(Baby) Sokoban meistern

Prof. Dr. Frank Bauernöppel, 30.10.2024

Inhalt

Einleitung

Das Ziel dieses Tutorials ist es, algorithmisches Denken zu schulen und in konkrete elementare Anweisungen für einen Computer umzusetzen. Wir tun dies am Beispiel des Spieles Sokoban.

Das bekannte Sokoban Spiel (Sokoban – Wikipedia) hat viele knifflige Level und ist allgemein algorithmisch extrem herausfordernd (Sokoban is PSPACE-complete | Semantic Scholar). Wir begnügen uns daher mit einer stark vereinfachten Variante: Baby Sokoban, und wollen uns einer algorithmischen Lösung nähern, indem wir eine ganze Reihe einfacher Teilaufgaben formulieren und nach und nach lösen.

Beim Baby Sokoban gibt es in einer rechteckigen Welt aus quadratischen Kästchen:

- einen Spieler (Sokoban, rot),
- eine Box (grün) und
- ein Target (gelb).

Der Spieler muss die Box auf das Target schieben. Er kann in jedem Zug einen Schritt nach oben, unten, links, oder rechts gehen. Ist die Box in einem Nachbarfeld, kann er sie um ein Feld verschieben (push) indem er selbst auf das Feld der Box geht.

Wir können als Menschen fast alle Welten (Level) im Kopf lösen. Wir erkennen auch intuitiv, wann eine Welt unlösbar ist, was manchmal vorkommen kann. Aber können Sie auch einen Algorithmus formulieren, der die Aufgabe automatisch löst und der, idealerweise, für alle Welten funktioniert?

Wir benutzen die Programmiersprache Python dafür als Hilfsmittel und nur soweit es zum Formulieren der Algorithmen erforderlich ist. Die direkte Arbeit am Computer gibt

unmittelbares und objektives Feedback über den Code. Tiefergehende Programmierkenntnisse sind nicht erforderlich.

Wir verwenden das Codegerüst aus dem Moodle Kurs. Lesen Sie dazu die dort beigefügte Readme.md Datei. Wenn Sie Ihren Code einfügen, tun Sie dies immer in der Datei main.py und nur zwischen dem BEGIN und dem END. Für jede einzelne Aufgabe beginnen Sie von Neuem, ohne den davor eingefügten Code. Fertigen Sie für sich zum Nachschlagen Kopien der bereits gelösten Aufgaben an (main1.py, main2.py, ...).

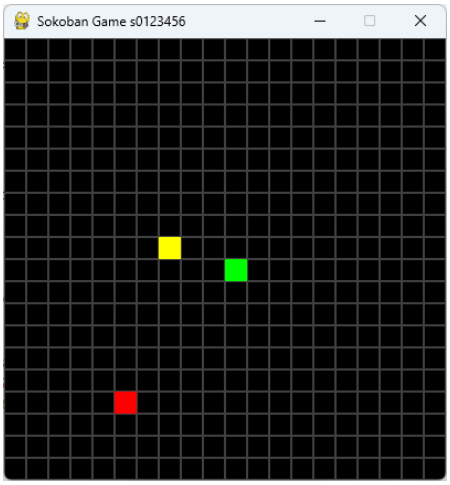


Abb 1: Die Referenzwelt

Die Kästchen sind, ähnlich wie eine Matrix mit Zeilennummer und Spaltennummer indiziert. Die Zählung beginnt mit Null. Das linke obere Kästchen hat den Index (0,0), das Kästchen darunter den Index (1,0), das Kästchen rechts neben der (0,0) den Index (0,1) und so weiter. Allgemein bezeichnen wir einen Index mit (x,y) wobei x die Spalten- und y die Zeilennummer ist. Als Koordinatensystem zeigt die x-Achse nach rechts und die y-Achse nach unten. Das ist in der Informatik (anders als in Mathematik und Physik) allgemein üblich und entspricht der (westlichen) Lese- und Schreibrichtung: von links nach rechts und von oben nach unten.

Wenn Sie den Sokoban bewegen (Tastatur oder später algorithmisch), wird in der Ausgabe als Protokoll die Zugfolge in einer einzigen Zeile ausgegeben, z.B. bei einer vollständigen Lösung:

```
using seed: s0123456 moves: RRRRRUUUUUURULLL.
```

Sokoban läuft einmal im Quadrat

Im Code steht als Vorlage bereits die Zugfolge

```
s.up()
s.left()
s.down()
s.right()
```

die den Sokoban einmal im Quadrat laufen lässt (Zugfolge ULDR). Jede Zeile ist eine Anweisung für den Sokoban, den entsprechenden Schritt zu tun. Diese Anweisungen sind in der Datei sokoban.py definiert auf die wir hier nicht näher eingehen werden.

