

Projekt: Spiel Tic-Tac-Toe in Python programmieren

Hintergrund

Spielablauf

Schritt für Schritt programmieren wir das Spiel Tic-Tac-Toe, was im Deutschen auch unter der Bezeichnung „Drei gewinnt“ bekannt ist.

Das Zweipersonen-Spiel wird auf einem Spielfeld mit 3x3 Feldern gespielt. Abwechselnd setzt jeder Spieler seine Marke (X und O) in ein freies Feld. Wer als erstes 3 Felder belegt, die in einer Reihe, in einer Spalte oder in der Diagonale liegen, gewinnt. Der Spielablauf ist in der nachfolgenden Abbildung beispielhaft aufgezeigt.

```
0 | 1 | 2
3 | X | 5
6 | 7 | 8

Spieler 0 am Zug.
Bitte Feldnummer eingeben (0 - 8): 3

0 | 1 | 2
3 | X | 0
6 | 7 | 8

Spieler X am Zug.
Bitte Feldnummer eingeben (0 - 8): |
```

Abbildung 1: Spielablauf von Tic-Tac-Toe

Wir verzichten für das Spiel auf eine fortgeschrittene grafische Ausgabe, sondern beschränken uns auf eine (farbige) Ausgabe des Spielfelds auf der Konsole.

Die einzelnen Felder des gesamten Spielfelds sind von 0 bis 8 durchnummeriert (zeilenweise von links oben nach rechts unten, siehe obige Abbildung). Die Spieler X und O können abwechselnd über die Eingabeaufforderung „Bitte Feldnummer eingeben:“ eintippen, welches Feld sie als nächstes mit ihrer Marke versehen möchten. Hierfür gibt der jeweilige Spieler die Ziffer eines freien Felds zwischen 0 und 8 ein. Das Spiel gibt dabei jeweils über die Konsolenausgabe „Spieler X/O am Zug“ an, welcher Spieler gerade am Zug ist. Nach jeder Eingabe wird das gesamte Spielfeld erneut unterhalb des vorherigen Spielzugs auf der Konsole ausgegeben und der jeweils andere Spieler ist am Zug.

Erforderliche Python-Kenntnisse

Für die Programmierung des Spiels sind folgende Python-Kenntnisse notwendig:

- Anwendung von Listen als Datentyp
- Definition und Verwendung von Funktionen
- Schreiben und Lesen von Dateien
- Schleifen als Kontrollstrukturen

Grundbausteine des Programms

Das fertige Spiel besteht aus mehreren Programmbausteinen, wovon einige bereits vorgefertigt sind. Die folgende Auflistung gibt eine Übersicht über alle Bausteine des Programms. Die ausstehenden Bausteine sind noch nicht implementiert und sollen im Rahmen des Projekts programmiert werden.

Nr.	Bausteine	Bereits implementiert
1	Spielfeld auf Konsole zeichnen	✓
2	Benutzereingabe anfordern	✓
3	Spieler wechseln	✓
4	Spielfeldstatus aktualisieren	Ausstehend
5	Sieg überprüfen	Ausstehend
6	Spielstand speichern	Ausstehend
7	Spielstand laden	Ausstehend
8	Hauptroutine des Spiels mit Datenstruktur festlegen	Ausstehend

Durchführung

Bestehendes Projekt in Visual Studio Code verwenden

Für die Programmierung des Spiels wird ein bestehendes Python-Projekt zur Verfügung gestellt. Dieses Projekt beinhaltet alle bereits implementierten Bausteine des Programms und stellt einen Rahmen für die Bausteine bereit, die noch zu implementieren sind.

- ▶ Öffnen Sie das Projekt, indem Sie zuerst VS Code öffnen und anschließend über die Navigation
A2 *File > Open Folder* das bereitgestellte Projekt *TicTacToe* von der Festplatte wählen.

- ▶ Machen Sie sich mit den einzelnen Programmbausteinen und Funktionen vertraut, indem sie
A3 diese im Projekt bzw. Code suchen und nachvollziehen.

