Modellierung und Simulation 2019/2020
Conway's Game of Life

Louis Donath, Dario Klepoch

Potsdam University

Zusammenfassung. Conway's Game of Life ist ein Automat... Hier kommt eine kurze Zusammenfassung des Projektes rein

1 Einfuehrung

'Conways game of life' (im folgenden: GoL)

2 Erklaerung unserer Implementierung

3 Implementierung

Bei der Betrachtung der Implementierung werden wir über die Implementierung der Randbedingungen sprechen. Weil wir mit unseren ersten Implementierung mit einer unzureichenden Performance hatten, werden wir im zweiten Teil über Performanceverbesserungen sprechen.

3.1 Randbedingungen

In unserer Implementierung des GoL haben wir drei unterschiedliche Randbedingungen implementiert. Eine Randbedingung sagt aus wie sich das Spiel verhält, wenn auf Zellen ausserhalb des Spielbrettes zugegriffen wird.
Absorbierende Randbedingung bedeutet das jede Zelle die ausserhalb des Spielbrettes ist als tot angenommen wird.
Wir haben die **periodische** und **reflektierende Randbedingungen** implementiert wie man den Schemata in Tabelle 1 entnehmen kann. Auf der linken Seite ist die periodische Randbedingung gezeigt und auf der rechten Seite die reflektierende. In der Mitte ist das eigentliche Spielbrett mit den Zahlen 1..9 dargestellt. Darum herum sind (gespiegelte) Kopien des Spielbrettes gezeigt. Die Zahlen $1'..9'$ symbolisieren Referenzen zu dem Spielbrett in der Mitte. Daher wenn die Zelle 1 lebt, leben auch alle Zellen 1'. Wenn sich nun eine Zelle auf den Rand zu bewegt wird sie bei der periodischen Randbedingung am Anfang des Spielbrettes neu erscheinen. Bei der reflektierende hingegen in die gleiche Zelle gespiegelt.

Tabelle 1. Darstellung der Randbedingungen

1' 2' 3'	1' 2' 3'	1' 2' 3'	9' 8' 7'	7' 8' 9'	7' 8' 9'
4' 5' 6'	4' 5' 6'	4' 5' 6'	6' 5' 4'	4' 5' 6'	4' 5' 6'
7' 8' 9'	7' 8' 9'	7' 8' 9'	3' 2' 1'	1' 2' 3'	3' 2' 1'
1' 2' 3'	**1 2 3**	1' 2' 3'	3' 2' 1'	**1 2 3**	3' 2' 1'
4' 5' 6'	**4 5 6**	4' 5' 6'	6' 5' 4'	**4 5 6**	6' 5' 4'
7' 8' 9'	**7 8 9**	7' 8' 9'	9' 8' 7'	**7 8 9**	9' 8' 7'
1' 2' 3'	1' 2' 3'	1' 2' 3'	9' 8' 7'	7' 8' 9'	9' 8' 7'
4' 5' 6'	4' 5' 6'	4' 5' 6'	6' 5' 4'	4' 5' 6'	6' 5' 4'
7' 8' 9'	7' 8' 9'	7' 8' 9'	3' 2' 1'	1' 2' 3'	3' 2' 1'

3.2 Performance

Nachbarn ermitteln

Partielle Updates Nach der ersten Implementierung des GoL war die Performance für grössere Spielfelder nicht ausreichend. Den grössten Einfluss dabei hatte das Zeichnen von den Zellen des Spiels. Daher wird bei unsere Implementierung bei der Berechnung einer neuen Generation eine Liste erstellt mit den Zellen welche sich, im Vergleich zur letzten Generation, verändert haben. Nur die Zellen welche sich in dieser Liste befinden werden neu gezeichnet und alle anderen Zellen bleiben gleich.

Numpyarray anstatt Pythonlist Um die Performance noch weiter zu steigern wurde für die Representation des Spielbrettes ein Numpyarray anstelle eines Pythonarrays gewählt. Ausserdem ist dieses Array ein Array von boolean. Daher wenn eine Zelle lebendig ist, ist der Wert True (1) sonst False (0).

