Programmieren C: Zweidimensionale Arrays, Zufallszahlen: Game of Life

Klaus Kusche

Das "Game of Life" ist ein beliebtes theoretisches Spiel bei Informatikern, Mathematikern usw.. Es soll die Entwicklung der "Bevölkerung" einer primitiven Zellkultur simulieren. Die "Natur" ist stark vereinfacht:

- Die gesamte besiedelbare Fläche ist in ein *rechteckiges (zweidimensionales) Raster* eingeteilt, jedes Feld entspricht dem Platz für *ein* Lebewesen und ist entweder lebendig oder tot.
- Das Leben in jedem Feld wird *Generation für Generation* betrachtet und hängt nur von den acht Nachbarfeldern ab (links, rechts, oben, unten und die vier diagonalen Nachbarn):
    - Auf einem *toten* Feld, das *genau 3 lebende* Nachbarn hat, entsteht in der nächsten Generation Leben: Das sind die optimalen Lebens-Bedingungen.
    - Ein *lebendes* Feld lebt auch in der nächsten Generation weiter, wenn es *genau 2 oder 3 lebende* Nachbarn hat: Bei mehr stirbt es an Überbevölkerung, bei weniger an Einsamkeit.
- Da auf diese Weise berechnete Bevölkerungen normalerweise nach ein paar hundert Generationen *stabil* werden (d.h. es ändert sich nichts mehr), lassen wir noch neues Leben durch *zufällige Mutation* entstehen: In jeder Generation entsteht in einer toten Zelle mit einer Wahrscheinlichkeit von 1/*n* *neues Leben*. Schon eine Mutationsrate von 1/100000 reicht aus, um auf Dauer ein bisschen Bewegung in die Landschaft zu bringen.

In unserem Programm soll jeder Bildpunkt am Schirm genau einer Zelle entsprechen. Unser "Land" hat also 800 * 600 Felder. Neues Leben wird gelb angezeigt, überlebendes Leben wird rot angezeigt, tote Zellen bleiben schwarz.

Aufgerufen wird unser Programm mit dem *Mutationsfaktor* *n* (wenn *n* gleich 0 ist, gibt es keine Mutationen, sonst mit Wahrscheinlichkeit 1/*n*, d.h. im Schnitt alle *n* Felder). Das Programm läuft endlos, bis man es abbricht.

Für die Grafik verwenden wir die SDL, die du auf meinen Webseiten samt Installationsanleitung zum Download findest.

Du kannst die Fenstergröße **SDL_X_SIZE** und **SDL_Y_SIZE** in **`sdlinterf.h`** an deinen Computer anpassen, wenn das Fenster für deine Grafik-Auflösung zu klein / zu groß ist.

Details:

- Das Programm enthält *zwei zweidimensionale Arrays*, jeweils mit **SDL_X_SIZE * SDL_Y_SIZE** Elementen (d.h. mit zwei Indices 0 … (SDL_X_SIZE – 1) und 0 … (SDL_Y_SIZE – 1)).
    - Ein Array zeigt das *Leben pro Feld* an: 0 für tot und 1 für lebendig.
    - Im anderen Array berechnen wir in jeder Generation für jedes Feld die *Anzahl der lebenden Nachbarn*.

Da die Variablen aller laufenden Funktionen incl. **`main`** in Windows maximal 0,5 bis 4 MB belegen dürfen (je nach Compiler, Einstellung, …), diese Arrays aber rund 4 MB groß sind, dürfen wir die beiden Arrays *nicht in* **`main`** deklarieren: Mach die Arrays global, d.h. schreibe die beiden Array-Deklarationen *vor* **`main`** !

