# Life

Kurt Nørmark
Aalborg Universitet

10. september 1997

**Resumé**

Dette skrift er et literate program for Life. Life simulerer generationer af “levende cellers” udvikling, med baggrund i tre simple regler for hvornår celler dør, og hvornår celler opstår. Programmet udvikles i Turbo Pascal 7.0.

# 1 Introduktion

Life er et fascinerende "spil", hvori man simulerer udviklingen af levende celler fra population til population.

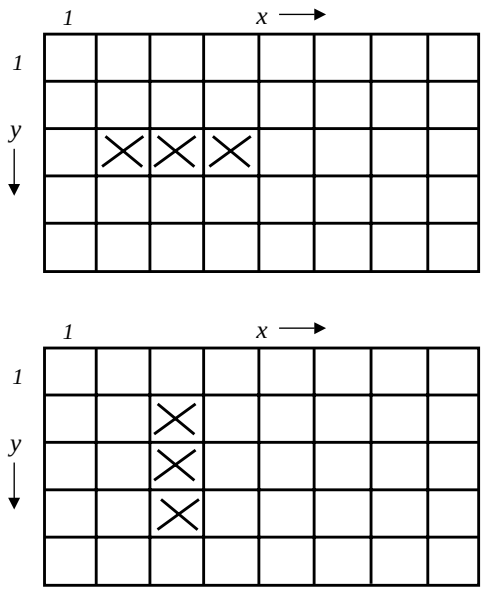
Her følger et “klip” fra et Computer spil, som forklarer lidt om historien af Life.

> The world first learned of John Conway’s Life rules when they were presented in Martin Gardner’s "Mathematical Games"column in the October 1970 issue of Scientific American. That and a follow-up article in February 1971 were enough to start amateur and not-so-amateur mathematicians scrambling to find new and interesting Life patterns and properties.
>
> Remember that this was before the dawn of personal computers, so much of the early work was done by hand on paper and on Go boards. Early questions concerned topics like the eventual fate of various initial patterns. The famous "R Pentomino"is a five-cell pattern that finally settles down to an oscillator after 1,103 generations. Such long-lived patterns are called Methuselahs. A seven-celled Methuselah called Acorn lives for over 5,000 generations.
>
> Other questions surrounded the theoretical existence of a Garden of Eden pattern, one that has no possible ancestor. Early proofs determined that such a pattern must exist within a grid 10,000,000,000 squares on each side. By 1974, two had been found, one of which has only 226 cells.
>
> A newsletter devoted to Life research called Lifeline was published from March 1971 to September 1973. Articles on Life appeared in Byte magazine, IBM Research Reports, and even Time magazine (January 1974).
>
> If you’d like to learn more about Life, a good place to start is Martin Gardner’s "Wheels, Life and Other Mathematical Amusements"published in 1983 by W. H. Freeman and Company.

# 2 Spilleregler

Der er tre simple regler i Life spillet:

1. En levende celle med mindre end to naboer dør af isolation.
2. En levende celle med mere end tre naboer dør af overbefolkning.
3. Nyt liv opstår i en tom celle, som har netop tre levende naboer.




Figur 1: *To generation af tre levende life celler.*

Figur 1 viser et simpelt eksempel på et bræt med tre levende celler (øverst) og det samme bræt efter at have anvendt reglerne (nederst).

# 3 Om programmet

Programmet, som udvikles på literate vis i dette skrift, er oprindelig udviklet til brug på basisuddannelsen af Kurt Nørmark (til den frie studieaktivitet “Eksempel på problemløsning med datamater”). I dets nuværende udformning tjener programmet som en illustration af literate programming med det sprogu-afhængige værktøj Nuweb.

Programmet skrives i Turbo Pascal 7.0.

# 4 Den abstrakte datatype LifeVerden

En “LifeVerden” repræsenterer et bræt med en life population. Fra et programmeringsperspektiv repræsenterer vi en LifeVerden som en abstrakt datatype. Vores sproglige tilgang til abstrakte datatyper er simpel, i og med vi ikke anvender hverken units eller objects fra Turbo Pascal; Vi anvender derimod sprogets simple faciliteter på en disciplineret måde, således at vi opnår fordelene ved dataabstraktion.

