Informatik I: Einführung in die Programmierung

Prof. Dr. Bernhard Nebel
Tim Schulte, Thorsten Engesser
Wintersemester 2017/2018

Universität Freiburg
Institut für Informatik

Übungsblatt 8

Abgabe: Freitag, 14. Dezember 2017, 20:00 Uhr

Wichtige Hinweise: Zur Bearbeitung der Übungsaufgaben legen Sie bitte ein neues Unterverzeichnis `sheet08` im Wurzelverzeichnis Ihrer Arbeitskopie des SVN-Repositories an. Ihre Lösungen werden dann in Dateien in diesem Unterverzeichnis erwartet. Beachten Sie bitte bei allen Aufgaben die *Hinweise zur Bearbeitung der Übungsaufgaben* unter der folgenden URL:

`http://gki.informatik.uni-freiburg.de/teaching/ws1718/info1/guide/hinweise.html`

Bewertet wird bei allen Aufgaben die letzte Version, die zur Deadline des Übungsblattes auf dem SVN-Server eingereicht ist.

Aufgabe 8.1 (Brainf*ck; Dateien: `bf.py`, `spam.b`; Punkte: 4+4)

In dieser Aufgabe soll die Sprache Brainf*ck um ein zusätzliches Sprachelement erweitert werden. Wir wollen eine einfache Möglichkeit zur bedingten Ausführung von Code realisieren, nämlich eine Verzweigung ohne *else*-Zweig. Dafür sollen die neuen Zeichen `{` und `}` eingeführt werden. Idee dabei ist, dass der Code in den geschweiften Klammern ausgeführt wird, *falls* bei Betreten der Verzweigung der Wert in der aktuellen Zelle gleich 0 ist; andernfalls wird der geklammerte Code übersprungen.
Genauer besitzen die beiden neuen Zeichen folgende Bedeutung:

- `{`: Falls der Wert in der aktuellen Zelle gleich 0 ist (`data[ptr] == 0`), dann setze die Ausführung mit dem Befehl nach der öffnenden Klammer fort. Andernfalls, springe zum Befehl nach der zugehörigen schließenden Klammer.
- `}`: Fahre mit dem Befehl nach der Klammer fort.

(a) Laden Sie den Brainf*ck-Interpreter von der Webseite der Vorlesung herunter und erweitern Sie diesen um das eben beschriebene Sprachelement.

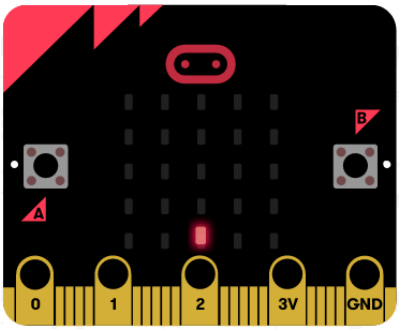
(b) Schreiben Sie ein Brainf*ck-Programm, das für einen beliebigen Eingabestring überprüft, ob dieser mit der Buchstabenfolge `HAM` (in Großbuchstaben) beginnt. Ist das der Fall, soll das Programm `KK` ausgeben. Andernfalls soll Ihr Programm nichts ausgeben. Greifen Sie hierbei auf das zuvor implementierte Sprachelement zurück.

Aufgabe 8.2 (Micro Bit: Snake Part I; Dateien: `snake.py`; Punkte: 4+4+4)

In dieser Aufgabe wollen wir die Grundlage für ein *Snake*[1]-ähnliches Spiel auf dem Micro Bit implementieren. Dieses werden wir dann auf dem nächsten Übungsblatt vervollständigen. Im ersten Schritt soll unser Programm zunächst einen einzelnen Punkt auf der LED-Matrix des Micro Bits abbilden. Dieser Punkt repräsentiert den Spieler, welcher sich kontinuierlich in eine Richtung bewegen soll. Dabei sollen Sie die Bewegungsrichtung des Spielers durch Drücken des A- bzw. B-Knopfes verändern können. Laden Sie zunächst das Template `snake.py` von der Kurswebseite herunter und machen Sie sich mit der allgemeinen Funktionsweise des Programms vertraut.