- Sorge dafür, dass dein Programm *bei jedem Lauf andere Zufallszahlen* berechnet.
- *Zu Beginn erzeugen wir zufällig Leben im Lebens-Array*: Wir gehen dazu einzeln über alle Elemente (geschachtelte Schleifen für den x- und y-Index!). Liegt ein Element *ganz am Rand* (x oder y gleich 0 oder gleich (SDL_X_SIZE – 1) bzw. (SDL_Y_SIZE – 1)), wird es auf 0 (*tot*) gesetzt, sonst mit *50 % Wahrscheinlichkeit zufällig auf 0 oder 1*.
- Dann kommt die *Endlos-Schleife über die Generationen*:
  Pro Durchlauf wird eine Generation berechnet und angezeigt.
  Im Detail passiert jedesmal folgendes:
  - Der interne Grafikspeicher wird auf "alles schwarz" gesetzt.
  - Dann wird Element für Element (wieder zwei geschachtelte Schleifen für x und y, *ohne* die Rand-Elemente!) die *Anzahl der lebenden Nachbarn* berechnet (*Summe der 8 benachbarten Elemente* im Lebens-Array)
    und im entsprechenden Element des Anzahl-Nachbarn-Arrays *gespeichert*.
  - Schließlich geht man das *Lebens-Array* nochmals Element für Element durch (nochmals zwei geschachtelte Schleifen für x und y, *ohne* Rand-Elemente!). Bei jedem Element muss man 3 Fälle unterscheiden:
    - Das Element *lebt* bisher (ist 1):
      - Es hat laut Anzahl-Array *2 oder 3 lebende* Nachbarn.
        Dann lebt es weiter: Am Lebens-Wert ändert sich nichts, aber wir zeigen an dieser Stelle einen *roten Punkt* an.
      - Sonst: Es *stirbt*. Der *Lebens-Wert* wird *auf 0* gesetzt,
        und angezeigt wird nichts.
    - Das Element ist *tot*, und es hat laut Anzahl-Array *3 lebende Nachbarn*, oder es hat Glück und wird (falls der Mutationsfaktor *n* größer 0 ist) *zufällig* mit einer *Wahrscheinlichkeit 1/n* durch Mutation *neu geboren*.

      Dann zeichnen wir einen *gelben Punkt* (farbtechnisch entsteht gelb aus rot plus grün) und setzen den *Lebens-Wert auf 1*.

      (Sonst müssen wir für tote Elemente nichts tun.)
  - Wenn wir alle Elemente neu berechnet haben,
    aktualisieren wir die Grafik-Anzeige.

Hinweise:

- Die Anzahl-der-Nachbarn-Berechnung muss für alle Elemente *vor* der Prüfung und Änderung der Lebens-Werte gemacht werden, denn die Anzahl muss noch für die *alte* Generation gerechnet werden, *bevor* Lebens-Werte der neuen Generation gespeichert werden. Man kann daher das Zusammenzählen der Nachbarn und das Berechnen des neuen Lebenswertes *nicht in einem Schritt* machen, d.h. die beiden Schleifen zum Durchlaufen der Anzahl-Elemente und der Lebens-Elemente *nicht zu einer einzigen Schleife* zusammenfassen.

- Alle Schleifen zur Berechnung der neuen Generation gehen nur von 1 bis (SDL_X_SIZE – 2) bzw. (SDL_Y_SIZE – 2): Für die *Zellen ganz am Rand* berechnen wir *keine Summe* (weil sie nicht 8 Nachbarn haben), und sie *bleiben immer tot* und am Bildschirm schwarz.
- Das *Berechnen der Anzahl der Nachbarn* eines einzelnen Elementes ist keine Schleife, sondern *eine einzige lange Rechnung*: Die Lebens-Werte der 8 Nachbarn werden direkt addiert (wenn x und y die Index-Werte des aktuellen Elementes sind, mit welchen Index-Werten greift man auf die acht Nachbarn zu?).
- Wie sieht das **`if`** für die *Mutationen* aus? Es muss prüfen, ob der Mutationsfaktor *n* überhaupt größer 0 ist, und wenn ja, eine Zufallszahl erzeugen und diese so prüfen, dass sich mit Wahrscheinlichkeit 1/*n* "wahr" ergibt.
- Die **`if`** werden deutlich leicher und kürzer, wenn du mehrere Bedingungen mit "**`&&`**" für "und" oder "**`||`**" für "oder" in ein **`if`** schreibst!