Programmierung der Hauptroutine mit Datenstruktur

Jedes Spiel besitzt eine sogenannte Hauptroutine, durch welche der Spielablauf realisiert wird. In vielen Fällen wird die Hauptroutine eines Spiels durch eine Endlosschleife (`while(1)` oder `while True`) realisiert, die wiederkehrende Anweisungen ausführt und auf Benutzereingaben reagiert. Außerdem werden über die Hauptroutine Rückmeldungen an den Spieler geliefert, welcher mit neuen Eingaben auf diese reagieren kann. Dadurch ergibt sich eine Feedbackschleife zwischen dem Spiel und dem Spieler.

- ▶ Schreiben Sie im Programmbaustein *Hauptroutine (8)* den Rahmen für die Hauptroutine des
A4 Spiels, indem Sie eine Endlosschleife formulieren (zunächst ohne Anweisungen im Schleifenkörper).

- ▶ Erweitern Sie die Endlosschleife im Schleifenkörper um den Aufruf folgender Funktionen in
A5 der gegebenen Reihenfolge:
 - 1. `eingabe_anfordern()`
 - 2. `spielfeld_status_aktualisieren()`
 - 3. `spielfeld_zeichnen()`
 - 4. `spieler_wechseln()`
 - 5. `sieg_pruefen()`

Machen Sie sich mit dem Spielablauf und den Funktionsdefinitionen vertraut. Prüfen Sie dabei, welche Rückgabewerte und Übergabeparameter die Funktionen liefern bzw. erwarten.

Sie werden feststellen, dass die Funktion `spielfeld_status_aktualisieren()` einen Übergabeparameter für einen fehlerfreien Aufruf erwartet. Darum kümmern wir uns später.

Es stellt sich die Frage, wie man die Informationen bzw. den aktuellen Status des gesamten Spielfeldes in Python abbilden kann. Wichtig ist zu verstehen, dass es bei der Fragestellung nach einer geeigneten Datenstruktur zunächst nicht um die grafische Darstellung des Spielfelds auf der Konsole geht. Der Status des Spielfelds ist losgelöst von der grafischen Darstellung. Es geht an dieser Stelle deshalb um die Beantwortung der Frage, wie man den Status des Spielfeldes am besten im Python-Programm abbildet. Hierfür kommt z.B. eine Liste in folgender Form in Frage:

```
spielfeld = ["0", "1", "2", "3", "4", "5", "6", "7", "8"]
```

Listing 1: Eine Liste als Datenstruktur für die Abbildung des Spielfeldstatus

Die Liste repräsentiert den Status des gesamten Spielfelds. Sie hat die Länge 9 und enthält bei Spielbeginn die Ziffern von „0“ bis „8“ als einzelne Zeichenketten. Im Verlauf des Spiels werden die Ziffern durch die Zeichen „X“ und „O“ für den jeweiligen Spieler an der entsprechenden Stelle ersetzt. Die einzelnen Elemente der Liste entsprechen also folgenden Positionen auf dem Spielfeld:

0	1	2
3	4	5
6	7	8



Unbespielte Felder werden durch die Einträge „0“ bis „8“ in der Liste gekennzeichnet. Bespielte Felder werden durch „X“ bzw. „O“ an der entsprechenden Stelle in der Liste vermerkt.



Definieren Sie im Programmbaustein *Hauptroutine (8)* eine Liste gemäß Listing 1.

A6

Um festzuhalten, welcher Spieler („X“ oder „O“) gerade am Zug ist, bietet sich die Definition einer Variable an, welche diese Information festhält. Für unseren Fall erscheint es sinnvoll, dass diese Variable einen der folgenden Werte speichert:

- „X“ als Zeichenkette für den Spieler X
- „O“ als Zeichenkette für den Spieler O

- Definieren Sie im Programmbaustein *Hauptroutine (8)* eine Variable mit dem Namen
A7 `spieler_am_zug` und weisen Sie der Variable zunächst die Zeichenkette „X“ zu.