Achten Sie darauf, dass der Code exakt so geschrieben und nicht eingerückt wird. Eingerückt werden muss und wird erst später, wenn wir eine Verzweigung im Code formulieren (if-Anweisung) oder eine Schleife (while-Anweisung) ausführen.

Position des Sokoban ausgeben

Wir können die x-Position an der der Sokoban steht im Code mit s.me.x abfragen. Dabei steht s für die Welt (Level) die im Code erzeugt wurde, me den Spieler und x seine x-Position. Mit der y-Position geht es ähnlich. Mit der print Anweisung geben Sie die Position aus:

Code:

```
print(s.me.x, s.me.y)
s.up()
print(s.me.x, s.me.y)
s.left()
print(s.me.x, s.me.y)
s.down()
print(s.me.x, s.me.y)
s.right()
print(s.me.x, s.me.y)
```

Ausgabe:

```
using seed: s0123456 moves: 5 16
U5 15
L4 15
D4 16
R5 16
```

In der Ausgabe steht abwechselnd die Position des Sokoban und der Zug der gemacht wurde. Die Startposition ist (5,16), dann wird Zug U (für s.up()) gemacht. Die Position des Sokoban ist nun (5,15), Zug L, (4,15) und so weiter.

Diese Ausgabe ist nicht besonders schön formatiert, genügt aber zum Zweck der Demonstration und bei Bedarf auch für Sie als Debughilfe um besser zu verstehen was der Algorithmus tut. Bei den Abgaben darf sie nicht verwendet werden, so dass die gesamte Ausgabe, wie anfangs, in einer einzigen Zeile steht.

Sokoban läuft zum linken Rand

Wir nutzen die x-Position um zu entscheiden ob der Sokoban bereits am linken Rand steht (s.me.x == 0) oder nicht (s.me.x > 0). Die == Bedingung ist genau dann erfüllt (wahr), wenn beide Seiten gleich sind. Wir formulieren zunächst in Textform, was der Algorithmus tun soll:

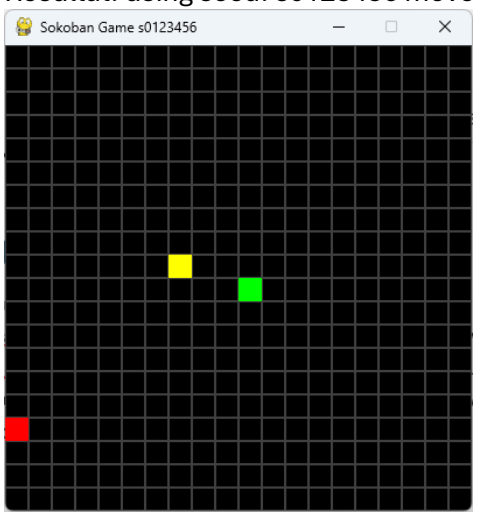
„Solange der Sokoban nicht auf dem linken Rand steht, soll er ein Feld nach links gehen".

„Solange" ist eine Wiederholung (Schleife) die im Code als while formuliert wird. Der dem while untergeordnete Code ist „ein Feld nach links gehen". Dies soll wiederholt getan werden, solange wie die Bedingung erfüllt ist. Dies kann einmal, mehrmals oder auch gar nicht sein, je nach der Situation am Anfang der Schleife.

Code (achten Sie alle Details: den Doppelpunkt am Ende, die Einrückung,...)

```
while s.me.x > 0:
    s.left()
```

Resultat: using seed: s0123456 moves: LLLLL



Die Einrückung ist in Python zwingend notwendig. In C benutzt man Klammern und sollte trotzdem konsistent einrücken, damit der Code lesbar und wartbar bleibt.

Sokoban läuft endlos hin und her (Endlosschleife)

Code:

```
while s.me.x > 0:
    s.left()
    s.right()
```

Die Bedingung s.me.x > 0 ist zu Beginn der Schleife erfüllt, weil der Sokoban in der Referenzwelt anfangs nicht am linken Rand ist. Der Sokoban geht bei jedem Durchlaufen der Schleife abwechselnd einen Schritt nach links und dann wieder nach rechts (zurück). Die Bedingung ist also nach jedem Durchlauf der while Schleife wieder erfüllt. Die Schleife wird also nie beendet werden und der Sokoban läuft endlos hin und her. Sie müssen den Algorithmus „mit Gewalt“ von außen beenden indem Sie das Fenster schließen.

