Einführung in die Programmierung
WS 2016/2017
Blatt 8

Alex Hirsch, Simon Hangl, Sebastian Stabinger

2017–01–10

- Abgabe bis spätestens Montag 21:59 über OLAT (`https://lms.uibk.ac.at/olat/dmz/`).
- Bereiten Sie jede Aufgabe so vor, dass Sie Ihre Lösung im Proseminar präsentieren können!
- Benennen Sie Ihre Abgabe nach folgendem Schema: *Gruppennummer*-*Nachname*-`blatt`*Übungsblattnummer*`.tar.gz` Wenn Sie also Max Mustermann heißen und Gruppe 1 besuchen, heißt die Datei von Übung 8: `1-mustermann-blatt8.tar.gz`
- Compilieren Sie alle Programme mit den Optionen `-Wall -Werror -std=c99`

Feedback

Nutzen Sie die angebotenen Möglichkeiten, uns Feedback zu geben (eMail, Tutorium, Proseminar). Hier können Sie uns auf Probleme, notwendige Stoffwiederholungen, Unklarheiten, aber auch positive Dinge, die beibehalten werden sollten, hinweisen.

Testen und Dokumentation

Stellen Sie sicher, dass alle Lösungen fehlerfrei kompilieren. Testen Sie Ihre Lösungen ausführlich (z.B. auf falsche Eingaben, falsche Berechnungen, Sonderfälle) und dokumentieren Sie sie. Dies hilft Ihnen bei der Präsentation und uns beim Nachvollziehen Ihrer Entscheidungen. Lesen Sie die Aufgaben *vollständig* durch.

Aufgabe 1 (7 Punkte)

Implementieren Sie *Conway's Game of Life*[1].

Lassen Sie sich nicht von der langen Erklärung abschrecken.

Conway's Game of Life ist ein einfacher, *zweidimensionaler*, zellulärer Automat. Das gerasterte Spielfeld hat eine bestimmte Höhe und Breite. Jede Zelle dieses Spielfeldes kann einen von zwei möglichen Zuständen einnehmen und ist entweder *tot* oder *lebendig* und wird von 8 Nachbarn umschlossen.

Die Simulation wird schrittweise ausgeführt, wobei aus dem Spielfeld des vorangegangenen Schrittes ein neues berechnet wird. Für jede Zelle des alten Feldes wird ermittelt, welchen Zustand die Zelle an der selben Position im neuen Spielfeld einnehmen soll.

Dabei werden die folgenden vier Regeln verwendet:

- Eine tote Zelle mit genau drei lebenden Nachbarn wird in der Folgegeneration neu geboren.
- Lebende Zellen mit weniger als zwei lebenden Nachbarn sterben in der Folgegeneration an Einsamkeit.
- Eine lebende Zelle mit zwei oder drei lebenden Nachbarn bleibt in der Folgegeneration am Leben.
- Lebende Zellen mit mehr als drei lebenden Nachbarn sterben in der Folgegeneration an Überbevölkerung.

Wenn Sie das Spiel gerne vom Erfinder persönlich erklärt bekommen würden, können Sie auch das folgende (unterhaltsame) Video anschauen: Youtube - Does John Conway hate his Game of Life?.

Ihr Programm sollte folgende Implementierungsdetails beachten:

- Lesen sie die Größe des Spielfeldes, die *Anfangs-Befüllung (Prozent)*, als auch die Anzahl an zu simulierenden Schritten via Commandline-Parameter ein (nicht interaktiv). Beispiel:

  ```
  $ ./gol 300 200 33 50
  ```

  Mit diesem Aufruf soll ein Spielfeld der Größe 300×200 simuliert werden. Es soll anfangs zu 33% mit lebenden Zellen gefüllt sein. Weiters werden 50 Schritte simuliert.

- Werden zu wenige oder zu viele Parameter übergeben soll das program *Usage-Information* (ähnlich wie andere Tools auf der Commandline) ausgeben und mit `EXIT_FAILURE` abbrechen. Beispiel:

  ```
  $ ./gol 300
  Usage: .gol <width> <height> <fill-rate> <steps>
  ```

[1] Wikipedia - Conway's Game of Life (en)

- Verwenden Sie zwei 2-dimensionale Arrays. Eines für das Spielfeld des aktuellen Simulations-Schrittes, das andere für den nächsten Simulations-Schritt. Dank C99's *Variable Length Arrays* können Arrays mit unterschiedlicher Größe zur Laufzeit (ohne Dynamic Memory Management) angelegt werden. Beispiel:

```
int main(int argc, char* argv[]) {
    int width = atoi(argv[1]);
    int height = atoi(argv[2]);

    int array[height][width];

    /* ... */
}
```

- Das Spielfeld soll, entsprechend der angegebenen Füllrate, zufällig initialisiert werden.
- Behandeln Sie Zellen, welche außerhalb des Spielfeldes liegen (Rand) als *tot*.
- Generieren Sie für jeden Simulations-Schritt ein PBM Bild[2]. Anschließend können Sie die simulierten Schritte mit dem `convert` Tool zu einem animierten GIF weiterverarbeiten.