Vi beslutter at en Life Verden skal repræsenteres af en type, som vi kalder LifeVerden. Data af denne type overføres som parameter og som resultat til og fra operationer i typen. Vi udskyder beslutningen om repræsentationen af en Life verden til senere (se afsnit 6).

Vi giver her et rids af den abstrakte datatype:

⟨LifeVerden ADT 4⟩ ≡

```
(* ******************** ADT LifeVerden ********************** *)

(* En life verden er en matrix af celler, som enten kan vaere
   levende eller doede *)

⟨LifeVerden constanter og typer 7⟩

⟨Procedure NulstilVerden(Var V: LifeVerden; x,y: Integer); 8⟩
⟨Procedure SaetStatus(var V: LifeVerden; i,j: integer; s: CelleStatus); 9a⟩
⟨Function Levende(var V: LifeVerden; i,j: Integer): Boolean; 9b⟩
⟨Function NaboLevende(var VR: LifeVerden; i,j: Integer; r: Retning): Boolean; 10a⟩
⟨Function AntalLevendeNaboer(var VR: LifeVerden; i,j: Integer): Integer; 10b⟩
⟨Procedure NyGeneration(var Gammel, Ny: LifeVerden); 11⟩
⟨Procedure PrintVerden(Var V: LifeVerden); 12a⟩

⟨Life Verden initialiseringsprocedurer 12b⟩

(* ***************** Slut paa ADT LifeVerden ***************** *)
```

◇

Macro referenced in scraps 5, 16.

Vi ser ovenfor hovedet af de centrale operationer på en LifeVerden. Lad os her forklare operationerne på grænseflade-niveau. Vi starter med de simple og basale. NulstilVerden er den basale initialiserings operation, som sætter alle celler til at være “døde”. (I slutningen af den abstrakte datatyper er der andre initialiseringsprocedurer, som definerer verden på forskellige interessante måder - mere om disse i 7.3). Proceduren SaetStatus sætter celle (i,j) til at være levende; SaetStatus er altså en mutations operation på en LifeVerden. Funktionen Levende aflæser status af celle (i,j); mere specifikt returnerer funktionen

hvorvidt denne celle er levende.

Nu kommer vi til de mere interessante: NaboLevende er en funktion, som undersøger hvorvidt en bestemt nabo til celle (i,j) er levende. Vi bemærker at en celle kan have op til 8 naboer (mindre hvis cellen er på grænsen af verden). Typen retning af parameter 4 tillader os at adressere en bestemt nabo til celle (i,j). Dette designes i detaljer i 7.2. Antal levende naboer tæller og returnerer antallet af naboer til celle (i,j) som er levende. Bemærk at dette er højst relevant i forhold til reglerne i Lifespillet.

Proceduren NyGeneration er den central procedure, som simulerer et skridt i en LifeVerdens udvikling. Ideen er her at vi genererer en helt ny verden (2. parameter) fra den gamle verden (1. parameter). Ideen er videre at den nye verden bliver den nuværende verden efter dette skridt. Mere om dette i selve simuleringsdelen af programmet (se 5 og 8).

Endelig har vi proceduren PrintVerden, som udskriver en verden på skærmen.

# 5 En simpel anvendelse af LifeVerden

Vi viser her en simpel anvendelse af vores abstrakte datatype. Vi erklærer verden V1 og V2, hvor V1 initialiseres passende, og udskrives. Dernæst kører spillet et trin, hvilket giver V2. V2 overtager V1's rolle, hvorpå der itereres.

"life-simple.pas" 5 ≡

```
Program life (input, output);
uses CRT, Dos;
var faerdig: Boolean;

⟨LifeVerden ADT 4⟩

Var V1,V2: LifeVerden;
begin  (* hovedprogram *)
  ⟨Lav en start situation i V1 6a⟩;
  repeat
      PrintVerden(V1);
      NyGeneration(V1,V2);
      V1 := V2
  until ⟨faerdig 6b⟩
end. ◇
```

Dette hovedprogram viser princippet i anvendelsen af LifeVerden; men hovedprogrammet er for simpelt i praksis. Vi ønsker dels lidt mere kontrol over start initialiseringen og dels over hvordan verden skal udvikle sig. Vi definerer derfor senere (afsnit 8) et mere realistisk hovedprogram.

