Game of Life

Inledning

En *cellulär automat* är en dynamisk metod som beskriver hur komplicerade mönster kan uppstå från väldigt enkla regler. Fenomenet som kallas *emergens* har betydelse inom fysikaliska och biologiska system. Det finns mycket material att finna på internet för den intresserade.

Den engelske matematikern John Conway har utvecklat en välkänd cellulär automat som kallas ”Game of Life” som har att göra med liv och död hos celler i en rektangulär värld.

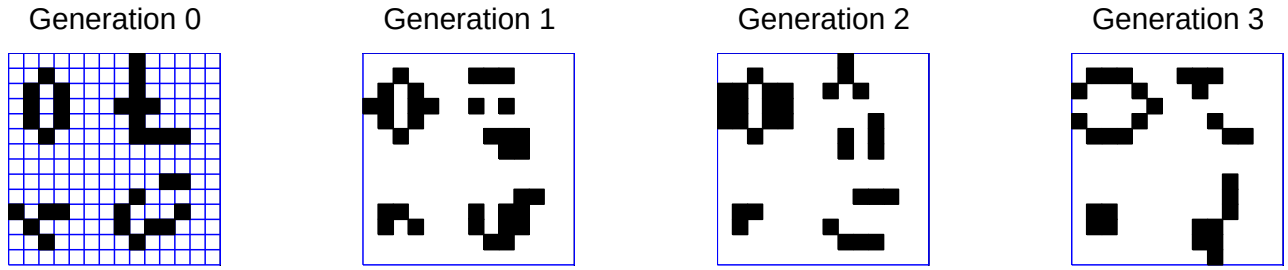
Uppgift

Detta är en variant av ”Game of Life” och syftet är att öva på programmering, inte att förstå cellulära automater.

Vi tänker oss en kvadratisk värld bestående av ett rutnät med säg 40×40 rutor. Utgående från en startgeneration, så bildas nästa generation enligt reglerna:

- En levande cell lever vidare om *två* eller *tre* av dess grannar är levande annars dör den.
- En tom cell blir levande om *tre* av cellens grannar är levande.

Med grannar till en viss cell avses celler som har kant eller hörn gemensamt med cellen. Generera ett rutnät där du kan markera en startgeneration. Följ t.ex. 50 generationer, lägg in en kort paus mellan uppritning av de olika generationerna.



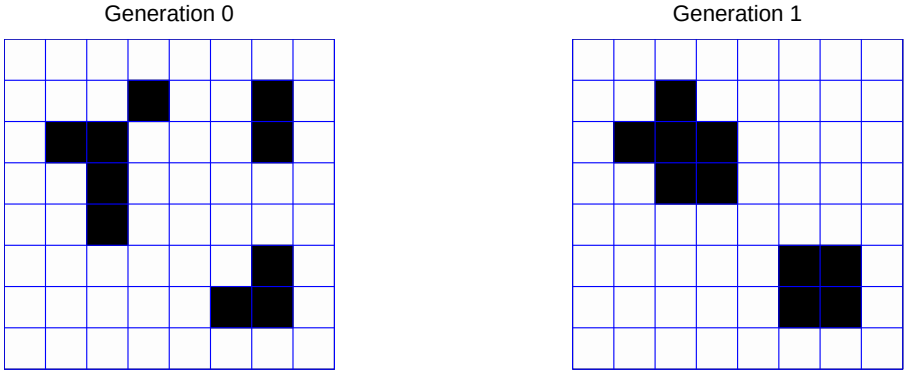
Skriv en funktionsfil som löser uppgiften på ett enkelt och bra sätt. Testa programmet på någon rolig startgeneration.

Förberedande övningar

Vi tittar lite extra på generationsväxlingen. Nästa generation bildas enligt reglerna:

- En levande cell lever vidare om *två* eller *tre* av dess grannar är levande annars dör den.
- En tom cell blir levande om *tre* av cellens grannar är levande.

Ta bilden nedan som hjälp för att tänka igenom vilka celler som förblir levande, var liv uppkommer och vilka celler som dör.



