Game of Life

Indholdsfortegnelse

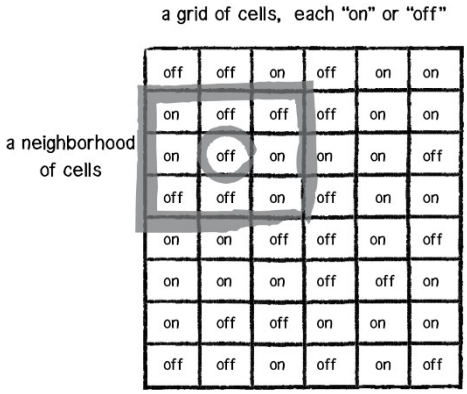
Kravspecifikation

a. Forklaring af principperne bag Game of Life.
b. Implementering af Game of Life med Classses i processing.

Cellular Automaton

Cellular automaton (ental) eller cellular automata (flertal) er et system af celler med følgende karakteristikker.

1. De lever på et "grid"
2. Hver celle har en game state. Det mest simple version af en cellular automaton har 2 hvor de typisk er "død" eller "i live"
3. Hver celle har nabolag. Dette kan være forskelligt fra hvert cellular automaton men det er typisk de celler som er ved siden af



(Fra Natur of Code af Daniel Shiftman)

Cellular automata blev opfundet af Stanisław Ulam og John von Neumann i 1940'erne.

Conway's Game of Life

Game of Life er et *Cellular Automaton* lavet af John Horton Conway i 1970. Det er et 0-spiller spil. Det fungerer på den måde at der er et *grid* med nogle celler på. Hver celle har 2 forskellige game states, død eller i live. Alle celler har 8 naboer. Disse naboer påvirker hvad der sker med cellerne. Hver gang der sker noget i spillet kan cellerne gøre 3 ting, dø, fødes eller intet. Dette bliver bestemt af nogle regler. Disse regler er:

1. En celle dør af ensomhed hvis der er mindre end 2 naboer i live.
2. En celle forbliver i live hvis der er 2 eller 3 naboer i live.
3. En celle dør af overpopulation hvis der er mere en 3 naboer i live
4. En ny celle fødes hvis der er præcis 3 naboer i live.

Biome test

Min version er lidt forskellig i forhold til Conways. Først og fremmest så kan cellerne som er i live gro, så der bliver flere. For det andet så har jeg lavet et nyt game state de kan være i, som jeg kalder for vand.

Design

Idéen med min version er at det skal sættes sammen med mit Autonomous Agents program, Hunting game. Med denne tanke valgte jeg farverne for mit Game of Life. Jeg valgte at de døde celler skulle være brune, så de lignede jord. Dem som har en game state på 1 er grønne for at ligne græs. Game state 2 er lidt mere lysegrøn dette er for at simulere at græsset gror, jeg komme ind på hvad game state 2 er senere når jeg forklarer reglerne. Game state 3 og 4 er begge relateret til Vand. 3 er normal vand og har en blå farve, hvor 4 er Vand som er i gang med at sprede sig, og fungerer lidt ligesom game state 2. Disse har en lyseblå farve.

Regler

Da både mit Vand og Græs kan sprede sig, så er mine regler også lidt forskellig til Conways Game of Life. Mine regler kan deles op i 2 forskellige kategorier, Græs og Vand. Mine regler for græs fungerer på denne måde:

regler for naboer:

Naboer bliver talt med cellernes game state. Så almindelig græs bliver talt som 1 nabo og almindelig vand bliver talt som 3 naboer.

regler for liv:

1. Hvis en celle ikke er Jord eller vand og har under 2 naboer, så dør den.
2. Hvis en celle ikke er Jord eller vand, har mere end 3 naboer, så bliver den til Græs.
3. Hvis en celle er Jord, har præcis 3 naboer så bliver den til enten til groende Græs, game state 2 eller normal Græs, game state 1. Dette udvælges tilfældigt
4. Hvis ingen af overstående regler gælder, forbliver de den det samme.

Reglerne for Vand fungerer på samme måde bare at den tæller hvor mange Vand naboer der er i stedet for at tælle naboernes game state.