(Det er imidlertid muligt i Nuweb at generere mere end én programfil som output. Vi genererer derfor en simpel version af programmet, foruden den egentlige version. Vi må derfor lige færdiggøre detaljerne i ovenstående simple hovedprogram:

⟨Lav en start situation i V1 6a⟩ ≡

```
LavTilfaeldigVerden(V1, xVerdenMax, yVerdenMax) ◇
```

Macro referenced in scrap 5.

⟨faerdig 6b⟩ ≡

```
false◇
```

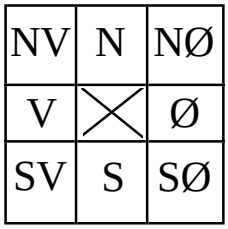
Macro referenced in scrap 5.

Vi beslutter altså, at det simple program skal tage udgangspunkt i en simpel verden, og køre indtil vi (brutalt) slår det ned (med Ctrl Break i Turbo Pascal).

)

# 6 Datarepræsentation i LifeVerden.

Nu hvor vi har introduceret grænsefladen af den abstrakte datatype, og hvor vi har skitseret princippet i anvendelsen af den centrale procedure NyGeneration er det tiden hvor vi skal beslutte os for, hvorledes vi repræsenterer en LifeVerden. Vi vælger den simple og oplagte løsning: et to-dimensionelt array. Vi introducerer to konstanter xVerdenMax og yVerdenMax, som angiver absolutte grænser for verdens størrelse. (Disse er her valgt til størrelser, som tillader os at vise enhver LifeVerden på en PC skærm i DOS mode, hvor vi benytter et tegn pr. celle). Bemærk, at vi sammen med tabellen har mulighed for at angive en mindre verden gennem felterne xmax og ymax i recorden LifeVerden.




Figur 2: *De otte verdenshjørner af en celle.*

⟨LifeVerden constanter og typer 7⟩ ≡

```
Const xVerdenMax = 74;  (* Absolutte graenser        *)
      yVerdenMax = 24;  (* for stoerrelsen af verden *)

Type CelleStatus = (I_live,Doed);
     LifeVerden = record
                    matrix: array[1..xVerdenMax,1..yVerdenMax] of Cel-
leStatus;
                    xmax: 1..xVerdenMax;
                    ymax: 1..yVerdenMax;
                  end;
     Retning = (N,NO,O,SO,S,SV,V,NV);
```

◇

Macro referenced in scrap 4.

Vi kan, i forbifarten, bemærke, at der naturligvis findes mange andre måder at repræsentere en verden i Life. Hvis verden er stor, og hvis der er få celler, vil det givetvis være mere oplagt at repræsentere verden som en liste af levende celler. I denne forbindelse er det beroligende for os, at vi programmerer en Life verden som en abstrakt datatype, hvor vi jo forholdsvis let kan udskifte repræsentationen.

Ovenfor har vi endvidere vist typen Retning, som blev omtalt i afsnit 4 ovenfor. Man skal tænke på konstanterne i opremsningen Retning som verdenshjørner: N for Nord, NO for Nord Øst, etc. Se endvidere figur 2.

# 7 Operationerne i den abstrakte datatype.

Vi kan nu gå igang med at programmere operationerne i den abstrakte datatype. Vi angriber disse i samme rækkefølge som vi introducerede dem ovenfor. Vi grupperer operationerne som basale, interessante og initialiserende.

## 7.1 De basale operationer

Vi starter med NulstilVerden, som blot sætter alle celler til at være døde:

⟨Procedure NulstilVerden(Var V: LifeVerden; x,y: Integer); 8⟩ ≡

```
Procedure NulstilVerden(Var V: LifeVerden; x,y: Integer);
var i,j: Integer;
begin
  V.xmax := x;  V.ymax := y;
  for j:= 1 to yVerdenMax
  do for i := 1 to xVerdenMax
     do V.Matrix[i,j] := Doed
end; (* NulstilVerden *)
```

◇

Macro referenced in scrap 4.