Reparatur der Endlosschleife

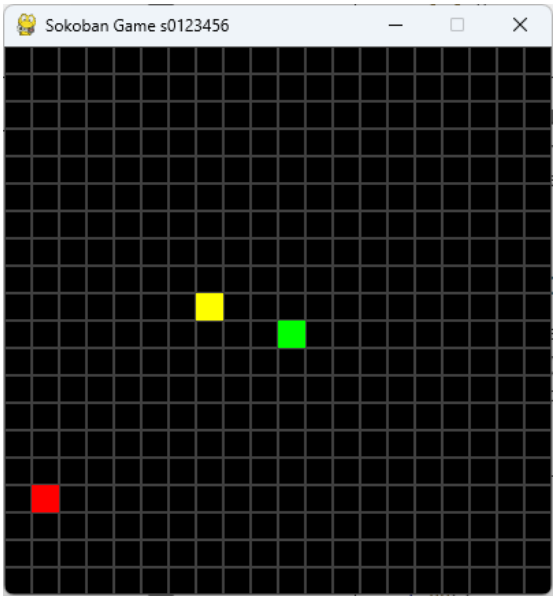
Code:

```
while s.me.x > 0:
    s.left()

s.right()
```

Der Sokoban wird bis zum linken Rand gehen, und danach einen Schritt nach rechts machen. Der Schritt nach rechts gehört nicht mehr zur Schleife, weil er nicht mehr eingerückt ist. Er wird danach ausgeführt, genau einmal. Die Leerzeile ist optional und wurde hier zur besseren Lesbarkeit eingefügt.

Resultat: using seed: s0123456 moves: LLLLLR



Sokoban läuft in die linke obere Ecke

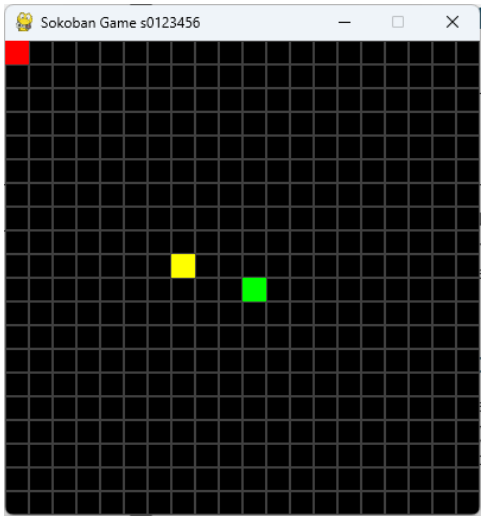
Der Sokoban soll erst an den linken Rand laufen und dann von dort an den oberen Rand. So kommt er auf das Feld (0,0) in der linken oberen Ecke. Das wird im Algorithmus mit zwei while-Schleifen formuliert, die hintereinander ausgeführt werden:

Code:

```
while s.me.x > 0:
    s.left()

while s.me.y > 0:
    s.up()
```

Resultat: using seed: s0123456 moves: LLLLLUUUUUUUUUUUUUUU



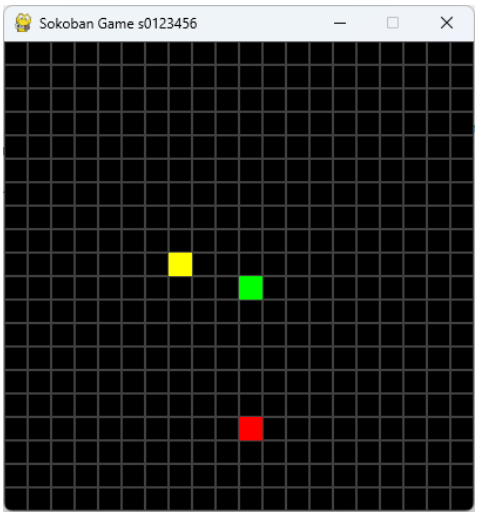
Es gibt auch andere Vorgehensweisen. Sie könnten z.B. die Reihenfolge der beiden while-Schleifen vertauschen. Ändert sich dadurch das Resultat? Ändert sich dadurch die Anzahl der Schritte (Züge) die der Sokoban insgesamt geht?

