TÆNDSTIKSPILLET NIM

ET GIER DEMONSTRATIONSPROGRAM

Jørgen Kjær

Haldor Topsøe, Vedbæk, Danmark

1966

Copy No. 1

Haldor Topsøe, Chemical Engineers

Vedbæk, Denmark

FORORD

En moderne elektronregnemaskine opfattes af de fleste mennesker som noget aldeles uforstaaeligt. Den følelse af afmagt, der griber en, naar man ser maskinen addere mange tusind tal eller beregne flere hundrede kvadratrødder i løbet af eet sekund, kan forvandles til en total aandelig lammelse, hvis man ogsaa bliver stillet overfor det krav at skulle drage nytte af regnemaskinen til løsning af ens egne regneproblemer.

Der er derfor et behov for oplysning af enhver art, som kan fremme forstaaelsen af elektronregnemaskinernes virkemaade og anvendelse. Vi lykkelige faa, der til daglig omgaas regnemaskinerne, maa søge at befri os for det hovmod, der let indfinder sig hos et præsteskab for nye og ukendte guder, og istedet søge at populærisere forstaaelsen af disse prægtige maskiner, der utvivlsomt er kaldet til at spille en stor rolle i menneskehedens historie.

Hos Haldor Topsøe anvendes GIER-regneanlægget i det væsentlige til beregninger indenfor den kemiske og mekaniske ingeniørvidenskab. De hertil benyttede programmer er ofte meget komplicerede og vanskelige at forstaa for begyndere i programmeringsfaget. Naar regneanlægget forevises firmaets udenlandske gæster, er disse rutineprogrammer næppe egnet til at give et slaaende indtryk af maskinens regnehastighed. Kun de færreste gæster vil blive ret meget klogere, naar man fortæller dem, at maskinen i dette øjeblik er i færd med at løse tretten samtidige differentialligninger, og at den bliver færdig med dette om ca. en halv time.

Jeg har derfor lavet nogle smaa programmer, der er særlig velegnet som demonstrationsprogrammer, idet de baade illustrerer maskinens store regnehastighed og ogsaa viser princippet i selve programmeringens ide, d.v.s. at problemet skal være gennemtænkt og programmeret indtil den mindste detaille, før en beregning kan løbe af stablen. I nærværende bog beskrives et af disse programmer, nemlig det, som underholder gæsten ved at spille tændstikspillet NIM med ham.

En beskrivelse af dette tændstikprogram har naturligvis en vis interesse i sig selv, men jeg har gjort en hel del mere ud af baade programmet og beskrivelsen, end man normalt ville gøre for et saa beskedent problem. Det er mit haab, at programmet ogsaa kan have en vis pædagogisk værdi for programmører. Det er et lille program, de matematiske problemer i det er ganske simple, og jeg har ved en passende opdeling i procedurer givet det en - forhaabentlig - overskuelig form, som baade gør det let at fatte og samtidig er en illustration af det procedure-orienterede ALGOL-sprog.

Det er ogsaa min hensigt at udgive lignende beskrivelser af andre demonstrationsprogrammer, f.eks. til beregning af store tal, ligesom beskrivelserne ogsaa vil komme paa engelsk.

Virum, Marts 1966
Jørgen Kjær.

INDHOLDSFORTEGNELSE

1. INDLEDNING

NIM-spillet er et tændstikspil, som spilles af to personer. Man begynder med nogle bunker af tændstikker, og de to parter skal skiftevis fjerne tændstikker fra bunkerne. Den, som tager den sidste tændstik, har vundet. En nøjere beskrivelse af spillet gives i kapitel 2.

Naar regnemaskinen skal spille dette spil med et menneske som modspiller, maa den først have indlæst et program, der fortæller den, hvordan den skal bære sig ad. Opbygningen af et saadant program beskrives i de følgende kapitler. Programmet er opdelt i et stort antal procedurer, samt det egentlige program.

I kapitel 3 omtales nogle administrationsprocedurer, der tjener det formaal at formidle maskinens kommunikation med modspilleren. Maskinen skal kunne stille spørgsmaal, som modspilleren besvarer med ja eller nej, og den skal ogsaa kunne spørge om værdien af et tal. Programmet er iøvrigt indrettet til at arbejde i et af de fire sprog: Dansk, Engelsk, Fransk eller Tysk, og man begynder spillet med at vælge sprog. Hvis modspilleren under spillet svarer ja paa et andet af de fire sprog end det først valgte, skifter maskinen automatisk til det nye sprog.

Maskinen begynder normalt med at vælge et tilfældigt antal tændstikker i bunkerne. I kapitel 4 forklares, hvorledes maskinen kan generere saadanne tilfældige tal.

Kapitel 5 omtaler de egentlige spilprocedurer. Foruden nogle mindre interessante hjælpeprocedurer, har vi her proceduren GAME, der administrerer spillets gang ved at skifte mellem maskinen og modspilleren indtil spillet er afgjort. Programmet er naturligvis indrettet til at give det bedst mulige spil fra maskinens side. Naar maskinen skal gøre sit første træk, kan den med sikkerhed afgøre, om den kan vinde, og hvis det er tilfældet, vil den ogsaa vinde. Maskinen benytter forskellige procedurer til at afgøre, hvad der er den bedste strategi. Desuden er der to procedurer: MAN og MACHINE, som administrerer trækkenes udførelse.

I kapitel 6 diskuteres det egentlige program, og der vises eksempler paa spillet.

Kapitel 7 indeholder referencer.

I appendikset kapitel 8 vises ALGOL-programmet og en udskrift fra et spil.

Programmet og procedurerne er skrevet i GIER ALGOL III, og det forudsættes, at læseren har det nødvendige kendskab til ALGOL, se f.eks. Kjær (1964).

2. NIM-SPILLET

2.1. Historisk.

Navnet NIM synes at være konstrueret af den amerikanske matematikprofessor, Bouton, der i 1901 skrev en artikel om spillet og dets matematiske teori. Han viste, at man altid kan afgøre, om den person, der skal trække, er i en sikker position eller ikke. Hvis en spiller er fortrolig med reglerne, og sørger for at efterlade bunkerne i en sikker position, efter at han har trukket, vil han med sikkerhed vinde spillet. Han viste ogsaa, hvorledes man med en simpel beregning kan afgøre, om en position er sikker. Endelig omtalte han en variant af spillet, hvori den som tager sidste tændstik, har tabt.

Ved forskellige udstillinger har man vist smaa elektronregnemaskiner, som var specielt bygget til at spille NIM med udstillingens gæster.

I det følgende gennemgaar vi først det normale spil og reglerne herfor, og derefter varianten.

2.2. Det normale spil.

Vi gennemregner nu først et typisk spil, hvor A er maskinen og B modspilleren. Vi tænker os, at vi har tre bunker med 4, 1 og 3 tændstikker:

Bunke nr.:	1	2	3
Antal:	4	1	3

Maskinen (A) begynder, og den fjerner først to tændstikker fra bunke nr. 1. Bunkerne indeholder da:

2 1 3

B kender ikke reglerne og fjerner paa maa og faa 2 fra bunke nr. 3. Dette giver:

2 1 1

Nu fjerner maskinen hele bunke nr. 1:

0 1 1

B har da valget mellem at fjerne hele bunke nr. 2 eller hele bunke nr. 3. Hvis han tager nr. 2, vil maskinen tage nr. 3 og omvendt. Maskinen vinder derfor.

2.3. Regler for normalt spil.

Vi skal nu vise, hvordan man kan opstille reglerne for et sikkert spil. Som nævnt ovenfor, er det et væsentligt element i spillets teori, at den dygtige spiller skal sørge for at efterlade bunkerne i en sikker position, naar han har trukket. Teorien siger endvidere, at enhver sikker position altid vil blive ændret til en usikker position, naar modspilleren har gjort sit træk, saaledes at man faar følgende skema:

A	B
SIKKER	
	USIKKER
SIKKER	
	USIKKER
SIKKER	
	USIKKER
SIKKER	

o.s.v.