  `pbm_writer.c`[3] enthält eine Funktion zum erstellen solcher Bilder, sowie ein Beispiel und detailierte Informationen wie Sie `convert` aufrufen können. *Falls sie nur einen Ausdruck dieses Übungsblatts vor sich haben: Die Datei finden sie verlinkt im PDF.*

Wenn Sie alles richtig implementiert haben, können sich die folgenden Muster bilden: Wikipedia - Conway's Game of Life - Examples of patterns.

Beachten Sie:

- Ihre Implementierung sollte möglichst modular sein. Kapseln Sie zusammengehörende Logik in Funktionen.
- Verwenden Sie *keine* globalen Variablen.
- Organisieren Sie zusammengehörige Daten in `structs`.
- Versuchen Sie Ihren Code von Anfang an durchdacht zu strukturieren.

Hinweis: *Abgabe: 1-mustermann-a1.c*

[2] The PBM Format
[3] `https://gist.github.com/W4RH4WK/6eb277d5a7fee81621219adbd843cbdf`

Aufgabe 2 (7 Punkte)

In dieser Aufgabe sollen Sie lernen, ein etwas größeres Programm eigenständig zu planen und zu implementieren. Sollten Sie das Spiel `Schiffe versenken` noch nicht kennen, können Sie die Regeln online[4] nachlesen. Ihr Spiel muss mindestens eine einseitige Singleplayerfunktion unterstützen:

- Der Computer setzt nach Zufallsprinzip die Schiffe auf das Spielfeld.
- Der Spieler kann über die Konsole eine Koordinate beschießen, worauf auf dem Spielfeld angezeigt wird, ob sich an dieser Stelle ein Schiff befand, oder nicht.
  Hinweis: *Das aktualisierte Spielfeld können Sie einfach aktualisieren, indem Sie genügend Leerzeilen einfügen und das neue Spielfeld auf der Konsole zeichnen.*
- Der Computer muss `nicht` zurückschießen können.

Weitere 7 Bonuspunkte gibt es, falls Sie mindestens eine der folgenden Erweiterungen implementieren:

- Entwickeln Sie eine KI die gegen den Einzelspieler spielt und (im besten Fall) intelligent zurückschießt. Dazu müssen Sie auch unterstützen, dass der Spieler seine Schiffe auf dem Feld platzieren kann.
- Entwickeln Sie eine Variante für zwei Spieler. Dabei müssen Sie sicherstellen, dass der jeweils aktuell aktive Spieler nicht die Schiffe des Gegners sehen kann.

[4] `https://de.wikipedia.org/wiki/Schiffe_versenken`

Aufgabe 3 (7 Punkte)

Dies ist eine Bonus-Aufgabe!

Implementieren Sie ein kurzes Text-Adventure mit folgenden Kriterien: Der Spieler interagiert über die Commandline mit Ihrem Programm und kann über eine handvoll Befehle das Spiel steuern.

Typischerweise sind einfache Text-Adventure in Räume unterteilt, welche in einem Gitter angeordnet sind. Über die vier Himmelsrichtungen (`north`, `south`, `east`, `west`) kann der Spieler von Raum zu Raum navigieren.

Der simpelste Befehl ist `look`, welcher dem Spieler Informationen über den Raum mitteilt, in dem er sich gerade befindet. Betritt der Spieler einen neuen Raum wird ihm automatisch eine Beschreibung des Raumes gegeben. Er muss hierfür nicht extra `look` ausführen.

Weiters sollte es einen Befehl `help` geben, der alle möglichen Befehle inklusive einer kurzen Beschreibung auflistet.

Implementieren Sie als erstes das Grundgerüst um ein Spiel zu beginnen, und den Spieler durch ein paar Räume mit unterschiedlichen Sehenswürdigkeiten navigieren zu lassen. Anschließend lassen Sie Ihrer Fantasie freien Lauf und implementieren ein bis zwei Mechaniken. Beispiel:

- Inventar System
- Verbale Interaktion mit einem *Non-Player-Character (NPC)*
- Simples Kampfsystem

Hier ein Beispiel wie so ein Text-Adventure aussehen könnte.

```
$ ./awesome-quest

You wake up.

The room is illuminated by torches mounted on the walls.
Your eyes scan the room for anything interesting,
but it's empty.

There is a doorway to the north.

  > north

The room you enter is dark, you are unable to see anything.

You cannot determine whether there are other exits,
you can still go back south.

  > south

The room is illuminated by torches mounted on the walls.
```

```
Your eyes scan the room for anything interesting,
but it's empty.

There is a doorway to the north.

  > pickup torch

You aquire a torch from the wall.

  > north

The room you enter is dark, but your recently picked
up torch allows you to examine it.

You see a chest in the center of the room.

There is also a door to the west.

  > west

You try to open the door, it is locked.

  > open chest

The chest opens without further ado and reveals a key.

  > pickup key

You take the key from the chest and continue your journey.

  > items

1: torch      2: key

  > west

You unlock the door with your new key.

You enter the next room and encounter a dragon.
It bites your head off, you are dead.
   _____                          ____
  / ____|                        / __ \
 | |  __  __ _ _ __ ___   ___  | |  | |_   _____ _ __
 | | |_ |/ _` | '_ ` _ \ / _ \ | |  | \ \ / / _ \ '__|
 | |__| | (_| | | | | | |  __/ | |__| |\ V /  __/ |
  \_____|\__,_|_| |_| |_|\___|  \____/  \_/ \___|_|
```