Om vi tänker oss världen som en spelplan, där vi markerat liv som ovan, så kan vi inte ändra några markeringar för att visa nästa generation förrän vi vet hur hela världen ser ut (vid nästa generation). Detta beror på att en cell är granne till sina grannar.

Här följer några övningar för att komma igång.

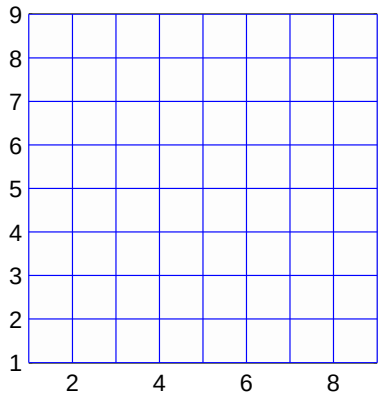
1. Låt $n = 40$. Rita en vit $n \times n$ kvadrat med ett rutnät på, som på bilden ovan. Använd `fill`, `plot` och `axis` på lämpligt sätt.
2. Bilda lämpliga matriser för att lagra information om var det finns liv. Det behövs en för nuvarande generation och en för nästa.
3. Gör en kommando-sekvens som gör det möjligt att markera levande celler, för att ge en startgeneration. Använd `ginput` och `fill`.
4. Gör en funktion som räknar en cells levande grannar. Använd `sum`.

Efter de förberedande övningarna är det dags att sätta ihop dem och göra färdigt funktionen för ”Game of Life”. En liten paus mellan generationsväxlingarna fås med `pause`, se hjälptexten.

Lösningar till förberedande övningar

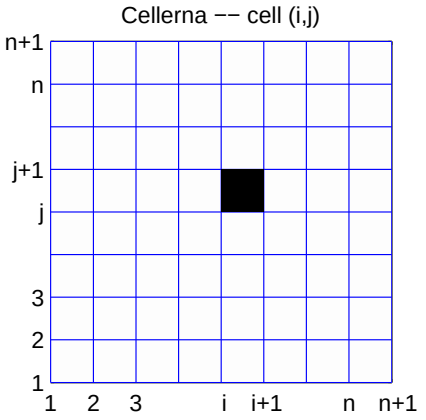
1. Vi ritar en bild av världen – cellerna

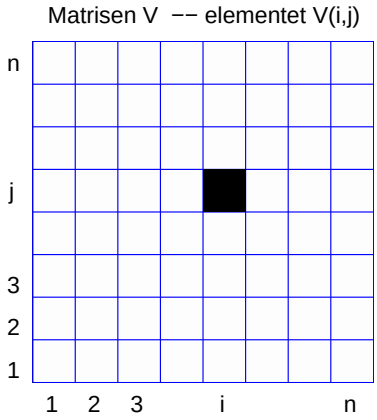
```
figure(1), clf
fill([1 n+1 n+1 1],[1 1 n+1 n+1],'w')
axis equal, axis([1 n+1 1 n+1])
hold on
for k=1:n+1
    plot([1 n+1],[k k])
    plot([k k],[1 n+1])
end
```



När vi är klara med hela programmet kommer vi lägga till `axis off` efter `axis equal` så att skalorna på axlarna (siffrorna) inte syns.

2. En matris `V` får sköta bokföringen av var det finns liv och inte. I bilden nedan till vänster har vi markerat liv i en cell, horisontell position `i` vertikal position `j`. I matrisen `V` låter vi element `V(i,j)` beskriva denna position. Vi låter värdet 1 motsvara liv och värdet 0 motsvara avsaknad av liv.



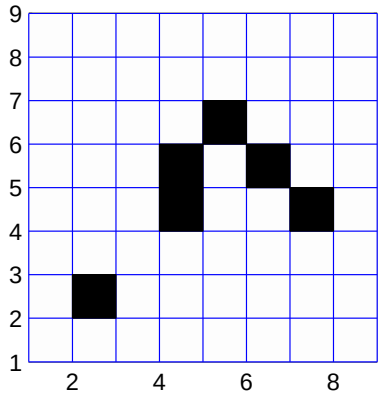


Vi behöver en matris för nuvarande generation och en för den nya. Med `zeros` får vi matriser fyllda med nollor (dvs. avsaknad av liv). Vi börjar ju med en öde värld.

```
V=zeros(n,n);
Vny=zeros(n,n);
```