Så følger SaetStatus. Vi har nu behov for at beslutte, hvordan vi vil håndtere “vores del af verden” i forhold til hele universet. Med andre ord, skal vi besluttet hvordan vi håndterer de celler, som ligger uden for rektanglet (1..xmax, 1..ymax). Vores beslutning er meget væsentlig for hvordan vi håndterer celler på randen af verden lidt senere i programmet. Beslutningen er følgende:

> *Alle celler uden for det rektangel, som vi repræsenterer eksplicit, opfattes som døde. Vi har ingen mulighed for at ændre celler udenfor rektanglet. Vi vil derimod kunne aflæse disse (og resultatet skal altid være, at disse er døde).*

Med disse beslutninger er SaetStatus enkelt at lave:

⟨Procedure SaetStatus(var V: LifeVerden; i,j: integer; s: CelleStatus); 9a⟩ ≡

```
Procedure SaetStatus(var V: LifeVerden; i,j: integer; s: CelleStatus);
(* Definer indholdet at celle (i,j) i verden V til status s *)
begin
  if (i >= 1) and (i <= V.xmax) and
     (j >= 1) and (j <= V.ymax)
  then V.matrix[i,j] := s
end;
```

◇

Macro referenced in scrap 4.

Ligeledes kan vi umiddelbart programmere aflæsningsfunktionen Levende:

⟨Function Levende(var V: LifeVerden; i,j: Integer): Boolean; 9b⟩ ≡

```
Function Levende(var V: LifeVerden; i,j: Integer): Boolean;
(* Returner hvorvidt felt (i,j) i verden V er levende. (i,j) kan angive
et felt uden for verden, i hvilket tilfaelde resultatet er falsk *)
begin
  if (i < 1) or (i > V.xmax) or
     (j < 1) or (j > V.ymax)
  then Levende := false
  else Levende := V.matrix[i,j] = I_live
end; (* Levende *)
```

◇

Macro referenced in scrap 4.

Vi bemærker hvorledes vi returnerer false for alle celler uden for den repræsenterede verden.

## 7.2 De interessante operationer

Nu kommer vi til en operation som er “lidt træls”. For at kunne implementere reglerne fra 2 har vi brug for at kunne spørge på status af nabocellerne af celle (i,j). Vi har besluttet at naboceller refereres ved brug af verdenshjørne angivelse, som omtalt i afsnit 6. Det er her vi betaler prisen for denne pæne måde at referere til nabocellerne. Med andre ord: Det er her vi implementerer hvad celle nord for celle (i,j) er for en celle, og så fremdeles.

⟨Function NaboLevende(var VR: LifeVerden; i,j: Integer; r: Retning): Boolean; 10a⟩ ≡

```
Function NaboLevende(var VR: LifeVerden; i,j: Integer; r: Ret-
ning): Boolean;
(* Returner hvorvidt feltet i retning r i forhold til feltet (i,j)
er levende. Et felt der falder uden for verden er altid doedt *)
begin
  case r of
    N:  NaboLevende := Levende(VR,i,j-1);
    NO: NaboLevende := Levende(VR,i+1,j-1);
    O:  NaboLevende := Levende(VR,i+1,j);
    SO: NaboLevende := Levende(VR,i+1,j+1);
    S:  NaboLevende := Levende(VR,i,j+1);
    SV: NaboLevende := Levende(VR,i-1,j+1);
    V:  NaboLevende := Levende(VR,i-1,j);
    NV: NaboLevende := Levende(VR,i-1,j-1);
  end (* case *)
end; (* NaboLevende *)
```

◇

Macro referenced in scrap 4.

Når man nærlæser reglerne i spillet kan man se, at alle tre regler refererer til antallet af (levende) naboceller. Vi nærmer os kraftig implementationen af reglerne hvis vi kan programmere en funktion, som returnerer antallet af levende naboer. Og fordi vi har været lidt smarte i forbindelse med introduktionen af de otte verdenshjørner omkring en celle kan vi direkte gennemløbe naboerne til en celle i en for-løkke:

⟨Function AntalLevendeNaboer(var VR: LifeVerden; i,j: Integer): Integer; 10b⟩ ≡

```
Function AntalLevendeNaboer(var VR: LifeVerden; i,j: Integer): Integer;
var r: Retning;
    antal: integer;
begin
  antal := 0;
  for r := N to NV do
    if NaboLevende(VR,i,j,r)
    then antal := antal + 1;
  AntalLevendeNaboer := antal
end; (* AntalLevendeNaboer *)
```

◇

Macro referenced in scrap 4.

