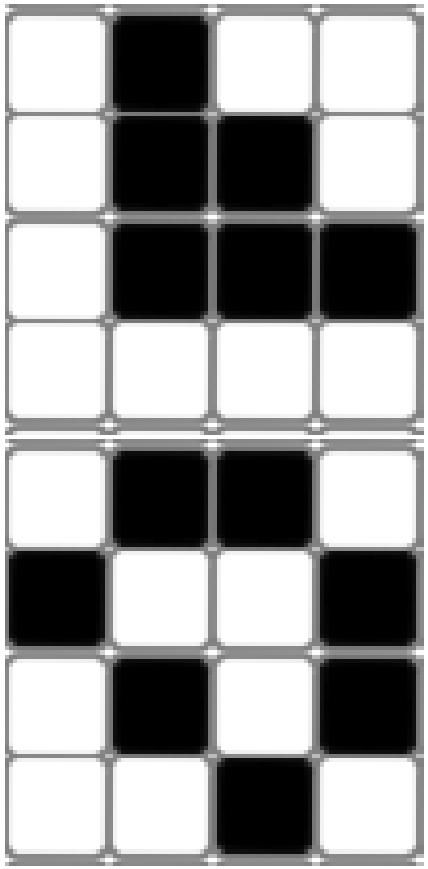
Conway's Game of Life

Valentino Bergamotto

31.01.2024

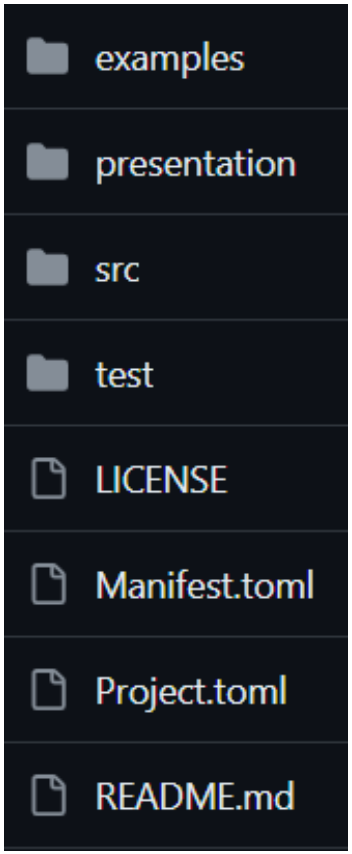
Einleitung

- Lebende Zellen mit weniger als 2 lebenden Nachbarn sterben.
- Lebende Zellen mit 2 oder 3 lebenden Nachbarn überleben.
- Lebende Zellen mit mehr als 3 lebenden Nachbarn sterben.
- Tote Zellen mit genau 3 lebenden Nachbarn werden lebendig.



Struktur

- examples enthält verschiedene Startkonfigurationen
- src enthält den Quellcode
- test enthält verschiedene Logiktests



src

- cell.jl, enthält eine struct "Cell" und verschiedene Funktionen, für diese
- gameboard.jl, enthält eine struct "Gameboard" und eine Funktion update_game()
- options.jl, enthält eine Funktion get_option(), um die Startkonfiguration zu bestimmen und ein paar Unterfunktionen, die verschiede Konfigurationen zurückgeben
- ConwaysGameOfLife.jl, führt alles wichtige aus und implementiert den letzten Schritt für die Interaktivität

Cell

- Cell speichert alle wichtigen Informationen, die eine Zelle braucht, sowie Funktionen, die diese verändern und aufrufen können
- Dazu gehören neighbours, alive, update und button
- Informationen, wie die Anzahl lebender Nachbarn werden über Hilfsfunktionen in der Datei bestimmt
- Die Hilfsfunktionen sind set_alive(), set_dead(), set_neighbours(cell, neighbour), set_button(cell, button), get_living_neighbours(cell), should_update(cell) und update(cell)

Gameboard

- Gameboard speichert eine Matrix von Zellen und implementiert die Visualisierung dieser
- Für die Visualisierung wurde GLMakie verwendet
- Die Initialisierung wird mit einer Matrix mit 0 und 1 ausgeführt, alle Zellen erhalten dann ihre relevanten Informationen
- Die Reihenfolge, in welcher man einer Zelle ihre Nachbarn zuweist, ist sehr wichtig. Deswegen sieht der Algorithmus dafür sehr komisch aus
- Mit der Funktion update_game(Gameboard), erhält man die nächste Iteration