Start

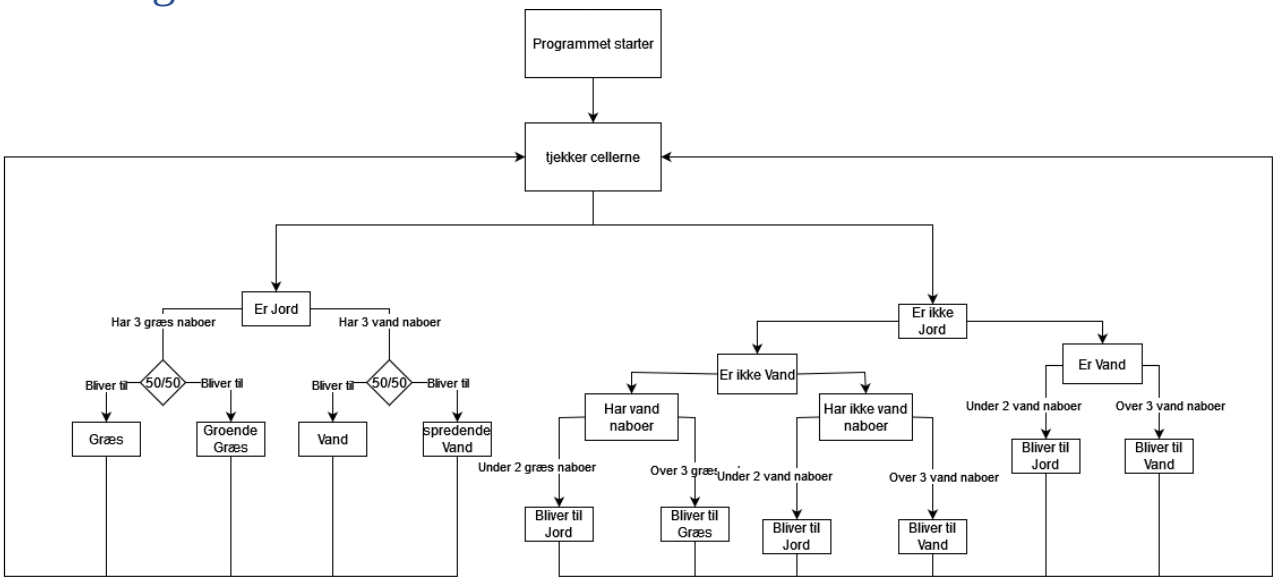
Når programmet starter har hver celle en procentdel chance for at blive til enten Vand, Græs eller Jord. De har en 20% chance for at blive til Vand. En 25% chance for at blive til Græs og en 55% chance for at blive til Jord. Hver celle har en større chance for at blive til Græs end Vand da jeg gerne vil have mere Græs end Vand. Vand har en 20% chance for at komme til, da jeg fandt at det var det bedste procent chance for at få mange mellemstore vandgrupper. Cellerne har en 55% chance for at blive til Jord, da jeg gerne vil have at både Vand og Græs skal kunne sprede sig.

Preload og Restart

Mit program har en form for preloader til at generere cellerne. Den fungerer på den måde at den kører en generation hver frame i de første 20 sekunder. Efter det køre den kun en generation igennem hver halve sekund. Dette gør den, da programmet senere skal sættes sammen med mit Autonomous Agents program. Jeg vil gerne have at ”prey’sne” fra mit hunting game skal have en chance for at kunne spise Græsset, så derfor køre den hvert halve sekund. Grunden til at preloaderen er aktiv i de første 20 sekunder er fordi jeg vil gerne have at Græsset skal sprede sig over hele skærmen, og i mine test har jeg fundet ud af det tager cirka 20 sekunder.

Programmet har også en restart funktion. Den fungerer på den måde, at hvis du klikker på skærmen så vil den restarte programmet og preloaderen. Jeg tilføjede denne funktion for at nemmere at teste hvordan den generer cellerne og deres generationer.

Flowdiagram



Dette flowdiagram er lavet for at vise reglerne. Den har en start med at programmet starter. Efter det vil den så tjekke cellerne. Her så viser den så hvad der sker hvis cellerne er Jord, ikke Jord og ikke Jord eller Vand. Og helt til sidst looper den tilbage til at tjekke cellerne.

Klassediagram

Cell
+ int columns - Hvor mange celler der er på x-aksen + int rows - Hvor mange celler der er på y-aksen + int[][] board - 2d array som holder cellerne + int size - størrelsen på cellerne
+ Cell() - Den bestemmer antallet af celler og bestemmer hvad cellerne er, jord, græs og vand, i starten af programmet. + void generate() - Tjekker naboer og kører cellernes regler. + void tegn() - Tegner cellerne og giver dem deres farver

Jeg har kun én class så derfor er mit klassediagram småt.