Vi kommer nu til den absolutte kulmination af vores programmeringsarbejde, nemlig til operationen som simulerer et helt skridt fra en gammel verden til en ny verden. Man kunne umiddelbart være fristet til

at mutere en LifeVerden, ved at anvende reglerne iterativt ned over alle celler. Dette går dog galt, idet reglerne skal anvendes simultant på alle celler, inden vi kan foretage forandringer. Vi vælger derfor her den lette løsning, hvor vi genererer en helt ny verden på basis af den gamle:

⟨Procedure NyGeneration(var Gammel, Ny: LifeVerden); 11⟩ ≡

```
Procedure NyGeneration(var Gammel, Ny: LifeVerden);
(* På basis af den gamle verden laves en ny verden *)
var i,j: integer;
begin
  NulstilVerden(Ny, Gammel.xmax, Gammel.ymax);
  for j:= 1 to Gammel.ymax
  do for i := 1 to Gammel.xmax
     do if Levende(Gammel,i,j) then
           if (AntalLevendeNaboer(Gammel,i,j) < 2) or
              (AntalLevendeNaboer(Gammel,i,j) > 3)
           then SaetStatus(Ny,i,j,Doed)
           else SaetStatus(Ny,i,j,I_live)
        else (* (i,j) i Gammel er doed *)
           if AntalLevendeNaboer(Gammel,i,j) = 3
           then SaetStatus(Ny,i,j,I_live)
           else SaetStatus(Ny,i,j,Doed)
end; (* NyGeneration *)
```

◇

Macro referenced in scrap 4.

Vi starter med at nustille den nye verden, hvorefter vi en en dobbelt for-løkke gennemløber alle cellerne i den gamle verden (og ny verden) og samtidigt definerer cellerne “passende” i den nye verden. Bemærk venligst hvor naturligt vi kan formulere de tre Life regler i programmet. Årsagen er naturligvis at vi har udført et grundigt forarbejde i forbindelse med programmeringen af funktionenerne AntalLevendeNaboer og Levende.

Vi kommer nu til proceduren, som udskriver en LifeVerden på skærmen:

⟨Procedure PrintVerden(Var V: LifeVerden); 12a⟩ ≡

```
Procedure PrintVerden(Var V: LifeVerden);
(* Udskriv verden V *)
var i,j: integer;
begin
  ClrScr;
  for j:= 1 to V.ymax
  do begin
       for i := 1 to V.xmax
       do if Levende(V,i,j) then write('X') else write(' ');
       writeln;
     end;
end; (* PrintVerden *)
```

◇

Macro referenced in scrap 4.

Vi ser, at en levende celle markeres med et kryds (et "X"), og at en død celle markes med "ingenting" (et mellemrum).

## 7.3 De initialiserende operationer

Nu viser vi en række operationer, som initialiserer en LifeVerden til at indeholde forskellige interessante mønstre. Bemærk at disse operationer naturligvis er i familie med operationen NulstilVerden fra 7.1 ovenfor.

⟨Life Verden initialiseringsprocedurer 12b⟩ ≡

```
(* PROCEDURURER SOM ALLE LAVER EN BESTEMT START PAA VERDEN *)
```

⟨Procedure LavTilfaeldigVerden(Var V: LifeVerden; x,y: Integer); 13a⟩
⟨Procedure LavTrafikLys(Var V: LifeVerden; x,y: Integer); 13b⟩
⟨Procedure LavDiagonal1(Var V: LifeVerden; x,y: Integer); 14a⟩
⟨Procedure LavKryds(Var V: LifeVerden; x,y: Integer); 14b⟩
⟨Procedure LavGlider(Var V: LifeVerden; x,y: Integer); 14c⟩
⟨Procedure LavBjaelke(Var V: LifeVerden; x,y: Integer); 15⟩
◇

Macro referenced in scrap 4.