Vi behøver da blot yderligere at forudsætte, at vinderpositionen er sikker, altsaa at lutter tomme bunker tilhører mængden af sikre positioner.

Vi skal nu finde frem til definitionen paa en sikker position. Først ser vi paa det tilfælde, hvor der kun er to bunker tilbage, de andre er tomme. Her vil man faa en sikker position, hvis man efterlader de to bunker med samme indhold:

1 2
17 17

Kan A efterlade bunkerne med dette indhold, vil han være i en sikker position. B skal nemlig enten fjerne hele den ene bunke, og A kan da tage

den anden og har vundet, eller B tager kun noget af den ene bunke, og A kan da fjerne lige saa meget fra den anden bunke, hvorved vi er tilbage i den samme sikre position. Vi faar da følgende skema:

Sikker: Samme indhold
Usikker: Forskelligt indhold.

Vi ser, at B altid maa ændre en sikker position til en usikker, samt at vinderpositionen (0 0) hører til mængden af sikre positioner.

Et andet simpelt tilfælde faar vi, hvis vi har mange bunker, men med kun een tændstik i hver, f.eks:

1	2	3	4	5
1	1	1	1	1

Her er forholdene endnu simplere. Hvert træk bestaar i, at en spiller fjerner en hel bunke med een tændstik. Man ser let, at vi her faar følgende definition paa sikre og usikre positioner:

Sikker: Lige antal bunker.
Usikker: Ulige antal bunker.

Ogsaa her vil B altid ændre en sikker position til en usikker, og vinderpositionen (0 bunker) tilhører mængden af sikre positioner.

I det almindelige tilfælde med mange bunker med mange tændstikker i hver har Bouton (1901) vist, at man kan beregne sikkerheden paa følgende maade. Vi tager det samme eksempel som ovenfor:

Bunke nr.:	1	2	3
Antal:	4	1	3

Indholdet af bunkerne skal nu nedskrives i totalsystemet, d.v.s. efter reglerne:

Titalsystem (normalt)	Totalsystem
0	0
1	1
2	10
3	11

4	100
5	101
6	110
7	111
8	1000
9	1001
10	1010
11	1011

o.s.v.

Gør vi dette med de tre bunker ovenfor, faar vi:

	Indhold	
Bunke nr:	Titalsystem	Totalsystem
1	4	100
2	1	1
3	3	11

Ligesom tallet 317 i titalsystemet betyder 7 enere, 1 tier og 3 hundreder, vil tallet 1011 i totalsystemet betyde 1 ener, 1 toer, 0 firere og 1 otter. Vi skal addere antallet af enere, toere, o.s.v. i bunkerne:

	Indhold				
Bunke nr:	Tital-system	Total-system	Firere	Toere	Enere
1	4	100	1	0	0
2	1	1			1
3	3	11		1	1
			---	---	---
	Sum		1	1	2

Summen bliver altsaa 1 firer, 1 toer og 2 enere. Teorien siger nu, at disse summer (1, 1 og 2) hver skal være lige, efter at man har gjort et træk, der fører til en sikker position.

Lad os vise, at dette er rigtigt. Hvis A har trukket, saaledes at alle summer er lige, som defineret ovenfor, da maa Bs træk nødvendigvis gøre een eller flere af summerne ulige. Han maa kun fjerne tændstikker fra een af bunkerne, og fjernelsen betyder, at een eller flere af enerne eller toerne eller firerne o.s.v. i denne bunke bliver fjernet eller tilføjet, d.v.s. bliver gjort ulige istedet for lige.

Naar B ved sit træk har ændret positionen fra at være sikker til at være usikker, kan A altid i det næste træk gøre positionen sikker igen. Han ser, i hvilke søjler summen er ulige. Hvis een af bunkerne indeholder enere, toere, firere o.s.v. i netop disse søjler, kan han fjerne hele denne kombination fra denne bunke. Findes der ikke en saadan bunke, maa han vælge een, hvori der findes den højeste potens med ulige sum. Denne potens fjernes da fra bunken, men der adderes samtidigt nogle af de lavere potenser. I kapitel 5 vises den nøjagtige procedure hertil. Vinderpositionen (lutter tomme bunker) har naturligvis kun lige summer (lutter nuller).

I eksemplet ovenfor med 4, 1 og 3 tændstikker fjerner maskinen først to fra den første bunke. Dette giver:

	Indhold			
Bunke nr:	Tital	Total	Toere	Enere
1	2	10	1	0
2	1	1		1
3	3	11	1	1
			---	---
	Sum		2	2

Summerne er nu lige. Naar B har fjernet 2 fra bunke nr. 3 har vi:

	Indhold			
Bunke nr:	Tital	Total	Toere	Enere
1	2	10	1	0
2	1	1		1
3	1	1		1
			---	---
	Sum		1	2

Her er toernes sum ulige. A fjerner derfor hele bunke nr. 1 og spillets udfald er klart.

Da vi kun har brug for at vide, om summerne er lige eller ulige, behøver vi blot at lave additionen modulo 2, d.v.s. en lige sum skrives som nul og en ulige som 1.

Vi tager nu et mere kompliceret eksempel med 6 bunker og nedskriver straks summerne modulo 2:

	Indhold	
Bunke nr:	Tital	Total
1	5	101
2	35	100011
3	9	1001
4	47	101111
5	22	10110
6	39	100111

Sum modulo 2		110001

Summerne indeholder altsaa 1 ener, 1 sekstener og 1 toogtredver. Da ingen af bunkerne indholder alle disse potenser, maa vi vælge en bunke, som indeholder en toogtredver. Lad os vælge bunke nr. 6:

Nr. 6	100111
Sum	110001

Fjerner vi 32 fra bunke nr. 6 er søjlen med toogtredvere i orden. Vi kan ogsaa fjerne en ener fra bunke nr. 6, saa er enerne ogsaa i orden:

Nr. 6	000110
Sum	010000

Men sekstenerne passer ikke, saa her maa vi tilføje en sekstener i bunke nr. 6:

Nr. 6	010110
Sum	000000

Positionen er nu sikker, og vi har ialt fjernet 32 + 1 - 16 = 17 fra bunke nr. 6, og det vil maskinen ogsaa gøre, hvis det er dens tur til at trække. Er det derimod B, som skal trække først, kan han vælge den sikre position, hvis han kender reglerne, og han vil derfor vinde, forudsat at han ikke senere laver forkerte træk. I denne situation vil maskinen nøjes med en henholdende politik, idet den fjerner en enkelt tændstik ad gangen (fra den største bunke).

2.4. Det omvendte spil.

I denne variant gælder det om ikke at fjerne den sidste tændstik. Her kan man vise, at man skal følge samme strategi til udvælgelse af en sikker position, som i det normale spil, indtil man er kommet saa langt ned, at ingen bunke indeholder mere en een tændstik. Da vil en sikker position være een, hvor man efterlader et ulige antal bunker.

Det er let at se, at et ulige antal bunker med een tændstik i hver maa være en sikker position, men vi maa sikre os, at man kan passere fra den normale strategi til den specielle for det omvendte spil uden vanskelighed. Saalænge der er mindst to bunker med mere end een tændstik vælger vi den normale strategi, men saa snart der kun er een bunke med mere end een tændstik, skal vi enten fjerne hele denne bunke, eller efterlade een tændstik, saaledes at antallet af bunker er ulige.

3. ADMINISTRATIONSPROCEDURER

Det er praktisk at udskille visse procedurer af administrativ karakter og at beskrive disse for sig. Nogle af disse procedurer er af generel karakter og kan derfor bruges i andre demonstrationsprogrammer.

3.1. Sideskift.

Maskinen og modspilleren meddeler sig til hinanden via den tilkoblede skrivemaskine. Da denne skriver paa en endeløs papirbane, som er perforeret paa tværs, saaledes at man kan afrive papiret i stykker med 70 linier paa hver side, er det praktisk, hvis maskinen sørger for, at der ikke bliver skrevet noget paa nogle faa linier lige før og lige efter perforeringen. Maskinen tæller til 70 paa hver side og laver nogle tomme linier omkring perforeringen. Programmet forudsætter, at man begynder øverst paa en side.

