_board)
```

Damit erhalten wir folgende Darstellung:

```
-------------
| - | - | - |
-------------
| - | - | - |
-------------
| - | - | - |
-------------
```

11.2 Setzen der Steine

Nachdem das Spielfeld bereit ist, wollen wir das Ganze mit Leben füllen. Laut Regelwerk wechseln sich die Spieler ab und platzieren an freien Positionen entweder ihr Kreuz oder ihren Kreis. Die folgende Methode prüft zunächst, ob die durch Zeile `row` und Spalte `col` übergebene Position im Spielfeld noch frei ist. Nur dann wird der übergebene Spielstein gesetzt.

```
def place_mark(board, row, col, current_player_mark):
    if board[row][col] == '-':
        board[row][col] = current_player_mark
        return True

    return False
```

Probieren wir das wiederum einmal aus:

```
place_mark(tictactoe_board, 0, 0, 'X')
place_mark(tictactoe_board, 1, 1, 'O')
place_mark(tictactoe_board, 0, 1, 'X')

print_board(tictactoe_board)
```

Das sieht doch schon prima aus:

```
-------------
| X | X | - |
-------------
| - | O | - |
-------------
| - | - | - |
-------------
```

Natürlich kann man hier noch feilen und erweitern. Darauf gehe ich später kurz ein.

Was auf jeden Fall noch implementiert werden soll, ist die Prüfung, ob ein Spieler gewonnen hat.

11.3 Prüfen auf Sieg

Um festzustellen, ob ein Spieler der Sieger ist, muss man alle Horizontalen und Vertikalen sowie die zwei Diagonalen auf drei gleiche Steine prüfen. Wie ab und zu schon mal erwähnt, empfiehlt es sich beim Programmieren, in kleinen Bausteinen und Funktionalitäten zu denken.

Grundgerüst

Daher setzt sich unsere Prüfung auf Sieg aus drei Aufrufen von speziellen Prüfungen der Richtungen zusammen:

```
def check_for_win(board, current_player_mark):
    return check_rows_for_win(board, current_player_mark) or \
           check_columns_for_win(board, current_player_mark) or \
           check_diagonals_for_win(board, current_player_mark)
```

Baustein 1: Prüfung auf drei gleiche Symbole

Okay, dann wollen wir mal die einzelnen Methoden implementieren. Bevor wir direkt loslegen, empfiehlt es sich, ein wenig nachzudenken. Es wird eigentlich immer eine Prüfung auf drei Felder benötigt, ob diese mit dem derzeitigen Spielersymbol übereinstimmen. Das kann man wie folgt implementieren:

```
def check_all_same(c1, c2, c3, current_player_mark):
    return c1 == current_player_mark and c1 == c2 and c2 == c3
```

Baustein 2: Prüfung der Horizontalen und Vertikalen

Mit dieser Hilfsfunktion können wir die horizontale Prüfung so gestalten, dass wir jeweils alle Werte aus den drei Spalten übergeben. Diese Prüfung erfolgt dann zeilenweise für alle drei Zeilen. Analog geschieht dies für die Spaltenprüfung. Dort laufen wir in x-Richtung, also Spalte für Spalte, und nutzen dann die Positionsangaben für die drei Zeilen:

```
def check_rows_for_win(board, current_player_mark):
    for row in range(3):
        if check_all_same(board[row][0], board[row][1], board[row][2],
              current_player_mark):
            return True

    return False


def check_columns_for_win(board, current_player_mark):
    for col in range(3):
        if check_all_same(board[0][col], board[1][col], board[2][col],
              current_player_mark):
            return True

    return False
```

Erneut sehen wir, dass es am einfachsten ist, wenn man Probleme in kleinere Aufgabenstellungen zerlegt und die entstehenden Lösungsbausteine geeignet kombiniert. Dadurch bleibt das Programm auch meistens gut verständlich.

Baustein 3: Prüfung der Diagonalen

So, nun benötigen wir nur noch die Prüfung für die zwei Diagonalen. Das sind die Positionen 0,0, 1,1, 2,2 sowie 0,2, 1,1 und 2,0. Diese prüfen wir folgendermaßen:

```
def check_diagonals_for_win(board, current_player_mark):
    return check_all_same(board[0][0], board[1][1], board[2][2],
                          current_player_mark) or \
           check_all_same(board[0][2], board[1][1], board[2][0],
                          current_player_mark)
```

11.4 Bausteine im Einsatz

Führen wir nun unser Spiel fort, bei dem der Spieler 'O' etwas unaufmerksam ist, wodurch der Spieler 'X' mit der oberen Horizontalen gewinnen sollte:

```
place_mark(tictactoe_board, 2, 1, 'O')
place_mark(tictactoe_board, 0, 2, 'X')

print_board(tictactoe_board)

print("Winner O?", check_for_win(tictactoe_board, 'O'))
print("Winner X?", check_for_win(tictactoe_board, 'X'))
```

Das Ganze wird wie folgt auf der Konsole protokolliert:

```
-------------
| X | X | X |
-------------
| - | O | - |
-------------
| - | O | - |
-------------
Winner O? False
Winner X? True
```

Nicht nur optisch, sondern auch durch unsere Prüffunktion wird X als Sieger ermittelt.

Zwischenfazit

Zwar lässt sich noch die eine oder andere Erweiterung, etwa eine Prüfung auf Gleichstand, ein richtiges Gameplay mit Abwechseln der Spieler usw., implementieren. Wir sind hier mit dem Erreichten aber sehr zufrieden. Warum?

Das Schöne an diesem Programm sind die vielen kleinen Bausteine in Form von Funktionen, die passend ineinandergreifen. Dieser Programmierstil empfiehlt sich vor allem auch für größere Projekte, um möglichst die Übersicht und leichte Erweiterbarkeit zu gewährleisten. Hier profitieren wir davon, dass jede Funktionalität praktischerweise in sich abgeschlossen und damit leicht nachvollziehbar sowie verständlich ist.

Bonus

Wir können die Darstellung des Spielfelds ein wenig abwandeln:

```
def print_board_nicer(board):
    for row in range(3):
        for col in range(3):
            print(" " + str(board[row][col]) + " ", end="")
            if col < 2:
                print("|", end="")
        print()
        if row < 2:
            print("---+---+---")
```

Damit erhalten wir folgende Darstellung:

```
 X | X | X
---+---+---
 - | O | -
---+---+---
 - | O | -
```

Mögliche Erweiterungen

Falls Sie interessiert sind, diesen Prototyp zu einem Spiel weiterzuentwickeln, dann können Sie sich an folgenden Erweiterungen versuchen:

- Modellierung der Spielsteine als `Enum`
- Spieler dürfen nur abwechselnd ziehen
- Benutzereingabe für gewünschte Positionen
- Prüfung auf Gleichstand