Først LavTilfaeldigVerden, der, som navnet antyder, drysser levende celler ud over verden på pseudo-tilfældig vis:

⟨Procedure LavTilfaeldigVerden(Var V: LifeVerden; x,y: Integer); 13a⟩ ≡

```
Procedure LavTilfaeldigVerden(Var V: LifeVerden; x,y: Integer);
(* Initialiser verden V tilfaeldigt.
Saet stoerrelsen af verden til [1 .. x, 1 .. y] *)
var i,j: integer;
    r: Word;
begin
  Randomize;
  NulstilVerden(V,x,y);
  for j:= 1 to V.ymax
  do for i := 1 to V.xmax
     do begin
          r := Random(2); (* enten 0 eller 1 *)
          if r = 0 then SaetStatus(V,i,j,Doed)
          else if r = 1 then SaetStatus(V,i,j,I_live)
          else halt
        end;
end; (* LavTilfaeldigVerden *)
```

◇

Macro referenced in scrap 12b.

Alle øvrige initialiseringsprocedurer laver nøje udvalgte mønstre, som udvikler sig på en sjov måde. Dette er næsten umuligt at forklare. *Det skal opleves!*

⟨Procedure LavTrafikLys(Var V: LifeVerden; x,y: Integer); 13b⟩ ≡

```
Procedure LavTrafikLys(Var V: LifeVerden; x,y: Integer);
var i: integer;
begin
  NulstilVerden(V,x,y);
  for i := 1 to 5 do
  SaetStatus(V,10,11+i,I_live);
end;
```

◇

Macro referenced in scrap 12b.

⟨Procedure LavDiagonal1(Var V: LifeVerden; x,y: Integer); 14a⟩ ≡

```
Procedure LavDiagonal1(Var V: LifeVerden; x,y: Integer);
var i: integer;
begin
  NulstilVerden(V,x,y);
  for i := 1 to V.xmax do SaetStatus(V,i,i,I_live);
end;
```

◇

Macro referenced in scrap 12b.

⟨Procedure LavKryds(Var V: LifeVerden; x,y: Integer); 14b⟩ ≡

```
Procedure LavKryds(Var V: LifeVerden; x,y: Integer);
var i: integer;
begin
  NulstilVerden(V,x,y);
  for i := 1 to V.xmax do SaetStatus(V,i,i,I_live);
  for i:= 1 to V.xmax do SaetStatus(V,V.xmax-i+1,i,I_live);
end;
```

◇

Macro referenced in scrap 12b.

⟨Procedure LavGlider(Var V: LifeVerden; x,y: Integer); 14c⟩ ≡

```
Procedure LavGlider(Var V: LifeVerden; x,y: Integer);
begin
  NulstilVerden(V,x,y);
  SaetStatus(V,12,12,I_live);
  SaetStatus(V,13,12,I_live);
  SaetStatus(V,14,12,I_live);
  SaetStatus(V,14,11,I_live);
  SaetStatus(V,13,10,I_live);
end;
```

◇

Macro referenced in scrap 12b.

⟨Procedure LavBjaelke(Var V: LifeVerden; x,y: Integer); 15⟩ ≡

```
Procedure LavBjaelke(Var V: LifeVerden; x,y: Integer);
var i: integer;
begin
  NulstilVerden(V,x,y);
  for i := 1 to 7 do
  SaetStatus(V,10,11+i,I_live);
end;
```

◇

Macro referenced in scrap 12b.

Hermed slutter vi arbejdet med den abstrakte datatype LifeVerden, og vi vender blikket mod hovedprogrammet, og hvad vi nu har brug for til dette.

