Rubik’s Cube Simulation

Projekt Python TIF20A

Name: Fabio Freund
Matrikelnummer: 1097240
Kurs: TIF20A
Dozent: Dr. Stephan Laage-Witt

Projektbeschreibung:

Ziel des Projektes ist die Implementierung eines 3x3 Rubik’s Cube. Dabei soll der Benutzer alle Drehungen, die auf einem gewöhnlichen Rubik’s Cube möglich sind, eingeben können, sodass das Programm dann den aktuellen Zustand des Würfels anzeigt. Außerdem soll das Programm für einen beliebig verdrehten Würfel einen Lösungsweg berechnen können.

In-Scope:

- graphische Darstellung eines Rubik’s Cube
- korrekte Simulation von Drehungen am Würfel
- Generierung eines zufällig verdrehten, lösbaren Würfels
- Kontrolle, ob ein Zustand eines Würfels durch Drehungen lösbar ist (Nur 8,3% aller theoretisch möglichen Zustände sind durch Drehungen erreichbar)

Nice-to-have:

- Eingabe eines Zustandes um den dazugehörigen Würfel angezeigt zu bekommen
  - Eingabe durch einfärben des Würfels
  - Angabe eine Sequenz mit welcher der Würfel verdreht wird
- Finden eines Lösungsweges für beliebige Zustände

Out-of-Scope:

- Finden einer optimalen Lösung für beliebige Zustände
- Simulation von anderen Rubuik’s Cubes (z.B. 2x2, 4x4)
- 3-Dimensionale, graphische Darstellung des Rubik’s Cubes

Projektdokumentation:

Zielsystem:

Das Programm wurde in Python 3.9 auf einem Linux-System entwikelt und darauf angepasst. Alle Funktionen sind allerdings auch auf einem Windows10 System gegeben, lediglich das graphische Interface kann unter Windows etwas abweichen (z.B. Positionierung von Knöpfen). Daher befinden sich Screenshots des unter Linux entwickelten GUI im Anhang.

Externe Komponenten/Module:

Das Projekt benötigt keine externen Komponenten.
Das einzige externe Python-Modul, welches verwendet wird ist *numpy*, ansonsten werden nur python-interne Module benötigt.

Struktur des Codes:

cube.py
Enthält die Klassen *Piece* und *Cube*. Innerhalb dieser Klassen findet die Simulation des Rubik's Cubes statt. *Cube* erstellt dabei den Rubik's Cube als Objekt, welches wiederrum mehrerer *Piece*-Objekte erstellt, die die einzelnen Steine des Würfels darstellen. Diese Klassen bieten Funktionen um verschiedene Operationen wie z.B. Drehungen auf dem Würfel auszuführen.

solver.py
Enthält die Funktion *solve*, welche dafür verantwortlich ist, einen Lösungsweg für einen verdrehten Rubik's Cube zu berechnen.
Weiterhin befinden sich hier einige Hilfsfunktionen, wie zum Beispiel die Funktion *is_solved*, die überprüft, ob ein Würfel gelöst ist, oder die Funktion *solvable*, welche berechnet, ob ein gegebener Würfel überhaupt lösbar ist.

app.py
Dieses Modul ist für das graphische Interface Verantwortlich. Es enthält die Klasse *App*, die mithilfe von *tkinter* ein Fenster erzeugt, auf dem der Rubik's Cube und alle implementierten Funktionen graphisch dargestellt werden.
Das Modul ist das Grundgerüst des Programms, welches in sich die anderen Module aufruft. Über diese Datei wird auch das gesamte Programm gestartet.

constants.py
In diesem Modul werden einige Konstanten erzeugt, die in mehreren anderen Modulen benötigt werden. Die hauptsächliche Funktion dieses Moduls ist es, den Code zu gliedern und die anderen Dateien übersichtlich zu halten.

kociemba/
Der gesamte Code in diesem Ordner stammt **nicht** von mir, sondern ist von Herbert Kociemba unter der GPL-3.0 Lizenz bereitgestellter Code (https://github.com/hkociemba/RubiksCube-TwophaseSolver) und wurde von mir nur zu kleinen Teilen auf mein Projekt angepasst. Dieser Code ist dafür verantwortlich einen optimierten Lösungsweg für einen Rubik's Cube zu finden (für eine genauere Erläuterung siehe "Erzielte Resultate").

precalc/
Bei den Dateien in diesem Ordner handelt es sich um Heuristiken, die von den Modulen innerhalb des Ordners *kociemba* vorberrechnet werden.

Erziehlte Resultate:

Alle unter “in-scope” definierten Anforderungen wurden erfüllt und funktionieren einwandfrei.

Die “nice-to-have” Anforderung, einen Zustand für einen Rubik’s Cube eingeben zu können wurde ebenfalls erfüllt.

Die Anforderung, einen Lösungsweg für einen Rubik’s Cube zu berechnen gestaltete sich als schwierig. Hier war mein erster Ansatz, mit einer Breitensuche nach dem kürzesten Weg zum gelösten Zustand des Würfels zu suchen. Hier kamen allerdings schnell Performance-Probleme ins Spiel, sodass mit diesem Algorithmus in realistischer Zeit nur maximal 6-7 Ebenen an Drehungen überprüft werden konnten, was nur in einer verschwindend geringen Anzahl Fällen ausreichend ist, um den Würfel zu lösen.
Es gibt auch weitaus fortgeschrittenere Algorithmen zum lösen eines Rubik’s Cubes, welche allerdings sehr komplex sind und aufwändige Heuristiken verwenden, wodurch es mir nicht möglich war einen solchen Algorithmus im Rahmen des Projektes selbst zu implementieren. Daher entschloss ich mich dafür, den von Herbert Kociemba entwickelten und in Python progammierten (https://github.com/hkociemba/RubiksCube-TwophaseSolver) Algorithmus mit in mein Programm einzubinden, wobei ich nur einige kleine Änderungen vornehmen musste. Da dieser Algorithmus aber vor allem bei einfach zu lösenden Rubik’s Cube Zuständen sehr schlecht ist, besteht mein entgültiger Löse-Algorithmus nun aus einer Kombination von Breitensuche und des Kociemba-Algorithmus.