Der benyttes to procedurer hertil: NEW PAGE og LINE. Begge procedurer benytter en global variabel:

```
integer linerest;
```

som angiver, hvormange frie linier, der er tilbage paa siden.

3.1.1. Proceduren NEW PAGE. Denne procedure har deklarationen:

```
procedure NEW PAGE;
for linerest := linerest - 1 while linerest ≥ 0, 69 do writecr;
```

Den skriver saa mange nye linier, som linerest angiver og sætter linerest lig med 69, svarende til værdien øverst paa næste side.

3.1.2. Proceduren LINE. Deklarationen herfor er:

```
procedure LINE;
if linerest < 8 then NEWPAGE
else
begin
```

```
    linerest := linerest -1;
    writecr
end LINE;
```

I programmet skal man ikke skrive:

```
writecr;
```

for at faa en ny linie, men derimod:

```
LINE;
```

LINE vil normalt udføre een writecr, samt trække een fra linerest, men hvis linerest i forvejen er mindre end 8, vil den kalde proceduren NEW PAGE.

3.2. Spørgsmaal.

Vi bruger fire specielle procedurer til udveksling af oplysninger mellem maskinen og modspilleren og omvendt.

3.2.1. Proceduren WRITE TEXT. Naar maskinen skal skrive et stykke tekst paa skrivemaskinen, kan dette gøres med den sædvanlige procedure:

```
writetext(«<Her skrives teksten»);
```

Men da vi ønsker at maskinen skal kunne skrive paa de fire forskellige sprog: dansk, engelsk, fransk eller tysk, er det mere bekvemt at lave en speciel procedure hertil:

```
procedure WRITE TEXT(dan, eng, fr, ger);
string dan, eng, fr, ger;
writetext(if lang = 1 then dan
     else if lang = 2 then eng
     else if lang = 3 then fr else ger);
```

Vi benytter her en global variabel:

```
integer lang;
```

Denne variable kan have fire værdier svarende til de fire sprog:

1: Dansk
2: Engelsk
3: Fransk
4: Tysk

Der er fire formelle parametre i proceduren, nemlig de fire tekststrenge paa de fire forskellige sprog. Et kald af proceduren kan se saaledes ud:

```
WRITE TEXT({<Hest}, {<Horse}, {<Cheval}, {<Pferd});
```

Proceduren virker naturligvis kun, hvis lang har en værdi mellem 1 og 4. Man indstiller værdien med den følgende procedure.

3.2.2. Proceduren SELECT LANGUAGE. Denne procedure har deklarationen:

```
procedure SELECT LANGUAGE;
begin
   LINE;
   writetext(
   {<Select language: d: danish, e: english, f: french, g: german.: });
   lang := typechar - 51;
   if lang < 1 then lang := 1;
   if lang > 4 then lang := 4;
   LINE;
   WRITE TEXT(
   {<Dansk},
   {<English},
   {<Francais},
   {<Deutsch});
   LINE
end SELECT LANGUAGE;
```

Først laves en ny linie, og der udskrives:

Select language: d: danish, e: english, f: french, g: german.:

Denne tekst udskrives altsaa paa engelsk, da man endnu ikke har valgt sprog. Man skal nu skrive et af de fire bogstaver: d, e, f og g paa skrivemaskinen. Disse bogstaver har talværdierne:

d	52
e	53
f	54
g	55

ALGOL-sætningen:

```
lang := typechar - 51;
```

vil da bevirke, at vi faar lang = 1 for d, lang = 2 for e, o.s.v.

For at sikre os, at man ikke driller maskinen ved at skrive noget helt andet end de fire bogstaver, har vi anbragt de to kontrolsætninger:

```
if lang < 1 then lang := 1;
if lang > 4 then lang := 4;
```

Proceduren laver derefter en ny linie og skriver navnet paa det valgte sprog ved et kald af **WRITE TEXT**. Til sidst laves endnu en ny linie.

SELECT LANGUAGE skal naturligvis anbringes i begyndelsen af programmet.

3.2.3. Proceduren ASK NUMBER. Dette er en real procedure, som programmet kalder, naar det skal have et tal at vide af modspilleren. Deklarationen er:

```
real procedure ASK NUMBER(dan, eng, fr, ger);
string dan, eng, fr, ger;
begin
   LINE;
   WRITE TEXT(dan, eng, fr, ger);
   writetext(‡<: ‡);
   ASK NUMBER := typein
end ASK NUMBER;
```

Der er fire formelle parametre, nemlig de fire tekststrenge, som maskinen skal bruge til at spørge paa et af de fire sprog. Proceduren laver først en ny linie, skriver spørgsmaalet, og afslutter det med et kolon (skrivemaskinen har intet spørgsmaalstegn). Derefter venter den paa input fra skrivemaskinen:

```
ASK NUMBER := typein;
```

En typisk anvendelse af denne procedure er følgende kald:

```
groups := ASK NUMBER(
          {<Hvor mange bunker},
          {<How many groups},
          {<Combien de groupes},
          {<Wieviel Haufen});
```

Her er den variable, groups, aabenbart en integer, og da ASK NUMBER er en real procedure, bliver der automatisk afrundet.

3.2.4. Proceduren QUESTION. Denne boolean procedure har deklarationen:

```
boolean procedure QUESTION(dan, eng, fr, ger);
string dan, eng, fr, ger;
begin
   integer symb;
AGAIN: LINE;
   WRITE TEXT(dan, eng, fr, ger);
   writetext({<: });
   symb := typechar;
   if symb = 37 then
   WRITE TEXT({<ej}, {<o}, {<on}, {<ein})
   else
   if symb = 24 ∨ symb = 33 ∨ symb = 38 then
   begin
      lang := if symb = 24 then 2
      else if symb = 38 then 3
      else if lang ≤ 2 then 1 else 4;
```

```
      WRITE TEXT(‹<a›, ‹<es›, ‹<ui›, ‹<a›)
   end if yes
   else
   begin
      LINE;
      WRITE TEXT(
      ‹<Det forstaar jeg ikke›,
      ‹<I do not understand this›,
      ‹<Je ne comprends pas cela›,
      ‹<Das verstehe ich nicht›);
      go to AGAIN
   end if nonsense;
   QUESTION := symb ≠ 37
end QUESTION;
```

Her har vi ogsaa de fire formelle parametre, som udgør de fire tekststrenge, for det spørgsmaal, som maskinen skal stille paa et af de fire sprog. Der laves først en ny linie, og derefter skriver maskinen spørgsmaalet, afsluttet af kolon.

Modspilleren skal nu svare ja eller nej paa et af de fire sprog. Der indlæses et enkelt bogstav med:

```
symb := typechar;
```

Hvis det læste bogstav er et n, fortolker proceduren det som et nej, og den kvitterer ved at skrive resten af ordet nej paa paagældende sprog:

```
1:   nej
2:   no
3:   non
4:   nein
```

Hvis det læste bogstav ikke er et n, undersøger proceduren, om det er et af de tre bogstaver: y, j eller o. Hvis dette er tilfældet, opfatter proceduren det som ja, og den skriver resten af ordet:

```
1:   ja
2:   yes
3:   oui
4:   ja
```

Her er der den komplikation, at modspilleren ved en fejltagelse eller bevidst kommer til at svare ja paa et andet sprog end det, som maskinen arbejder i for øjeblikket. Hvis maskinen arbejder med dansk (lang = 1), og man skriver y, vil den skifte lang til 2 og skrive: es. Proceduren indstiller derfor simpelthen værdien af lang paany med sætningen:

```
lang := if symb = 24 then 2
else if symb = 38 then 3
else if lang ≤ 2 then 1 else 4;
```

Da j baade kan være begyndelsen til ja paa dansk og begyndelsen til ja paa tysk, opstaar der problemet, hvilket af de to sprog man skal skifte til, hvis maskinen i forvejen arbejder med engelsk eller fransk. Proceduren er indrettet saaledes, at et j bevirker skift til dansk, hvis den i forvejen arbejder med dansk eller engelsk og vi faar skift til tysk, hvis den i forvejen arbejder i fransk eller tysk.