# 8 Det rigtige hovedprogram

Vi har tidligere (afsnit 5) programmeret et simpelt hovedprogram, som vi nu vil forfine til vores egentlige hovedprogram. Vi viser samtidig arkitekturen af det samlede program:

"life.pas" 16 ≡

```
Program life (input, output);
uses CRT, Dos;

var faerdig: Boolean;
```

⟨LifeVerden ADT 4⟩

⟨procedure LavStartSituation(var V: LifeVerden); 18⟩

⟨procedure KoerUendeligt(var V1: LifeVerden); 17⟩

```
Var V1,V2: LifeVerden;
    c: Char;
begin
  ClrScr; faerdig := false;
  LavStartSituation(V1);
  repeat
    PrintVerden(V1);
    c := ReadKey;
    if c = ' '
    then begin
           NyGeneration(V1,V2);
           V1 := V2
         end
    else if c = 'r'
    then KoerUendeligt(V1)
    else begin
           clrScr;
           LavStartSituation(V1)
         end
  until faerdig
end.
```

◇

Nogle få ord om hovedprogrammet: Vi ønsker at have tre muligheder i hovedprogrammet:

1. At kunne udvikle generationerne én for én, på en kontrolleret måde. Interaktionsmæssigt følger én generation efter en anden når vi trykker på mellemrumstasten på tastaturet.
2. At kunne udvikle generationerne lige så hurtigt som maskinen formår at beregne og udskrive dem. Dette er nyttigt hvis vi ønsker at undersøge, om verden konvergerer mod et eller andet stabilt eller semi-stabilt mønster. Dette løses af proceduren KørUendeligt.

3. At kunne angive en ny, interessant startverden blandt dem vi har programmeret i afsnit 7.3. Dette løses af proceduren LavStartSituation.

Ud over disse kunne man være fristet til interaktivt at angive en ny start verden. Vi modstår dog fristelsen her.

Proceduren KoerUendligt går ind i en uendelig løkke, som skal termineres via operativsystemet. I Turbo Pascal gøres dette med Control Break tastaturkombinationen. Dette er nok ikke så pænt et design, men det var let at lave...

⟨procedure KoerUendeligt(var V1: LifeVerden); 17⟩ ≡

```
procedure KoerUendeligt(var V1: LifeVerden);
var V2: LifeVerden;
begin
  repeat
    NyGeneration(V1,V2);
    V1 := V2;
    PrintVerden(V1)
  until false;
end; (* KoerUendeligt *)
```

◇

Macro referenced in scrap 16.

Her kommer så den lidt trælse og primitive dialog, hvor vi spørger brugeren om, hvordan startsituationen skal være. Bemærk hvordan vi i case 7 afslutter' programudførelsen på normal vis.

⟨procedure LavStartSituation(var V: LifeVerden); 18⟩ ≡

```
procedure LavStartSituation(var V: LifeVerden);
(* Spørg brugeren om starten paa verden *)
var valg: Integer;
begin
  writeln('Vælg mellem foelgende startsituationer:');
  writeln('1. Tilfalldig verden');
  writeln('2. Diagonal');
  writeln('3. Kryds');
  writeln('4. Trafiklys');
  writeln('5. Glider');
  writeln('6. Bjaelke');
  writeln('7. Slut');
  readln(valg);
  case valg of
    1: LavTilfaeldigVerden(V,xVerdenMax,yVerdenMax);
    2: LavDiagonal1(V,YVerdenmax,YVerdenMax);
    3: LavKryds(V,YVerdenmax,YVerdenMax);
    4: LavTrafikLys(V,xVerdenMax,yVerdenMax);
    5: LavGlider(V,xVerdenMax,yVerdenMax);
    6: LavBjaelke(V,xVerdenMax,yVerdenMax);
    7: faerdig := true;  (* faerdig er en global variabel *)
  end; (* case *)
end; (* LavStartSituation *)
◇
```

Macro referenced in scrap 16.

# 9 Register

⟨Procedure NyGeneration(var Gammel, Ny: LifeVerden); 11⟩ Referenced in scrap 4.
⟨Procedure PrintVerden(Var V: LifeVerden); 12a⟩ Referenced in scrap 4.
⟨Procedure SaetStatus(var V: LifeVerden; i,j: integer; s: CelleStatus); 9a⟩ Referenced in scrap 4.
⟨faerdig 6b⟩ Referenced in scrap 5.
⟨procedure KoerUendeligt(var V1: LifeVerden); 17⟩ Referenced in scrap 16.
⟨procedure LavStartSituation(var V: LifeVerden); 18⟩ Referenced in scrap 16.