Sokoban läuft nach rechts zur Box

Die Position der Box lässt sich mit s.box.x und s.box.y ermitteln. Wir wissen, dass in der Referenzwelt der Sokoban links und unterhalb der Box startet. Wir können ihn folgendermaßen zur Spalte s.box.x laufen lassen in der auch die Box steht:

```
while s.me.x < s.box.x:
    s.right()
```

Resultat: using seed: s0123456 moves: RRRRR



Sokoban läuft zur Spalte der Box

In einer allgemeine Welt wissen wir nicht, ob der Sokoban anfangs in einer Spalte links oder rechts von der Spalte der Box ist oder sogar schon in derselben Spalte startet. Wir ändern die while-Bedingung so, dass wir die Spalten auf Gleichheit testen . Das geschieht wieder mit dem == Operator. Wir bleiben solange in der Schleife bis die Gleichheit erreicht ist. Im Schleifenkörper müssen wir nun eine Fallunterscheidung machen. Dazu dienen die if-Anweisungen:

Code:

```
while not s.me.x == s.box.x:
    # die Spalten von me und box unterscheiden sich

    if s.me.x < s.box.x:
        # die Bedingung der 1. if-Anweisung ist hier erfüllt:
        # Sokoban ist links von der Box
        # mache einen Schritt nach rechts
        s.right()

    if s.me.x > s.box.x:
        # die Bedingung der 2. if-Anweisung ist hier erfüllt:
        # Sokoban ist rechts von der Box
        # mache einen Schritt nach links
        s.left()
```

Beide if-Anweisungen sind eingerückt und gehören folglich komplett zum Körper der while-Anweisung. Die Leerzeile dient wieder nur zur Übersichtlichkeit des Codes. Alle Kommentare (nach dem # in grün bis zum jeweiligen Zeilenende) sind nur für den Menschen gedacht und werden vom Computer ignoriert.

Es gibt viele Varianten des Algorithmus die zum selben Ziel führen. Wenn Ihre Programmier-fertigkeiten voranschreiten, können Sie diese ausprobieren, z.B. eine if-Anweisung mit else Teil oder zwei while-Schleifen hintereinander ohne if oder Verwendung des != Operators anstelle von not. Wir wollen darauf hier nicht näher eingehen.

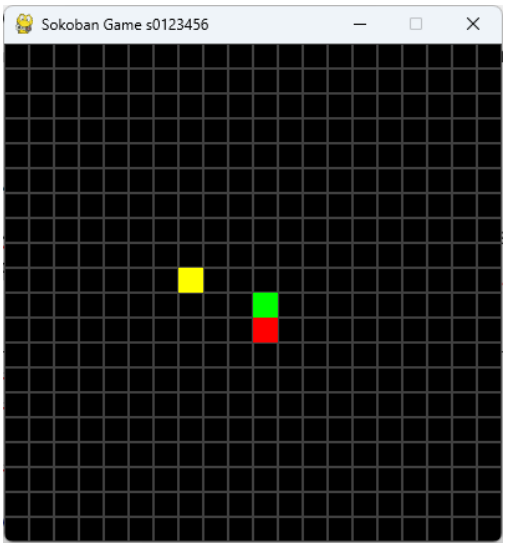
Sokoban stellt sich direkt unter die Box

Wir wissen nun, wie wir den Sokoban in die richtige Spalte bekommen. Nun soll er in der Referenzwelt nach oben gehen, solange bis er genau ein Feld unter der Box steht um diese anschließend schieben zu können. Dieses Feld hat die dieselbe x-Position wie die Box und als y-Position eins mehr als die Box. Denken Sie daran, dass die y-Werte nach unten hin wachsen. Der Code beginnt wie vorhin, nur ohne Kommentare und Leerzeilen.

```
while not s.me.x == s.box.x:
    if s.me.x < s.box.x:
        s.right()
    if s.me.x > s.box.x:
        s.left()

while s.me.y > s.box.y+1:
    s.up()
```

Resultat: using seed: s0123456 moves: RRRRRUUUUU



Beachten Sie, dass Sie im allgemeinen Fall einer beliebigen Welt wieder mehr Fallunterscheidungen machen und behandeln müssten.