Hvis det læste bogstav hverken er n, y, j eller o, skriver maskinen en ny linie og teksten:

Det forstaar jeg ikke

eller det tilsvarende paa de andre sprog. Derefter begynder den forfra med at gentage spørgsmaalet og forvente et svar.

Naar proceduren er færdig, sættes den logiske værdi af QUESTION til falsk, hvis der er indlæst et n, ellers til sand.

Bemærk, hvorledes proceduren kun skelner mellem de to typer af information:

n, y, j eller o:	Svar
Alt andet:	Uforstaaeligt.

Hvis modspilleren haardnakket nægter at opgive et af de fire forstaaelige bogstaver, vil man aldrig komme ud af proceduren.

4. TILFÆLDIGE TAL

4.1. Tilfældige tal i elektronregnemaskiner.

Det lyder paradoksalt, at man i en elektronregnemaskine skulle kunne regne med tilfældige tal, da regnemaskinen jo netop ikke gør noget tilfældigt, hvert skridt i dens virke er givet ved det program, den arbejder med i øjeblikket, d.v.s. at det næste skridt er fuldstændigt bestemt.

Hvis vi ser bort fra det specielle tilfælde, hvor man lader de tilfældige tal genereres af et specielt tilkoblet apparat (en støjdiode eller et apparat med en radioaktiv kilde), som ved at arbejde med en helt anden fysisk process giver virkeligt tilfældige tal, maa det erkendes, at de tal, der normalt genereres af procedurer til tilfældige tal, bør betegnes som pseudo-tilfældige tal. Den talrække, som en givet procedure til tilfældige tal kan generere, er fuldstændigt bestemt. Man kan beregne dem paa forhaand, ligesom man kan lade maskinen trykke en tabel over dem. Det afgørende er, at tallene er tilfældige i forhold til den anvendelse man ønsker at gøre med dem. Løst formuleret kan man sige, at hvis den numeriske proces, der genererer tallene, er tilstrækkeligt forskellig fra de numeriske processer, der indeholdes i anvendelsen, da kan proceduren til de tilfældige tal anvendes. Ønsker man at være helt sikker, maa man foretage en statistisk prøve, for at se om en foreliggende procedure til tilfældige tal kan anvendes til et givet problem.

I NIM-programmet benytter vi blot proceduren til tilfældige tal til at udvælge hvormange tændstikker, der skal være i bunkerne ved spillets begyndelse. Saalænge vi betragter hvert spil som et isoleret fænomen, og vi ikke laver statistik over udfaldet af et større antal spil, er det ikke nødvendigt med nogen statistisk prøve.

4.2. Proceduren RANDOM INTEGER.

En hyppigt anvendt metode til generering af tilfældige tal benytter følgende princip. Lad R være et eller andet tilfældigt heltal i rækken af de pseudo-tilfældige heltal, vi skal generere. Vi kommer da til det

næste tal i rækken ved først at multiplicere R med en konstant faktor:

K×R

Dette produkt divideres saa med en anden konstant, M, og det næste R er da resten ved denne division. Man kan ogsaa sige, at produktet K×R dannes modulo M.

Man kan bruge mange forskellige værdier af K og M, men de skal tilfredsstille visse betingelser, hvis man ønsker at faa genereret samtlige heltal i omraadet 1 til M - 1.

Vi anvender her proceduren RANDOM INTEGER, der har deklarationen:

```
integer procedure RANDOM INTEGER(n);
value n;
integer n;
begin
   integer new, mod;
   mod := 2796203;
   new := 125×oldrand;
   oldrand := new - mod×entier(new/mod);
   RANDOM INTEGER := n×oldrand/mod
end RANDOM INTEGER from 0 to n;
```

Her benytter vi:

K = 125
M = 2796203

Denne kombination er kopieret direkte fra proceduren random, offentliggjort af L. Hansson (1963).

Proceduren har den formelle parameter, n, og hvert kald af proceduren vil generere et heltal i omraadet fra 0 til n-1. Proceduren arbejder med en global variabel, oldrand, som svarer til R i forklaringen ovenfor. Med andre ord, oldrand vil hele tiden indeholde den øjeblikkelige værdi af det tilfældige heltal i vores række af tal. Værdien af oldrand vil altid ligge i omraadet fra 1 til 2796202.

Naar vi kalder RANDOM INTEGER, beregner vi først:

```
new := 125×oldrand;
```

Dette svarer til beregningen af K×R ovenfor. Derefter beregnes en ny værdi af oldrand som resten ved division af new med modulen (M = mod = 2796203):

```
oldrand := new - mod×entier(new/mod);
```

Vi har nu faaet en ny værdi af oldrand, altsaa i moderrækken af de pseudo-tilfældige heltal, men proceduren skulle jo finde et tilfældigt heltal mellem 0 og n - 1. Dette findes til slut som:

```
RANDOM INTEGER := n×oldrand/mod;
```

For at proceduren RANDOM INTEGER kan virke efter hensigten, maa man have indsat en passende startværdi af oldrand. Dette sker med proceduren START RANDOM.

4.3. Proceduren START RANDOM.

Denne procedure har deklarationen:

```
procedure START RANDOM;
begin
   integer sum, i;
   oldrand := 100001;
   LINE;
   WRITE TEXT(
   ¢<Skriv Deres initialer her og slut med mellemrum¢,
   ¢<Please write your initials here ending with a space¢,
   ¢<Ecrivez vos initiales ici et terminez avec une espace¢,
   ¢<Bitte, schreiben Sie Ihre Initialen hier (Zwischenraum ist Schluss)¢);
   writetext(¢<: ¢);
   sum := 0;
   for i := typechar while i ≠ 0 do sum := 64×sum + i;
   sum := sum - sum:512×512;
   for i := 1 step 1 until sum do RANDOM INTEGER(1)
end START RANDOM;
```

Vi ser, at proceduren først indsætter startværdien 100001 af oldrand. Men derefter kommer der noget mere. Hvis man altid begyndte programmet med

denne startværdi, ville det første sæt tændstikbunker, som programmet valgte, altid være nøjagtigt det samme. Vi vil gerne have en vis variation i spillet, saaledes at operatøren, der er tilstede, naar maskinen spiller med den besøgende, ikke sløves af altid at skulle forklare det samme spil.

Vi giver derfor modspilleren en mulighed for at sætte sit personlige præg paa spillet ved at indstille en ny startværdi af oldrand, som afhænger af ham selv. Dette gør vi ved at køre et vist antal pladser frem i moderrækken af tilfældige tal. Proceduren laver en ny linie og skriver følgende paa skrivemaskinen:

Skriv Deres initialer her og slut med mellemrum:

Proceduren fortsætter nu med de to sætninger:

```
sum := 0;
for i := typechar while i ≠ 0 do sum := 64×sum + i;
```

Hvis man skriver de to bogstaver, j og k, afsluttet af mellemrum, bliver værdien af sum:

$$33 \times 64 + 34 = 2146$$

idet j = 33 og k = 34. Det er nu meningen, at vi skal køre 2146 pladser frem i moderrækken af tilfældige tal. Men da dette tal kan blive ret stort, især hvis modspilleren skriver mere end to bogstaver i initialerne, er det praktisk at reducere tallet modulo 512:

```
sum := sum - sum:512×512;
```

Da 2146:512 = 4, bliver værdien af sum: 2146 - 4×512 = 98.
Fremrykningen af oldrand sker da ved:

```
for i := 1 step 1 until sum do RANDOM INTEGER(1);
```

Proceduren START RANDOM kaldes i begyndelsen af programmet, lige efter SELECT LANGUAGE.