Die Benutzereingabe des jeweiligen Spieler (das zu bespielende Feld) soll in einer weiteren Variable festgehalten werden. Folgende Benutzereingaben sollen auf der Konsole vom Spieler entgegen genommen werden können:


- Die Ziffern 0 bis 8 für das zu bespielende Feld des Spielers
- Der Buchstabe „s“ zum Speichern des Spielstands in einer Textdatei
- Der Buchstabe „l“ zum Laden des Spielstands aus einer Textdatei

- Definieren Sie im Programmbaustein *Hauptroutine (8)* eine Variable mit dem Namen
A8 `spieler_eingabe` und weisen Sie der Variable zunächst eine leere Zeichenkette „“ zu.

Um das noch unfertige Programm zumindest lauffähig, also frei von Laufzeitfehlern, zu bekommen, müssen wir an den Funktionsaufruf von `spielfeld_status_aktualisieren()` ein Argument übergeben.

- Prüfen Sie nochmals die Funktionsdefinitionen und die Dokumentationen der beiden
A9 Funktionen `eingabe_anfordern()` und `spielfeld_status_aktualisieren()`. Überlegen Sie, welche Information an den Funktionsaufruf zur Aktualisierung des Spielfeldstatus übergeben werden muss, sodass das gewünschte Verhalten umgesetzt wird.

- Richtig: Speichern Sie den Rückgabewert des Funktionsaufrufs von `eingabe_anfordern()` in
A10 der Variable `spieler_eingabe`. Übergeben Sie diese Variable als Argument an den Funktionsaufruf von `spielfeld_status_aktualisieren()`.

- Starten Sie das noch unfertige Programm über die Schaltfläche  im oberen rechten Bereich
A11 und prüfen Sie die Angaben auf der Konsole. Es sollte ohne Fehler ausführbar sein, allerdings wird es noch eine sehr eingeschränkte Funktion vorweisen. Was macht das Programm? Was kann es noch nicht? Stimmt das mit dem überein, was Sie erwarten?

Die Konsole wird in VS Code nicht als separates Fenster angezeigt, sondern erscheint unter der Bezeichnung „Run“ in der unteren Bildhälfte.




Lassen Sie sich nicht vom Schlüsselwort `global` innerhalb der Funktionsdefinitionen ablenken. Dies bedeutet lediglich, dass wir damit *innerhalb* der Funktionen auf die Variablen zugreifen, welche wir in den vorherigen Schritten *außerhalb* der Funktionen (`global`) definiert haben.

Programmierung der Spielfeldstatusaktualisierung

Wie Sie vermutlich erkannt haben, ist es in der jetzigen Form des Programms noch nicht möglich eine Marke auf dem Spielfeld zu platzieren. Dies liegt daran, dass die Eingabe des aktuellen Spielers zwar entgegengenommen wird und an die Funktion `spielfeld_status_aktualisieren()` übergeben wird, allerdings fehlt im Funktionsrumpf dieser Funktion noch eine Anweisung, um den Status des Spielfelds in der Liste `spielfeld` tatsächlich auch zu aktualisieren.

Die Funktion `spielfeld_status_aktualisieren()` nimmt die auf Gültigkeit geprüfte Eingabe des aktuellen Spielers (das zu bespielende Feld) in Form eines Integerwerts (0 - 8) als Übergabeparameter mit dem Namen `index_eingabe` entgegen. Der Übergabeparameter kann direkt als Index für den Zugriff auf die Liste des Spielfelds verwendet werden. Die Liste `spielfeld` soll an der Stelle des übergebenen Index auf „X“ bzw. „O“ gesetzt werden. Der zu setzende Wert (ob „X“ oder „O“) befindet sich in der zuvor definierten Variable `spieler_am_zug`, auf welche auch innerhalb der Funktion zugegriffen werden kann.

► Implementieren Sie den Funktionskörper im Programmbaustein *Spielfeldstatus aktualisieren*
A12 (4), sodass die Funktion `spielfeld_status_aktualisieren()` das beschriebene Verhalten umsetzt.