Weitere Performancesteigerung sind durch sehr viele unterschiedliche Veränderungen möglich. Eine sehr grossen Performancesteigerung ist dadurch möglich einen effizientere Datenstruktur als ein (numpy-)array zu verwenden. In verschiedenen anderen Implementierungen des GoL wird hierfuer ein Quadtree benutzt. Ein Quadtree wird meistens dafür verwendet effizient 2-dimensionale Daten zu speichern [1]. Da das GoL auch 2D-Daten sind ist es ein perfekter Anwendungsbereich für einen Quadtree. Mit 'Haslife' wurde das GoL auf diese Weise implementiert [2].
Um die Performance noch weiter zu steigern ist es moeglich den Quadtree parallel aufzubauen. Hier kann entweder die CPU oder auch die GPU benutzt werden. In [3] wurden lineare Quadtrees verwendet um einen Quadtree vollstaendig auf der GPU aufzubauen.

3.3 Bedienung

Um eine möglichst geeignete Bedienung für die Software zu entwickeln, haben wir uns zunächst einen Überblick über alle nötigen Bedienelemente verschafft. Bevor das Spiel gestartet wird, müssen Eigenschaften wie Spielfeldgröße, Randbedingungen und Zustand des Spielfeldes (z.B. alle Zellen sind tot) festgelegt werden. Um eine benutzerfreundliche Bedingen zu ermöglichen haben, wir uns entschieden diese Eigenschaften über ein Menü einzustellen.

Hauptmenü (vor Spielstart) Mit dem ersten Menüpunkt „New board" ist es möglich ein neues Board zu erstellen und alle Eigenschaften für dieses festzulegen. Neben Randbedingungen, Anzahl an Pixeln in Horizontaler und Vertikaler Richtung, kann auch die Option „Play random" gewählt werden, um das Feld (mit den Gewählten Eigenschafen) zufällig mit lebendigen und toten Zellen auszufüllen. Der „Back" Button ermöglicht außerdem wieder in das Hauptmenü zurückzukehren.
Unter dem Menüpunkt „Load Board" können Spielfelder ausgewählt werden, die mit interessanten Beispielen starten. Auch hier kann zusätzlich die Randbedingung festgelegt werden. Außerdem ermöglicht die Option „Load from File" den Zustand des Spielfeldes aus einer Datei zu laden, die der Benutzer im Vorhinein abgespeichert hat. Der Menüpunkt „Options" ermöglicht die Auswahl verschiedener Musikstücke, die im Hintergrund des Spiels laufen. Mit der Option „Quit" kann das Programm geschlossen werden.

Menüleiste (während des Spiels) Wurde das Spiel über den Button „Play"/ "Play random" im Untermenü von „New Board" gestartet, oder unter „Load board" ein Beispiel ausgewählt, so startet der Spielmodus. Hier kann das Spielfeld angepasst werden, indem eine Zelle mit der linken Maustaste als lebendig, bzw. der rechten Maustaste als tot markiert wird.
Die Menüleiste im, unteren Bereich des Fensters, ermöglicht es das Spiel zu starten und zu pausieren (➤/ ■), die Geschwindigkeit zu erhöhen (+), oder zu verlangsamen (-), sowie einzelne schritte zu machen (➡). Außerdem kann der aktuelle Zustand des Spielfeldes über den „Save" Button als Datei gespeichert werden. Diese Datei kann später über das Hauptmenü ausgewählt werden, um das gespeicherte Spielfeld wieder zu laden. Der „Menu" Button beendet das aktuell laufende Spiel und bringt den Benutzer zurück in das Hauptmenü.

Literatur

1. `https://www.geeksforgeeks.org/quad-tree/`, letzter Zugriff: 27.3.2020
2. `https://en.wikipedia.org/wiki/Hashlife`, letzter Zugriff: 27.3.2020
3. Dupuy, Jonathan & Iehl, Jean-Claude & Poulin, Pierre. (2018). Quadtrees on the GPU. 10.1201/9781351052108-12.