5. SPILPROCEDURER

5.1. Hjælpeprocedurer, GROUP SUM og PRINT GROUPS.

I programmet lader vi tændstikbunkerne repræsentere af talsættet:

```
integer array GROUP[1:10];
```

Der kan saaledes højst være 10 bunker. Endvidere bruger vi de to globale variable:

```
integer g, groups;
```

Her er groups det faktiske antal bunker, som der regnes med i et givet spil, medens g er en tællevariabel, som vi bruger i for-sætninger, der tæller fra 1 til groups.

De to hjælpeprocedurer, GROUP SUM og PRINT GROUPS, har deklarationerne:

```
integer procedure GROUP SUM;
begin
   integer sum;
   sum := 0;
   for g := 1 step 1 until groups do
   sum := sum + GROUP[g];
   GROUP SUM := sum
end GROUP SUM;

procedure PRINT GROUPS;
for g := 1 step 1 until groups do
write(‹-ndd›, GROUP[g]);
```

Proceduren GROUP SUM beregner summen af bunkernes indhold. Naar GROUP SUM er nul, er spillet afsluttet.

Proceduren PRINT GROUPS trykker en linie med antallet af tændstikker i hver bunke.

5.2. Administration af spillet.

Administrationen af spillet er bygget op omkring de to generelle procedurer, GAME og STRATEGY. Proceduren GAME sørger for at trækkene skiftevis udføres af maskinen og modspilleren indtil spillet er slut. Proceduren STRATEGY benyttes kun af maskinen, nemlig til udvælgelse af den rigtige strategi, afhængigt af bunkernes indhold.

5.2.1. Proceduren GAME. Denne har følgende deklaration:

```
procedure GAME(MAN, MACHINE, man next);
boolean MAN, MACHINE, man next;
begin
   boolean finished, man, machine;
   man := machine := finished := false;
   draw := if man next then 0 else -1;
   for draw := draw + 1 while ¬ finished do
   begin
      if man next then man := MAN
      else machine := MACHINE;
      man next := ¬ man next;
      finished := man ∨ machine
   end for draw
end GAME;
```

Vi har de tre formelle parametre:

```
boolean MAN, MACHINE, man next;
```

Her er MAN og MACHINE to procedurer af typen boolean, som administrerer henholdsvis modspillerens og maskinens spil. Den variable, man next, er en global variabel, der er sand, hvis det er modspillerens tur til at trække, og falsk, hvis det er maskinens tur.

Proceduren har de tre lokale variable:

```
boolean finished, man, machine;
```

Her er finished sand, hvis spillet er afsluttet, man er sand, hvis modspilleren har afgjort spillet, og machine er sand, hvis maskinen har afgjort spillet. Proceduren begynder med at sætte alle tre variable til falsk. Den sætter ogsaa en startværdi af den globale variable:

```
integer draw;
```

Hvis modspilleren begynder, sættes draw til nul, og hvis maskinen begynder til -1. GAME øger draw med 1 før hvert enkelttræk udføres, og draw vil derfor være lige, hver gang maskinen skal trække.

Kernen i proceduren er for-sætningen:

```
for draw := draw + 1 while -, finished do
```

hvori trækkene tælles frem, indtil spillet er afgjort. I selve for-sætningen udføres først:

```
if man next then man := MAN
else machine := MACHINE;
```

Vi husker, at MAN og MACHINE er de to procedurer, der administrerer spillet af henholdsvis modspiller og maskine. Det er procedurer af typen boolean, som sættes til sand, saa snart spillet er afgjort. Den næste sætning i forsætningen er:

```
man next := -, man next;
```

som holder regnskab med hvis tur, det er. Endelig undersøges det, om spillet er afsluttet med:

```
finished := man ∨ machine;
```

Den variable, finished, bliver altsaa sand, saa snart spillet er afgjort, enten under udførelsen af MAN (hvis værdi er gemt i: man) eller af MACHINE (hvis værdi er gemt i: machine).

Vi ser, at proceduren GAME er af helt generel natur, idet den kan bruges i alle tilfælde, hvor to spillere skiftes til at trække, og hvor spillet afgøres under een af spillernes træk.

5.2.2. Proceduren STRATEGY. Denne procedure har deklarationen:

```
procedure STRATEGY(condition, TEST, best group, take);
boolean condition;
procedure TEST;
integer best group, take;
if condition ∧ ¬ settled then
begin
   TEST;
   actual group := best group;
   removed := take;
   settled := true
end if and STRATEGY;
```

Denne procedure kaldes tre gange af proceduren MACHINE, nemlig for at undersøge hvilken af de tre mulige strategier, maskinen skal følge og for at beregne, hvor mange tændstikker, der skal fjernes fra hvilken bunke. Proceduren MAN benytter naturligvis ikke proceduren STRATEGY, da vi ikke kan programmere modspillerens opførsel.

Der er fire formelle parametre i STRATEGY:

boolean condition. Hvis denne betingelse er opfyldt, skal denne strategi følges.

procedure TEST. Hvis condition er sand, kaldes proceduren TEST, som iværksætter en nærmere undersøgelse af forholdende.

integer best group. Nummeret paa den gruppe, fra hvilken der skal fjernes tændstikker. Beregnes af TEST.

integer take. Antallet af tændstikker, som skal fjernes. Beregnes ogsaa af TEST.

Proceduren arbejder med følgende globale variable:

boolean settled. Denne sættes til falsk af MACHINE, inden de tre strategier undersøges. Naar STRATEGY har fundet den rigtige strategi, bliver settled sat til sand.

integer actual group, removed. Dette er nummeret paa den bunke, der skal fjernes tændstikker fra, samt antallet, der skal fjernes. Disse to variable er globale, idet de benyttes baade af MAN og af MACHINE.

Procedurens virkemaade er ganske simpel. Hvis condition er sand, og settled er falsk, udføres TEST, actual group sættes lig best group, removed sættes lig take, og settled sættes til sand.

Vi ser, at den egentlige analyse udføres af TEST-procedurerne, der beskrives i det følgende afsnit.

5.3. Prøveprocedurer.

Der er tre TEST-procedurer: SAFE, NORMAL TEST og SPECIAL TEST.

Proceduren SAFE foretager den vigtige undersøgelse af om positionen er sikker eller usikker og optæller forskellige hjælpestørrelser. Hvis positionen er sikker efter at modspilleren har trukket, d.v.s. usikker for maskinen, anvendes den simple strategi, at maskinen blot fjerner een tændstik fra den største bunke.

De to andre procedurer: NORMAL TEST og SPECIAL TEST, anvendes, naar maskinen er i en gunstig stilling, d.v.s. den kan gøre et træk, som fører til en position, der er sikker for den. Normalt anvendes NORMAL TEST, medens SPECIAL TEST anvendes, naar den, der fjerner den sidste tændstik, taber, og bunkerne er ved at være tomme (højst een bunke med mere end een tændstik).

5.3.1. Proceduren SAFE. Deklarationen herfor er:

```
boolean procedure SAFE;
begin
   pack(boole sum, 0, 39, 0);
   ones := not one := 0;
   biggest group := 1;
   max group := GROUP[1];
   for g := 1 step 1 until groups do
   begin
      number := GROUP[g];
      if number > 1 then not one := not one + 1;
      if number = 1 then ones := ones + 1;
      if number > max group then
      begin
         max group := number;
         biggest group := g
      end if bigger;
      pack(group boole, 0, 39, number);
```

```
      boole sum := boole sum ≠ group boole
   end for g;
   split(boole sum, 0, 39, number);
   odd ones := ones:2×2 ≠ ones;
   SAFE := if -, last wins ∧ not one = 0 then odd ones else number = 0
end SAFE;
```

Der er ingen formelle parametre og alle variable er globale. Det er en boolean procedure, og den skal antage værdien sand, hvis modspilleren har efterladt en position, der er sikker for ham, d.v.s. usikker for maskinen.

