

## Fortgeschrittene Algorithmen

### Übung 11

#### Einführung in die Programmierung


#### Über dieses Übungsblatt

In diesem Übungsblatt beschäftigen wir uns mit etwas fortgeschrittenen Algorithmen.

## Aufgabe Newton'sche Physik 2

*Letzte Änderung: 12. July 2022, 12:00 Uhr*

**8 Punkte** — [im Detail](#)

Ansicht:  | 



Das zu simulierende Objekt

Wir möchten nun unsere Physiks simulation der letzten Woche erweitern. Eine Musterlösung der letzten Woche und animiertes gif der Lösung liegen in LMS bereit.



Wir möchten nun, dass wenn die Taste "u" gedrückt wird, unsere Kugel nach oben gestoßen wird; die Kugel bekommt also kurzzeitig Auftrieb, fällt dann aber weiter nach unten. Konkret wird also bei jedem Drücken der Taste "u" von der Geschwindigkeit ein sensibel gewählter, konstanter Wert abgezogen, so dass diese kurzzeitig negativ wird und das Objekt nach oben fällt. Da die Geschwindigkeit aber nach Aufgabenteil 4) im letzten Blatt sowieso konstant erhöht wird, sollte es nicht lange dauern bis die Geschwindigkeit wieder positiv ist und das Objekt wieder fällt.

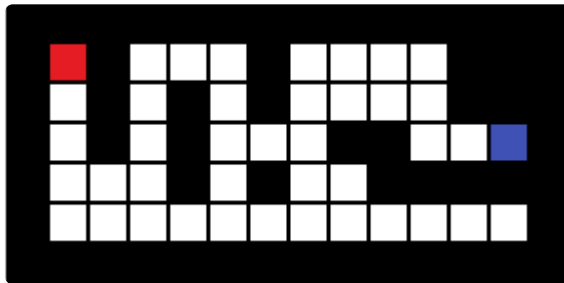
5) Programmieren Sie die oben beschriebene Erweiterung der Physiksimulation.

# Aufgabe Backtracking

Letzte Änderung: 14. July 2023, 12:00 Uhr

32 Punkte — [im Detail](#)

Ansicht:  | 



Das eingelesene Labyrinth, in Basic-IO angezeigt.

Das Ziel dieser Aufgabe ist es den Backtracking Algorithmus zu verwenden, um einen Weg durch ein Labyrinth zu finden. Dazu müssen wir zunächst das Labyrinth aus einer Datei einlesen. Das Labyrinth ist als .txt Datei folgendermaßen codiert:

```
" " == Begehbare Zelle (Durchsichtiger Pixel)
"E" == Eingang (Roter Pixel)
"A" == Ausgang (Blauer Pixel)
"." == Wand (Schwarzer Pixel)
```

0) Verwenden Sie die folgende Funktion um ein wie oben beschriebenes Labyrinth einzulesen. In LMS finden Sie eine Beispieldatei für ein Labyrinth zum Testen.

```
def read_map(filename):
    map = []
    # einlesen der map aus textdatei
    with open(filename, 'r') as f:
        for line in f:
            map.append([char for char in line][:-1])
    return map
```

Wir möchten nun das Labyrinth auf dem Bildschirm anzeigen. Dazu verwenden wir das Objektorientierte Interface von Basic-IO: für jedes Feld unseres Labyrinths erstellen wir einen großen Pixel mit `ioobj.Rectangle`; wir müssen dann nur noch die Pixel mit `ioobj.VerticalStack` bzw `ioobj.HorizontalStack` stapeln und können das erstellte Objekt dann einfach mit einem einzigen `bio.draw_object` Befehl anzeigen.

1) Erstellen Sie eine Funktion `print_map(map)`, dass das eingelesene Labyrinth wie auf der Abbildung zu sehen anzeigt.

2a) Verwenden Sie einen Backtracking Algorithmus um *einen* Pfad vom Eingang des Labyrinths zu dessen Ausgang zu finden. Implementieren Sie dazu eine rekursive Funktion `def find_way(map, path)`, wobei `path` der aktuelle Lösungspfad ist. Der Pfad soll aus einer Liste von Koordinaten (Tupeln) bestehen. Geben Sie den gefundenen Pfad auf der Basic-IO Konsole als Text aus.

**Tip 1:** Sie können sich am Pseudocode dieses [Wikipedia Artikels](#) oder an den Vorlesungsfolien orientieren.

**Tip 2:** Angenommen, wir besuchen gerade die Zelle mit Koordinaten  $(x, y)$ . Überlegen Sie sich zunächst, was alle *möglichen* Kandidaten für den nächsten Schritt sind. Überlegen Sie dann, wie Sie von den *möglichen* Kandidaten alle *validen* Kandidaten (begehbaren Zellen) extrahieren können.

**Tip 3:** Markieren Sie alle schon besuchten Punkte auf der Karte (ersetzen Sie z.B. " " -> "B" in `map`). Ein Kandidat ist ebenfalls nicht valide, falls er schon besucht wurde: so vermeiden Sie Endlosschleifen.

**Tip 4** Die Liste `path`, z.B. `path = [(2,3), (1,5), (1,9)]`, soll den aktuell gefundenen Pfad darstellen. Anfänglich enthält sie nur einen Punkt: den Eingang. Sie können mit `path.append((x,y))` einen neuen Punkt mit Koordinaten  $(x,y)$  zum Pfad hinzufügen und mit `del path[-1]` den letzten Punkt wieder entfernen, falls er zu keiner Lösung geführt hat.

**2b)** Animieren Sie nun ihre Lösungssuche: ergänzen Sie ihren Backtracking Code so, dass während der Ausführung alle besuchten Punkte hellblau eingefärbt werden. Wenn der entgültige Lösungspfad gefunden wurde, soll der gefunden Pfad vom Eingang zum Ausgang schrittweise rot eingefärbt werden. In LMS finden Sie ein animiertes .gif der Lösung.

**Hinweis:** Verwenden Sie `sleep` Befehle damit die Animation nicht zu schnell abläuft.