Sokoban schiebt die Box in die richtige Zeile

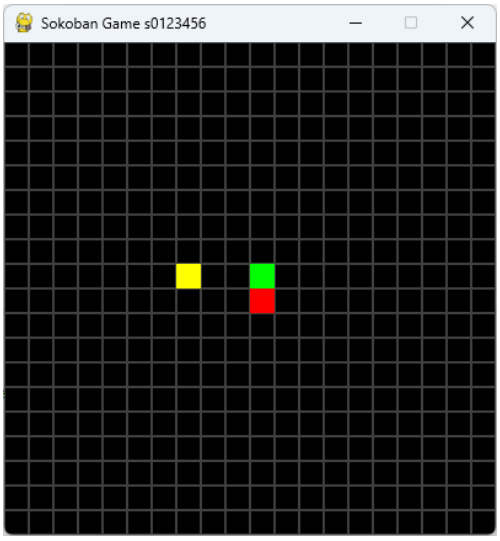
Jetzt kommt erstmalig das Target ins Spiel. In der Referenzwelt funktioniert der folgende Code:

```
while not s.me.x == s.box.x:
    if s.me.x < s.box.x:
        s.right()
    if s.me.x > s.box.x:
        s.left()

while s.me.y > s.box.y+1:
    s.up()

while s.box.y > s.target.y:
    s.up()
```

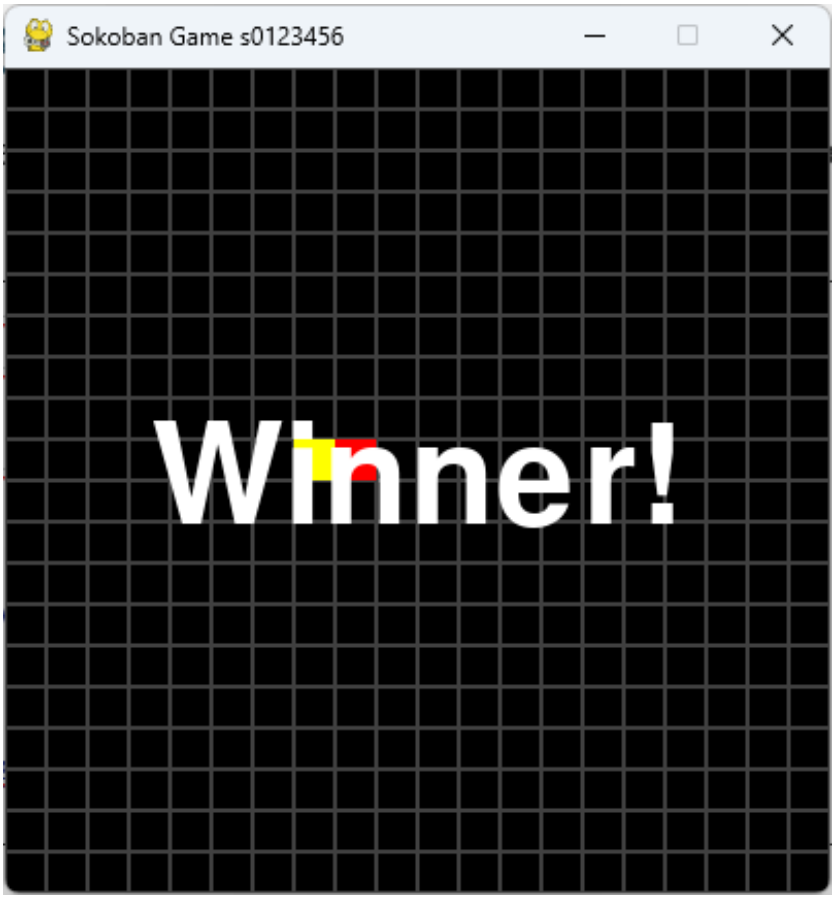
Resultat: using seed: s0123456 moves: RRRRRUUUUUU



Sokoban löst die Referenzwelt

Nun ahnen Sie bestimmt schon, wie Sie die Referenzwelt lösen können: Sie stellen den Sokoban rechts neben die Box und lassen ihn die Box solange nach links schieben, bis sie auf dem Target steht. Den Code dafür sollten Sie jetzt selbst schreiben können 😊

Resultat: using seed: s0123456 moves: RRRRRUUUUUURULLL.



Herzlichen Glückwunsch!

Wenn Sie selbstständig mehr machen wollen, sehen Sie sich die allgemeinen Sokoban Regeln an (Hindernisse, mehrere Boxen und Targets), lesen im Sokoban Wiki, oder schauen sich die Implementierung von Bruno Andrade näher an: xbandrade/sokoban-solver-generator: Sokoban puzzle generator and solver with BFS, A* and Dijkstra. Einige der dort verwendeten Algorithmen werden Ihnen im Studium noch begegnen.