Som vist i kapitel 2 skal man summere de enkelte bunkers indhold af enere, toere, firere, o.s.v. modulo 2 for at afgøre om positionen er sikker eller usikker. Denne summation foretages i den globale variable:

```
boolean boole sum;
```

som i virkeligheden skal opfattes som et boolean array, idet bit 39 af boole sum skal give summen af enere, bit 38 summen af toere, bit 37 summen af firere, o.s.v. altsammen modulo 2. Sætningen:

```
pack(boole sum, 0, 39, 0);
```

bevirker, at alle bits i ordet boole sum bliver sat til nul.

Inden summationen begynder, træffes indledende forberedelser til afprøvning af andre ting. Vi bruger her følgende globale variable:

integer ones. Antallet af bunker med netop een tændstik i.

integer not one. Antallet af bunker med mere end een tændstik i.

integer biggest group. Nummeret paa den bunke, som indeholder flest tændstikker.

integer max group. Antallet af tændstikker i den største bunke.

Summationen og de andre afprøvninger sker med for-sætningen:

```
for g := 1 step 1 until groups do
```

For hvert gennemløb gemmes antallet af tændstikker i den aktuelle bunke i den variable: number:

```
number := GROUP[g];
```

Tællingen af antallet af bunker med een tændstik eller med flere tændstikker, samt kontrollen af den største bunke kræver ingen nærmere forklaring. Summationen modulo 2 foregaar ved at vi først overfører heltallet: number til en logisk variabel:

```
boolean group boole;
```

Dette sker med sætningen:

```
pack(group boole, 0, 39, number);
```

Den variable, group boole, indeholder nu tallet: number i form af enerne i bit 39, toerne i bit 38, firerne i bit 37, o.s.v. Endelig sker summationen med:

```
boole sum := boole sum = group boole;
```

Summationen sker samtidigt for alle bits i de to variable. Hvis bit nr. N er ens i boole sum og group boole (altsaa to enere eller to nuller), vil bit nr. N i boole sum blive sat til sand, d.v.s. til nul. Er bit nr. N derimod forskellige i boole sum og group boole (altsaa eet-nul eller nul-eet), vil bit nr. N i boole sum blive sat til falsk, d.v.s. til 1. Dette svarer til additionsreglerne:

```
          0   1   1   0
          0   1   0   1
          -   -   -   -
Sum:      0   0   1   1
```

Man ser, at dette netop er reglerne for addition modulo 2 af enerne, toerne, firerne, o.s.v.

Naar for-sætningen er slut, overfører vi summen fra den logiske variable (boole sum) til en heltalsvariabel (number) ved hjælp af sætningen:

```
spilt(boole sum, 0, 39, number);
```

Endvidere bruger vi en logisk variabel:

```
boolean odd ones;
```

til at angive om antallet af bunker med netop een tændstik er ulige:

```
odd ones := ones:2×2 ≠ ones;
```

Endelig kan vi nu beregne værdien af SAFE:

```
SAFE := if -, last wins ∧ not one = 0 then odd ones else number = 0;
```

Den globale variable:

```
boolean last wins;
```

er ved spillets begyndelse blevet sat til sand, hvis den som tager sidste tændstik har vundet, og til falsk, hvis den som tager sidste tændstik, taber. I kapitel 2 saa vi, at hvis den som tager sidste tændstik taber, og ingen bunke indeholder mere end een tændstik, skal en sikker position bestaa af et ulige antal bunker med netop een tændstik. I alle andre tilfælde er en sikker position een som har summen modulo 2 til nul, d.v.s. number = 0.

5.3.2. Proceduren NORMAL TEST. Denne procedure har deklarationen:

```
procedure NORMAL TEST;
begin
   split(boole sum, 0, 39, number);
   factor := 1;
   nmax := 40;
   for nmax := nmax - 1 while number:factor ≥ 1 do
   factor := 2×factor;
   for g := 1 step 1 until groups do
   begin
      if number = GROUP[g] then
      begin
         best group := g;
         max possible := number;
         go to EX
      end if all removed
      else
      begin
```

```
            pack(group boole, 0, 39, GROUP[g]);
            split(group boole, nmax + 1, nmax + 1, bit);
            if bit = 1 then best group := g
         end if not all removed
      end for g;
      max possible := 0;
      factor := 1;
      pack(group boole, 0, 39, GROUP[best group]);
      for n := 39 step -1 until nmax do
      begin
         split(boole sum, n, n, bit);
         if bit = 1 then
         begin
            split(group boole, n, n, bit);
            max possible := max possible + (if bit = 1 then factor else - factor)
         end if bit = 1;
         factor := 2×factor
      end for n;
EX:   end NORMAL TEST;
```

Proceduren NORMAL TEST bliver kaldt, naar modspilleren har efterladt en usikker position og maskinen skal gøre stillingen sikker ved at tilvejebringe en summation modulo 2, som er nul. Naar NORMAL TEST kaldes, har proceduren SAFE lige været kaldt, og den variable, boole sum, indeholder derfor resultatet af summationen. NORMAL TEST overfører først summen fra den logiske variable til en heltalsvariabel, number:

```
split(boole sum, 0, 39, number);
```

Inden vi begynder at undersøge de enkelte bunker, skal vi bestemme den største to-potens, som er indeholdt i tallet number. Dette sker med sætningerne:

```
factor := 1;
nmax := 40;
for nmax := nmax - 1 while number:factor ≥ 1 do
factor := 2×factor;
```

Efter gennemløbet heraf vil den højeste to-potens i tallet number være givet ved bit nr. nmax + 1.

Som eksempel kan vi tage number = 13, der i totalsystemet skrives:1101. Da enheden placeres i bit 39, har vi:

bit nr.	36	37	38	39
	1	1	0	1

Efter beregningen er factor = 16 og nmax = 35, d.v.s. den højeste to-potens i number er placeret i position 36.

Nu gennemløber proceduren for-sætningen:

```
for g := 1 step 1 until groups do
```

For hvert gennemløb prøves det først, om number er lig med antallet af tændstikker i den paagældende bunke. Naar en saadan bunke er fundet, sætter proceduren den variable, best group, lig med den aktuelle gruppe, g, og den variable, max possible, sættes lig med number. Derefter hoppes direkte til slutningen, EX, af proceduren.

Hvis number ikke er lig med antallet af tændstikker i den foreliggende bunke, undersøger proceduren, om bunken indeholder den samme to-potens, som svarer til bit nr. nmax + 1, altsaa om bunken indeholder den største to-potens, som findes i summen. Først overføres tændstikantallet, GROUP[g], til den logiske variable, group boole:

```
pack(group boole, 0, 39, GROUP[g]);
```

Derefter overføres bit nr. nmax + 1 til heltalsvariablen, bit:

```
split(group boole, nmax + 1, nmax + 1, bit);
```

Værdien af bit bliver altsaa 0 eller 1. Hvis den er 1, betyder det, at vi kan bruge denne bunke til at fjerne tændstikker fra:

```
if bit = 1 then best group := g;
```

Hvis proceduren kommer igennem for-sætningen uden at faa udhop til EX, fordi ingen af bunkerne er eksakt lig med number, vil værdien af best group være nummeret paa den sidste bunke, der har en to-potens fælles med den

højeste to-potens i number. Vi mangler da blot at bestemme hvormange tændstikker, der skal fjernes fra denne bunke. Beregningen heraf sker ved sammenligning af to-potensindholdet i summen (findes i boole sum) og i den udpegede bunke. Bunkeindholdet anbringes i den logiske variable, group boole, med sætningen:

```
pack(group boole, 0, 39, GROUP[best group]);
```

Først sættes det antal, der skal fjernes, til nul:

```
max possible := 0;
```

Ligeledes sætter vi:

```
factor := 1;
```

Derefter gennemløbes for-sætningen:

```
for n := 39 step -1 until nmax do
```

For hvert gennemløb undersøger vi, om bit nr. n er 1 i summen:

```
split(boole sum, n, n, bit);
```

Her bliver bit = 1, hvis bit nr. n er en ener i summen. Vi skal da enten fjerne denne to-potens fra bunken, hvis den er der i forvejen, eller tilføje den, hvis den ikke er der. Vi prøver derfor, om bunken har en to-potens i bit nr. n:

```
split(group boole, n, n, bit);
```

Hvis bit = 1, skal to-potensen fjernes, ellers tilføjes:

```
max possible := max possible +
(if bit = 1 then factor else - factor);
```

Vi husker, at factor blev først sat til 1, altsaa værdien af den to-potens, som staar i bit 39. Sidst i for-sætningen ganges factor op med to:

```
factor := 2×factor;
```

Naar for-sætningen er slut, indeholder max possible det antal, der skal fjernes fra best group.