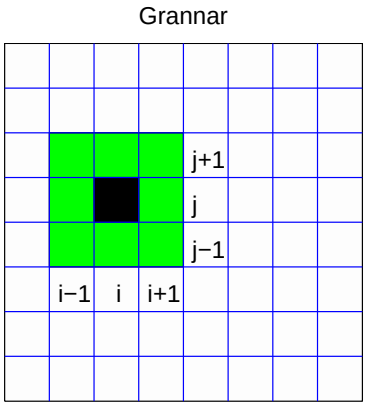
3. Trycker vi på vänster musknapp markerar vi liv i cellen annars avbryter vi.

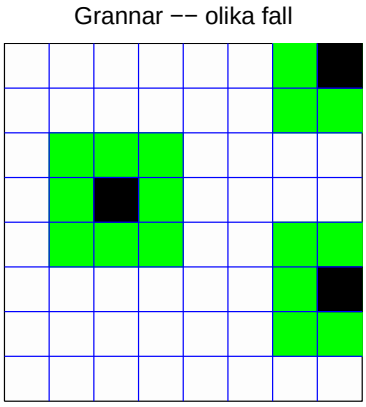
```
while 1
    [i,j,Knapp]=ginput(1);
    i=floor(i); j=floor(j);
    if Knapp==1
        V(i,j)=1;
        fill([i i+1 i+1 i],[j j j+1 j+1],'k')
    else
        break
    end
end
```



Samtidigt som vi fyller cellen med svart för att markera liv, sätter vi in värdet 1 på motsvarande plats i matrisen.

4. Vi skall räkna antal grannar. Antag att vi ser på cell `i,j`. Cellen är svart markerad i figuren och dess grannar är markerade med grönt. Om cellen är levande eller inte får vi veta genom att se på matriselemenet `V(i,j)`





Antag vi vill ta reda på hur många grannar till en cell `V(i,j)` som har liv. Om vår cell ligger i det inre, dvs. om alla dess grannar ligger i världen, så gäller att blocket `V(i-1:i+1,j-1:j+1)` beskriver vår cell och dess grannar och vi får antal levande grannar med

```
antal=sum(sum(V(i-1:i+1,j-1:j+1)))-V(i,j)
```

För att klara av fallen med att vår cell ligger längs en kant eller i ett hörn gör vi två indexvektorer `igrannar` och `jgrannar`. T.ex. om `i-1` ligger utanför till vänster om världen så skall vi bara gå till horisontell position 1. Vi tar därför gränsen åt vänster i horisontell riktning som `max(1,i-1)`. På motsvarande sätt för de övriga riktningarna. Med dessa indexgränser klarar vi alla fall och vi beräknar antal grannar med

```
igrannar=max(1,i-1):min(n,i+1)
jgrannar=max(1,j-1):min(n,j+1)
antal=sum(sum(V(igrannar,jgrannar)))-V(i,j)
```

Vi packeterar det hela i en funktion enligt

```
function antal=raeknagrannar(V,i,j,n)
    igrannar=max(1,i-1):min(n,i+1);
    jgrannar=max(1,j-1):min(n,j+1);
    antal=sum(sum(V(igrannar,jgrannar)))-V(i,j);
```

Överkurs. Vi kan byta ut `[i,j,Knapp]=ginput(1)`, som ger ett hårkors som markör, mot

`[i,j,Knapp]=min_ginput;`

som behåller pilen som markör. Detta är en egen variant av `ginput` för inläsning av en enda punkt.

```
function [h,v,knapp]=min_ginput
    fig=gcf; figure(gcf);
    keydown=waitforbuttonpress;
    switch lower(get(fig,'SelectionType'))
        case('normal')
            knapp=1;
        case('extend')
            knapp=2;
        case('alt')
            knapp=3;
    end
    pt=get(gca,'CurrentPoint'); h=pt(1,1); v=pt(1,2);
```

Vi behöver inte förstå alla detaljer, vi kan använda `min_ginput` ändå.