Die “out-of-scope” Anforderungen konnten leider nicht erfüllt werden.

Weitere Hinweise:

Bei der Darstellung des Rubik’s Cubes und der Drehungen habe ich mich an die offiziellen Vorgaben der “WorldCubeAssociation” gehalten:

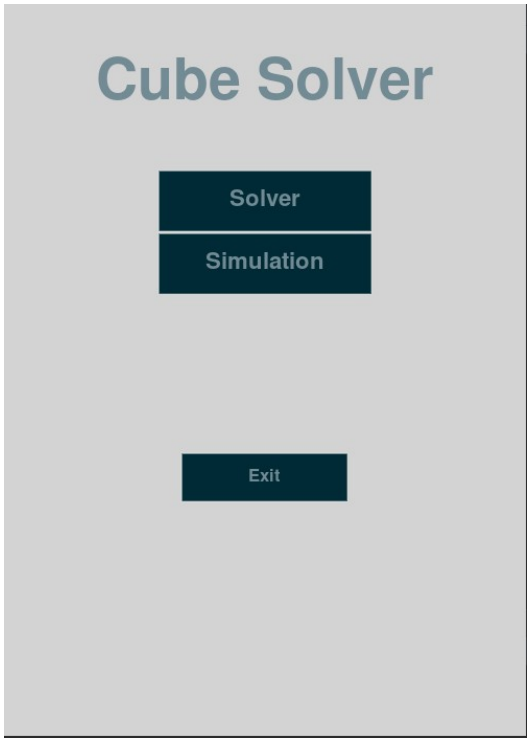
- https://www.worldcubeassociation.org/regulations/#4d
- https://www.worldcubeassociation.org/regulations/#12a1

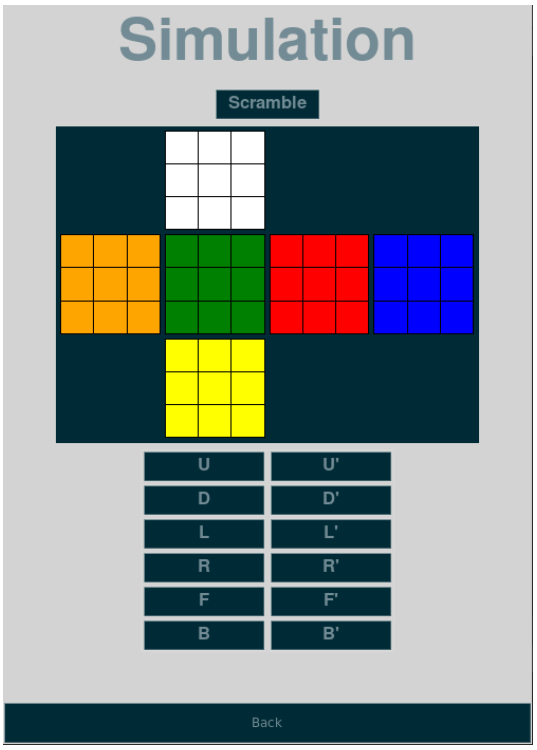
Bei der Benutzung des Programms wird die Kenntnis dieser Vorgaben vorausgesetzt.

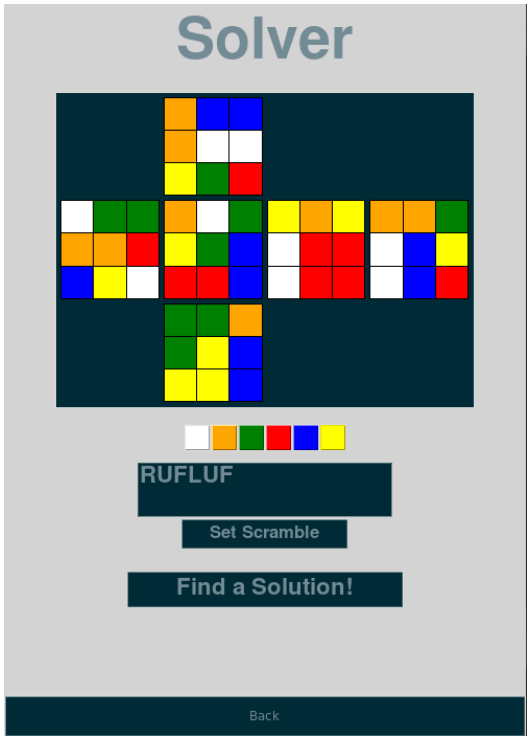
Beim Testen habe ich festgestellt, dass die Dateien im Ordner das die Dateien im Ordner *precalc* (vermutlich beim entpacken) beschädigt werden können, wodurch das Programm beim suchen einer Lösung zum Absturz kommt. Die genaue Ursache dafür konnte ich leider noch nicht finden. Allerdings kann man in diesem Fall einfach den Ordner *precalc* löschen, wodurch die Dateien beim nächsten Programmstart neu generiert werden (dies benötigt je nach Hardware ca. 20-30 Minuten).

Das Programm wird durch Ausführen der Datei *app.py* gestartet.

Anhang:


Cube Solver
Solver
Simulation
Exit


Simulation
Scramble
U
U'
D
D'
L
L'
R
R'
F
F'
B
B'
Back


Solver
RUFLUF
Set Scramble
Find a Solution!
Back