5.3.3. Proceduren SPECIAL TEST. Deklarationen herfor er:

```
procedure SPECIAL TEST;
begin
   for best group := 1 step 1 until groups do
   begin
      number := GROUP[best group];
      if number > 1 then
      begin
         possible := if odd ones then number else number - 1;
         go to EX
      end if number > 1;
      if not one = 0 ∧ number = 1 then
      begin
         possible := 1;
         go to EX
      end if only ones
   end for best group;
EX: end SPECIAL TEST;
```

SPECIAL TEST anvendes i det specielle tilfælde, hvor vi spiller det omvendte spil og der højst er een bunke tilbage med mere end een tændstik.

Proceduren bestaar af en for-sætning:

```
for best group := 1 step 1 until groups do
```

som gennemprøver alle bunkerne. For hvert gennemløb overfører vi først bunkeantallet til den simple variable, number:

```
number := GROUP[best group];
```

Først undersøger vi, om number > 1. Hvis dette er tilfældet, er det denne bunke der skal fjernes, enten helt eller paa nær 1. Vi skal nemlig sørge for, at der bliver et ulige antal bunker med 1 tændstik tilbage. Fra det tidligere kald af proceduren SAFE er maskinen klar over, hvordan situationen er, idet denne information er lagret i den logiske variable, odd

ones, som er sand , hvis der er et ulige antal bunker med netop een tændstik. Det antal tændstikker, der skal fjernes, bliver derfor:

```
possible := if odd ones then number else number - 1;
```

Hvis maskinen saaledes har fundet en bunke med mere end een tændstik, behøver de resterende bunker ikke at undersøges, og der hoppes til EX i udgangen af proceduren.

Hvis bunken kun indeholder een tændstik, har den kun interesse for os, hvis der slet ingen findes med mere end een tændstik. Saa kan maskinen nemlig lige saa godt fjerne denne bunke. Vi ved fra SAFE-proceduren, hvor mange bunker der findes med mere end een tændstik, det er gemt i den variable, not one. Vi kan derfor skrive:

```
if not one = 0 ∧ number = 1 then
begin
   possible := 1;
   go to EX
end if only ones;
```

Efter udhoppet staar bunkenummeret i den variable, best group, og det antal som skal fjernes, i den variable, possible. Bemærk iøvrigt, at man aldrig vil komme helt igennem for-sætningen.

5.4. Spillernes procedurer.

Efter at de strategiske og beregningsmæssige detailler nu er gennemgaaet, er det ret let at forstaa de egentlige procedurer, MAN og MACHINE, som administrerer de to spilleres træk.

5.4.1. Proceduren MAN. Deklarationen er vist paa de følgende sider:

```
boolean procedure MAN;
begin
AA: actual group := if first then
      ASK NUMBER(
   {<Skriv nummeret paa den bunke, fra hvilken De vil fjerne tændstikker},
   {<Write the number of the group from which you will remove matches},
   {<Ecrivez le numero du groupe dont vous voulez prendre des allumettes},
   {<Schreiben Sie die Nummer des Haufens, von dem Sie Hoelzer wegnehmen})
   else
   ASK NUMBER(
   {<Vælg Deres bunke}, {<Choose your group},
   {<Choisissez votre groupe}, {<Bitte Haufen waehlen});
   if actual group < 1 ∨ actual group > groups then
   begin
      LINE;
      WRITE TEXT(
      {<Undskyld, men tallet er for },
      {<Sorry, but the number is too },
      {<Pardon, le nombre est trop },
      {<Die Zahl ist zu });
      if actual group < 1 then
      WRITE TEXT({<lille.}, {<low.}, {<petit.}, {<klein.})
      else
      WRITE TEXT({<stort.}, {<high.}, {<grand.}, {<gross.});
      go to AA
   end if out of range;
   if GROUP[actual group] = 0 then
   begin
      LINE;
      WRITE TEXT(
      {<Undskyld, men denne bunke er tom.},
      {<Sorry, but this group is empty.},
      {<Pardon, ce groupe est vide.},
      {<Dieser Haufen ist leer.});
      go to AA
   end if empty group;
comment
```

```
   removed := GROUP[actual group];
   if removed ≠ 1 then
   begin
BB:   removed := if first then
      ASK NUMBER(
      {<Og antallet af tændstikker, De vil fjerne},
      {<And the number of matches you want to remove},
      {<Et le nombre d allumettes que vous prenez},
      {<Und die Anzahl der Zuendhoelzer, die Sie wegnehmen})
      else
      ASK NUMBER(
      {<Og antallet}, {<And the number},
      {<Et le nombre}, {<Und die Anzahl});
      if removed < 1 then
      begin
         LINE;
         WRITE TEXT(
         {<De skal fjerne mindst een tændstik.},
         {<You must remove at least one match.},
         {<Il faut prendre au moins une allumette.},
         {<Sie muessen mindestens ein Zuendholz wegnehmen.});
         go to BB
      end if removed < 1;
      if removed > GROUP[actual group] then
      begin
         LINE;
         WRITE TEXT(
         {<Saa mange er der ikke i bunken. De fjerner altsaa hele bunken.},
         {<There are not so many in the group. You are removing the whole group},
         {<Il n y en pas tant. Vous prenez donc tout le groupe.},
         {<So viele sind da nicht. Sie nehmen also den ganzen Haufen.});
         removed := GROUP[actual group]
      end if too many
   end if more than one;
comment
```

```
GROUP[actual group] := GROUP[actual group] - removed;
first := false;
newsafe := SAFE;
if draw = 1 then
begin
   LINE;
   writechar(29);
   if newsafe then
   WRITE TEXT(
   ¢<Hvis De spiller rigtigt, kan De vinde dette spil.¢,
   ¢<If you play correctly, you may win this game.¢,
   ¢<Si vous jouez correctement, vous pouvez gagner ce jeu.¢,
   ¢<Wenn Sie richtig spielen, koennen Sie dieses Spiel gewinnen.¢)
   else
   WRITE TEXT(
   ¢<De kan ikke vinde dette spil.¢,
   ¢<You cannot win this game.¢,
   ¢<Vous ne pouvez pas gagner ce jeu.¢,
   ¢<Sie koennen dieses Spiel nicht gewinnen.¢);
   writechar(62)
end if draw = 1
else
if oldsafe ∧ ¬ newsafe then
begin
   LINE;
   writechar(29);
   WRITE TEXT(
   ¢<Det var forkert. Nu kan De ikke vinde.¢,
   ¢<That was wrong. You cannot win now.¢,
   ¢<Voila une erreur. Maintenant vous ne pouvez pas gagner.¢,
   ¢<Das war falsch. Jetzt koennen Sie dieses Spiel nicht mehr gewinnen.¢);
   writechar(62)
end if blunder;
oldsafe := newsafe;
comment
```

```
   if ¬ newsafe ∧ draw:3×3 = draw then
   begin
      if QUESTION(
      {<Giver De fortabt},
      {<Do you want to give up the game},
      {<Vous vous declarez vaincu},
      {<Wollen Sie aufgeben}) then
      begin
         MAN := true;
         man wins := false;
         go to EX
      end give up
   end question;
   MAN := GROUP SUM = 0;
   man wins := last wins;
EX: end MAN;
```

Her begynder maskinen med at spørge om nummeret paa den bunke, modspilleren vil operere paa. Det sker ved hjælp af en sætning af formen:

```
actual group := ASK NUMBER(.....
```

Med proceduren ASK NUMBER kan maskinen spørge paa et af de fire tilladte sprog. Paa dansk bruges tekststrengen:

Skriv nummeret paa den bunke, fra hvilken De vil fjerne tændstikker:

Da maskinen vil stille dette spørgsmaal mange gange i løbet af spillet, kan det betale sig at bruge et kortere spørgsmaal de følgende gange. Vi bruger derfor en logisk variabel, first, som sættes til sand, naar spillet begynder. Naar first er falsk, lyder spørgsmaalet ovenfor:

```
Vælg Deres bunke:
```

Modspilleren skal nu skrive bunkenummeret paa skrivemaskinen. Maskinen prøver derefter, om det opgivne bunkenummer ligger i omraadet fra 1 til groups, der er det samlede antal grupper. Hvis nummeret ligger udenfor dette omraade, skriver maskinen at tallet er for stort eller for lille og spørger igen om nummeret.