Gameboard

```
function Gameboard(initial_state)
    fig = Figure()
    alivecol = "black"
    deadcol = "white"
    boxsize = 25
    gamesize = size(initial_state)
    temp_size = (gamesize[1]+2, gamesize[2]+2)
    current_state = fill(Cell(), temp_size)
    nextButton = Button(fig[gamesize[1]+1, gamesize[2]+1], label = "Next", width = 40, height = boxsize, buttoncolor = "white", strokecolor = "black")
    for i in 2:gamesize[1]+1
        for j in 2:gamesize[2]+1
            if initial_state[i-1,j-1] == 1
                current_state[i,j] = Cell()
                set_alive(current_state[i,j])
                set_button(current_state[i,j], Button(fig[i-1,j-1], label = "", width = boxsize, height = boxsize, buttoncolor = alivecol, strokecolor = "gray"))
            else
                current_state[i,j] = Cell()
                set_button(current_state[i,j], Button(fig[i-1,j-1], label = "", width = boxsize, height = boxsize, buttoncolor = deadcol, strokecolor = "gray"))
            end
            for k in 0:1
                for l in 0:2
                    if k == 1 && l > 0
                        continue
                    end
                    set_neighbours(current_state[i,j], current_state[i+k-1,j+l-1])
                end
            end
            set_neighbours(current_state[i,j-1], current_state[i,j])
            set_neighbours(current_state[i-1,j], current_state[i,j])
            set_neighbours(current_state[i-1,j-1], current_state[i,j])
            set_neighbours(current_state[i-1,j+1], current_state[i,j])
        end
    end
    map(i->rowsize!(fig.layout, i, boxsize), 1:gamesize[1])
    map(i->colsize!(fig.layout, i, boxsize), 1:gamesize[2])
    rowgap!(fig.layout, 0)
    colgap!(fig.layout, 0)
    new(initial_state, current_state, gamesize, fig, nextButton)
end
```

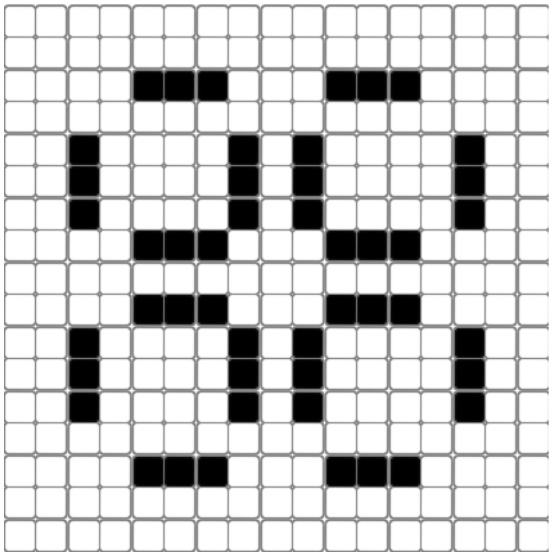
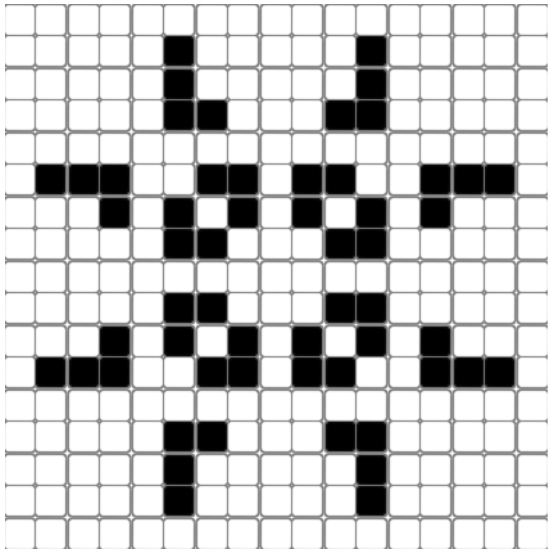
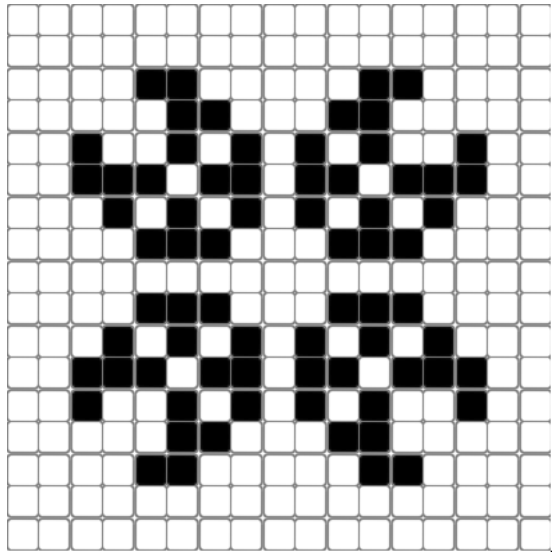
Festlegen von Nachbarn

```
for k in 0:1
    for l in 0:2
        if k == 1 && l > 0
            continue
        end
        set_neighbours(current_state[i,j], current_state[i+k-1,j+l-1])
    end
end
set_neighbours(current_state[i,j-1], current_state[i,j])
set_neighbours(current_state[i-1,j], current_state[i,j])
set_neighbours(current_state[i-1,j-1], current_state[i,j])
set_neighbours(current_state[i-1,j+1], current_state[i,j])
```

Resultate

- Ursprüngliche Initialisierung hat eine Komplexität von $O(y \cdot x)$ (40x40 braucht ca. 90 Sekunden)
- Danach ist jeder nächste Schritt auch für große Matrizen sehr schnell (für 40x40 sofort fertig)

Resultate

Fazit

- Mit einer anderen Visualisierungsmöglichkeit, kann man die Initialisierung vermutlich stark verbessern. Dabei geht aber die Interaktivität verloren.
- Conway's Game of Life kann mit vier sehr simplen Regeln, sehr komplexe Gebilde erschaffen
- Deswegen ist es auch noch nach über 50 Jahren eins der beliebtesten "Programmierspiele" und viele Menschen suchen nach neuen Gebilden

Vielen Dank für eure Aufmerksamkeit