[1] `https://de.wikipedia.org/wiki/Snake_(Computerspiel)`

(a) Implementieren Sie die Funktion `render(player)`. Diese erhält die Position des Spielers (ein Tupel mit x,y-Koordinaten) als Argument und gibt ein `Image`-Objekt zurück. Die Pixelhelligkeit im erzeugten `Image`-Objekt soll hierbei an der Koordinate `player` den Wert `PLAYER_BRIGHTNESS` haben, während alle anderen Pixel die Helligkeit 0 haben sollen. Sie können Ihre Funktion testen, indem Sie den Main-Loop vorübergehend auskommentieren und Ihre Funktion in der REPL des mu-Editors mit `display.show(render((x, y)))` für beliebige $0 \leq x, y \leq 4$ aufrufen. Für den initialen Wert von `player` sollte das Programm die folgende LED-Ausgabe erzeugen:



Informationen zu den `Image` und `Display` Modulen finden Sie unter folgenden URLs:
`http://microbit-micropython.readthedocs.io/en/latest/image.html`
`http://microbit-micropython.readthedocs.io/en/latest/display.html`

(b) Implementieren Sie die Funktion `update(player, direction)`. Diese erhält die aktuelle Position des Spielers (`player`) und dessen Ausrichtung (`direction`) und gibt ein Tupel zurück, welches die neue Position des Spielers repräsentiert. Es gibt vier mögliche Richtungen, *Norden*, *Süden*, *Westen* und *Osten*, welche jeweils durch die Strings `'N'`, `'S'`, `'W'` und `'E'` repräsentiert werden. Der Spieler soll sich bei jedem Aufruf von `update` eine Einheit (Pixel) in die vorgegebene Richtung bewegen. Berücksichtigen Sie dabei, dass sich die Spielerposition niemals außerhalb des Spielfelds (der LED Matrix) befinden kann. Betrachten Sie das folgende Beispiel:

```
>>> player = (2, 4)       # der Spieler ist in der linken unteren Ecke
>>> direction = 'N'       # und nach Norden (oben) ausgerichtet
>>> update(player, direction)
(2, 3)  # neue Position des Spielers
>>> update((0, 0), 'W')
(0, 0)  # der Spieler kann den Spielfeldrand nicht verlassen,
        # seine Position ändert sich nicht
```

(c) Implementieren Sie die Funktion `process_input(direction)`. Diese erhält die Ausrichtung des Spielers und gibt dessen neue Ausrichtung zurück. Dazu soll die Funtion Abfragen, ob einer der beiden Knöpfe auf dem Micro Bit gedrückt wurde. Ist der B-Knopf gedrückt, soll die Ausrichtung des Spielers 90° **im Uhrzeigersinn** gedreht werden, ist der A-Knopf gedrückt, soll die Ausrichtung des Spielers 90° **gegen den Uhrzeigersinn** gedreht werden. Verwenden Sie die Funktionen `button_a.was_pressed()` und `button_b.was_pressed()` um die hierfür relevanten Eingaben abzufragen.

Informationen zum `Buttons` Modul finden Sie unter folgender URLs:
`http://microbit-micropython.readthedocs.io/en/latest/button.html`

Aufgabe 8.3 (Erfahrungen; Datei: `erfahrungen.txt`; Punkte: 2)

Legen Sie im Unterverzeichnis `sheet08` eine Textdatei `erfahrungen.txt` an. Notieren Sie in dieser Datei kurz Ihre Erfahrungen beim Bearbeiten der Übungsaufgaben (Probleme, benötigter Zeitaufwand nach Teilaufgabe, Bezug zur Vorlesung, Interessantes, etc.).