Kodeforklaring

Mine celler er skrevet som en class.

```
1  class Cell {

4      int[][] board;
```

Noget af det første jeg gør, er at lave et 2d array for at holde mine cellers game state. Det er et 2d array da cellerne har en x og en y position.

```
8   Cell() {
9     columns = width/size;
10    rows = height/size;
11
12    board = new int[columns][rows];
```

Efter det definerer jeg mit board inde i min constructor. Dette gør jeg med 3 variabler columns, rows og size. Columns er cellernes x værdi. Rows er cellernes y værdig og size er deres størrelse. Jeg vil gerne have at cellerne fylder hele skærmen så derfor når jeg giver columns og rows en værdig deler jeg width og height med deres størrelse.

```
13    for (int x = 0; x < columns; x++) {
14      for (int y = 0; y < rows; y++) {
15        int temp = int(random(100)); //
16        if (temp < 20) { //Has a 20% cha
17          board[x][y] = 3;
18        } else if (temp > 75) { // has a
19          board[x][y] = 1;
20        } else { // is dirt if not the o
21          board[x][y] = 0;
22        }
23      }
24    }
25  }
```

Efter det giver jeg dem deres game state, bestemmer om de er Vand, Græs eller Jord. For at gøre det give hver enkelt celle en game state. Dette kan man gøre med en dobbelt for-løkke som kører igennem x og y. For at bestemme deres game state har jeg en variabel som kan være et tal mellem 0 og 100 og an på den værdig den har giver den cellernes game state.

```
27  void generate() {
28    int[][] next = new int[columns][rows];
```

Det næste jeg gør at lave en funktion for at køre reglerne. Det første den gør er at lave et nyt 2d array. Dette gør det da det ville være problematisk vis jeg ændre direkte på mit board.

```
30    for (int x = 1; x < columns-1; x++) {
31      for (int y = 1; y < rows-1; y++) {
32        //resets neighbors and water neighbors
33        int neighbors = 0;
34        int water = 0;
```

Efter det køre jeg igennem end nu en dobbelt for-løkke for at køre igennem alle cellerne. Denne gang tæller jeg ikke de celler som er i kanten af skærmen. Dette skyldes at de ikke har 8 naboer så det kan lave problemer. Det første som sker for hver celle, er jeg sætter deres græs naboer og vand naboer til at være lig med 0.

```
36          for (int i = -1; i <= 1; i++) {
37            for (int j = -1; j <= 1; j++) {
38              neighbors += board[x+i][y+j]; //counts the gamestat
39              if (board[x+i][y+j] == 3 || board[x+i][y+j] == 4) {
40                water++;
41              }
42            }
43          }
```

Jeg kører igen igennem en dobbelt for-løkke men denne køre jeg igennem cellens naboer. Her så tæller jeg game staten for cellens naboer og tæller hvor mange vand naboer den har. Grunden til at den kører fra -1 til 1 er for at komme til alle naboer. -1 i den første for-løkke er den til venstre og 1 er den til højre. I for-løkke nummer 2 er -1 den over og 1 er den under. I begge er 0 den på samme x eller y position som dem selv.

```
44          neighbors -= board[x][y];
```

Efter det minusser jeg naboernes game state med cellens game state. Dette gør jeg fordi da jeg talte naboernes game state talte jeg også cellens egen game state.

```
45        //Rules:
46        //Grass
47        if      ((board[x][y] > 0) && (neighbors <  2) && (board[x][y] < 3)) next[x][y] = 0;
48        else if ((board[x][y] > 0) && (neighbors >  3) && (board[x][y] < 3)) next[x][y] = 1;
49        else if ((board[x][y] == 0) && (neighbors == 3) && (board[x][y] < 3)) next[x][y] = int(random(1, 3));
50
51        //Water
52        else if ((board[x][y] > 0) && (water <  2)) next[x][y] = 0;
53        else if ((board[x][y] > 0) && (water >  3)) next[x][y] = 3;
54        else if ((board[x][y] == 0) && (water == 3)) next[x][y] = int(random(3, 5));
55      }
56    }
```

Efter det køre jeg så reglerne som jeg har forklaret tidlige.

```
57      board = next;
```

Til sidst sætter jeg boardet til at være lig med mit midlertidige board.

```
60    void tegn() { //this could also be call
61    //goes through every celle to give ther
62      for (int i = 0; i < columns; i++) {
63        for (int j = 0; j < rows; j++) {
64          if (board[i][j] == 1) { //grass
65            fill(0, 255, 0);
```

Den sidste funktion jeg har er min displayfunktion. Her køre jeg igen igennem en dobbelt for-løkke for at køre igennem hver celle. Jeg giver dem så en bestemt farve an på hvilken game state de har.

```
75            stroke(0);
76
77            rect(i*size, j*size, size, size);
```

Det sidste jeg gør, er at jeg så tegner dem.

Dette var min kode forklaring af min class som mine celler er lavet med.

Testcase

I min testcase vil jeg vise mit program køre op til 30 sekunder 2 gange og blive restartet.

https://youtu.be/3iPWc2rDnNw

Konklusion

Jeg føler jeg fik lavet et fint program. Det er bygget videre på Conways Game of Life. Det er lavet så jeg nemt kan sætte det sammen med Autonomous Agents programmet.