► Starten Sie das immer noch unfertige Programm über die Schaltfläche  im oberen rechten
A13 Bereich und prüfen Sie die Angaben auf der Konsole. Es sollte ohne Fehler ausführbar sein, allerdings wird es weiterhin noch eine sehr eingeschränkte Funktion vorweisen. Was macht das Programm? Was kann es noch nicht? Stimmt das mit dem überein, was Sie erwarten?

Programmierung der Siegüberprüfung

Sie werden feststellen, dass der Spielablauf jetzt zwar grundsätzlich funktioniert, jedoch kann das Spiel noch weder gewonnen noch verloren werden. Außerdem kann der Spielstand noch nicht abgespeichert oder geladen werden. Die Programmierung dieser Programmbausteine ist der Inhalt der folgenden Abschnitte.

Siegbedingung: Das Spiel wird von einem Spieler gewonnen, wenn dieser entweder in einer Zeile, in einer Spalte oder in einer der Diagonalen alle seine Marken gesetzt hat.

Ein Spiel kann auch als unentschieden gewertet werden:

Bedingung für Unentschieden: Das Spiel wird als unentschieden gewertet, wenn keiner der Spieler bisher gewonnen hat und keine freien Felder mehr zum Bespielen zur Verfügung stehen.

Die Überprüfung dieser Bedingungen soll in der Funktion `sieg_pruefen()` erfolgen, deren Funktionskörper noch nicht implementiert ist. Die Funktion soll folgendes Verhalten umsetzen:

- Übergabeparameter: Keine
- Rückgabewert: True, falls Bedingung für Sieg oder Unentschieden zutrifft
- Rückgabewert: False, falls Bedingung für Sieg oder Unentschieden nicht zutrifft

Bei Sieg bzw. Unentschieden soll die Funktion die passende Ausgabe auf der Konsole ausgeben:

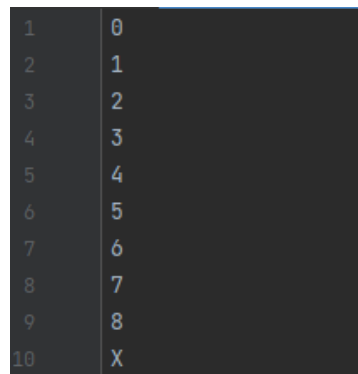
- „Spieler X hat das Spiel gewonnen!“
- „Spieler O hat das Spiel gewonnen!“
- „Das Spiel ist unentschieden!“

► Implementieren Sie den Funktionskörper im Programmbaustein *Sieg überprüfen* (5), sodass die
A14 Funktion `sieg_pruefen()` das beschriebene Verhalten umsetzt. Beginnen Sie mit der Implementierung der Siegbedingung und erweitern Sie die Funktion anschließend um die Bedingung für Unentschieden.

Programmierung der Speicherfunktion

Der aktuelle Spielstand soll über die Laufzeit des Programms hinaus in einer Textdatei gespeichert werden können. Das Speichern des Spielstands soll in der Funktion `spielstand_speichern()` erfolgen, deren Funktionskörper noch nicht implementiert ist. Die Funktion erhält keine Übergabeparameter und liefert keinen Rückgabewert.

Innerhalb der Funktion soll im aktuellen Ordner eine Textdatei `spielstand.txt` zum Schreiben angelegt werden. Sofern die Textdatei bereits vorhanden ist, soll diese überschrieben werden. Der Aufbau der Textdatei soll sich wie folgt gestalten:

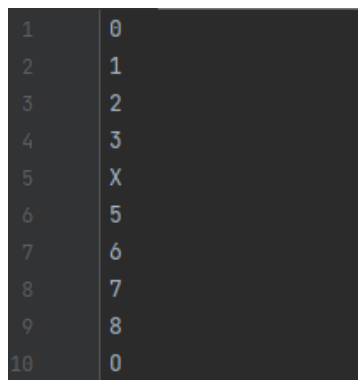


1	0
2	1
3	2
4	3
5	4
6	5
7	6
8	7
9	8
10	X

Abbildung 2: Textdatei `spielstand.txt` bei unbespieltem Spielfeld

In den Zeilen 1 – 9 sollen dabei die Elemente der Liste `spielfeld` gespeichert sein, welche den Spielfeldstatus enthält. In der obigen Abbildung ist das Spielfeld noch komplett unbespielt, da alle Ziffern von 0 bis 8 gespeichert wurden. In Zeile 10 soll der Spieler gespeichert werden, der zum Zeitpunkt des Speichern am Zug war; in obiger Abbildung war das Spieler X.

In nachfolgender Abbildung ist beispielhaft der Spielstand für ein bereits bespieltes Feld gezeigt. In diesem Fall wurde das Feld mit Index 4 (Zeile 5) vom Spieler X einmal bespielt. Spieler 0 war zum Zeitpunkt des Speicherns am Zug (Zeile 10). Achtung: Mit dem Auge sind die Buchstaben O (Zeile 10) und die Ziffer 0 (Zeile 1) schwer zu unterscheiden.



1	0
2	1
3	2
4	3
5	X
6	5
7	6
8	7
9	8
10	0

Abbildung 3: Textdatei `spielstand.txt` bei bespieltem Spielfeld

- Implementieren Sie den Funktionskörper im Programmbaustein *Spielstand speichern* (6),
A15 sodass die Funktion `spielstand_speichern()` den aktuellen Status des Spiels und den
aktuellen Spieler wie beschrieben in einer Textdatei `spielstand.txt` zeilenweise
abspeichert.

Programmierung der Ladefunktion


Die Ladefunktion `spielstand_laden()` soll den in der Textdatei `spielstand.txt` gespeicherten Status des Spielfelds und den zu diesem Zeitpunkt am Zug gewesenen Spieler wieder laden. Hierfür soll die Funktion die Textdatei zum Lesen öffnen und den Inhalt zeilenweise interpretieren. Über das Laden des Spielstands sollen die Variable `spieler_am_zug` und die Liste `spielfeld` aktualisiert werden.

- Implementieren Sie den Funktionskörper im Programmbaustein *Spielstand laden* (7),
A16 sodass die Funktion `spielstand_laden()` den gespeicherten Status des Spiels und den zu diesem Zeitpunkt am Zug gewesenen Spieler aus der Textdatei `spielstand.txt` einliest und die Variable `spieler_am_zug` und die Liste `spielfeld` entsprechend aktualisiert.



Beim zeilenweise Einlesen der Datei werden Sie vermutlich auf das Problem stoßen, dass die Zeilen in der Datei mit dem Zeilenumbruch „\n“ abgeschlossen sind. Sie können den Zeilenumbruch aus der eingelesenen Zeichenkette mit der Python-Stringfunktion `replace()` entfernen, indem Sie das Sonderzeichen „\n“ durch eine leere Zeichenkette ersetzen „“.

Was fehlt noch?

- Starten Sie das fast Programm über die Schaltfläche  im oberen rechten Bereich und prüfen
A17 Sie den Spielablauf. Stimmt alles? Was fehlt noch?

Richtig erkannt: Das Programm ist weiterhin in einer Endlosschleife gefangen. Wir benötigen als Letztes noch eine Abbruchbedingung für die Hauptroutine des Spiels. Die Abbruchbedingung ist erfüllt, wenn ein Spieler gewinnt oder das Spiel unentschieden gewertet wird. Diese Information liefert uns die Funktion `sieg_pruefen()` als Rückgabeparameter.

- Betten Sie den Aufruf der Funktion `sieg_pruefen()` in eine if-Abfrage ein (siehe A5). Falls
A18 der Funktionsaufruf `True` zurückliefert, soll die Hauptroutine mit der `break`-Anweisung beendet werden.

Geschafft! Herzlichen Glückwunsch! 😊