Derefter prøver maskinen, om den opgivne bunke er tom. Er den det, skriver maskinen:

Undskyld, men denne bunke er tom.

og spørger igen om nummeret.

Nu skal maskinen spørge om, hvor mange tændstikker, der skal fjernes fra bunken. Det sker med en sætning af formen:

removed := ASK NUMBER(.....

Først sætter den dog removed lig med det samlede antal tændstikker i den valgte bunke:

removed := GROUP[actual group];

Hvis removed = 1, spørger maskinen ikke om hvormange, der skal fjernes. Spørgsmaalet om tændstikantallet stilles først som en lang tekst:

Og antallet af tændstikker, De vil fjerne:

og de følgende gange i en kortere form:

Og antallet:

Maskinen kontrollerer derefter, at det opgivne antal er ≥ 1. Hvis ikke, skriver den:

De skal fjerne mindst een tændstik.

og spørger igen. Hvis antallet derimod er større end antallet af tændstikker i bunken, skriver maskinen:

Saa mange er der ikke i bunken. De fjerner altsaa hele bunken.

og sætter removed lig med hele bunkens indhold.

Nu sker den egentlige fjernelse med:

```
GROUP[actual group] := GROUP[actual group] - removed;
```

Nu sættes first til falsk, saaledes at vi i næste træk faar de kortere spørgsmaal.

Proceduren er indrettet saaledes, at maskinen paa dette sted kan give modspilleren oplysning om vinderchancerne. Den undersøger først, om modspilleren har bragt sig en sikker position:

```
newsafe := SAFE;
```

newsafe vil være sand, hvis modspilleren nu er i en sikker position.

Hvis det er modspillerens første træk, vil maskinen udskrive en bemærkning med rødt paa skrivemaskinen. Hvis newsafe er sand, skriver den:

Hvis De spiller rigtigt, kan De vinde dette spil.

Hvis derimod newsafe er falsk, skriver den:

De kan ikke vinde dette spil.

Programmet gemmer sikkerhedssituationen fra forrige træk i den logiske variable, oldsafe. Hvis det ikke er modspillerens første træk, sammenligner maskinen oldsafe og newsafe. Hvis oldsafe er sand og newsafe falsk, har modspilleren aabenbart gjort en bommert, og maskinen skriver (ligeledes med rødt):

Det var forkert. Nu kan De ikke vinde.

Derefter sættes oldsafe lig med newsafe.

Hvis maskinen er i en sikker position (newsafe er falsk) vil maskinen i hvert tredje af modspillerens træk spørge ham, om han giver fortabt. Det sker med betingelsen:

```
if newsafe ∧ draw:3×3 = draw then
```

Svarer modspilleren ja, sætter maskinen MAN til sand, som tegn paa at spillet nu er afgjort. Endvidere sætter den den logiske variable, man wins, til falsk, som tegn paa at modspilleren har tabt. Endelig hoppes til slutningen af proceduren.

hvis modspilleren ikke giver forlabt, eller ikke bliver spurgt derom, nemlig maskinen om spillet er afgjort ved at sætte:

MAN := GROUP SUM = 0;

og den sætter ogsaa:

man wins := last wins;

Hvis spillet er afgjort, har modspilleren vundet, hvis det er den, som tager sidste tændstik, der vinder.

Efter udhoppet fra MAN, vendes tilbage til proceduren GAME, hvor MAN nu er kaldt.

5.4.2. Proceduren MACHINE. Deklarationen herfor er følgende:

```
boolean procedure MACHINE;
begin
    settled := false;
    STRATEGY(SAFE, NONE, biggest group, 1);
    comment That was the strategy if SAFE for man, i.e. unsafe for machine;
    STRATEGY(-, last wins => not one > 1, NORMAL TEST, best group, max possible);
    comment That was the strategy if last wins or (last looses and more than one
    group containing more than one);
    STRATEGY(true, SPECIAL TEST, best group, possible);
    comment That was the strategy if last looses and not more than one group
    with more than one;
    GROUP[actual group] := GROUP[actual group] - removed;
    LINE;
    WRITE TEXT(
    ‹Jeg fjerner nu›, ‹I now remove›,
    ‹Je prends›, ‹Ich nehme jetzt›);
    if GROUP[actual group] = 0 then WRITE TEXT(
    ‹ hele bunke nr.›, ‹ the entire group no.›,
    ‹ tout le groupe no.›, ‹ den ganzen Haufen Nr.›)
    else
    begin
```

```
      write(‹-ndd›, removed);
      WRITE TEXT(
      ‹< fra bunke nr.›, ‹< from group no.›,
      ‹< du groupe no.›, ‹< von Haufen no.›)
   end if not empty;
   write(‹-ndd›, actual group);
   WRITE TEXT(
   ‹<. Bunkerne indeholder nu:›,
   ‹<. The groups now contain:›,
   ‹<. Les groupes contiennent:›,
   ‹<. Die Haufen enthalten jetzt:›);
   LINE;
   LINE;
   PRINT GROUPS;
   MACHINE := GROUP SUM = 0;
   man wins := -, last wins
end MACHINE;
```

Proceduren MACHINE er væsentlig kortere end MAN, da det ikke er nødvendigt at stille spørgsmaal og kontrollere svarene. Iøvrigt er de strategiske beregninger jo henlagt til andre procedurer, der kaldes af MACHINE.

Først sættes den logiske variable, settled, til falsk. Den bliver rettet til sand, saa snart en rigtig strategi er fundet. Der laves tre kald af proceduren STRATEGY.

I kald nr. 1 er betingelsen for at adlyde strategi nr. 1, at SAFE er sand, d.v.s. situationen er sikker for modspilleren eller usikker for maskinen. Her fjerner maskinen een tændstik fra den største bunke. Paa denne maade bliver spillet meget langsomt, og modspilleren har en risiko for at trække forkert, især hvis det er rent tilfældigt, at han har gjort det første træk rigtigt. Bemærk, at der ikke kræves nogen speciel TEST-procedure i strategi nr. 1, fordi prøven er indeholdt i SAFE. Vi bruger derfor en helt tom procedure, NONE, som TEST-procedure her:

```
procedure NONE;;
```

Den tomme procedure staar imellem de to semikoloner.

I kald nr. 2 er betingelsen for at adlyde strategi nr. 2 følgende. Enten spiller vi normalt spil, eller ogsaa skal der være mere end een bunke med mere end een tændstik. Betingelsen kan skrives saaledes:

- last wins => not one > 1

idet vi husker, at a => b er sand for alle værdier af a og b undtagen a sand, b falsk. For denne strategi beregnes bunkenummeret og antallet som vist i proceduren NORMAL TEST.

Endelig vil det tredje kald af STRATEGY tage sig af den sidste mulighed, hvor vi bruger proceduren SPECIAL TEST.

Saa snart een af betingelserne i kaldet af STRATEGY er opfyldt, vil STRATEGY sætte settled til sand, saaledes at der kun følges een strategi.

Efter de tre kald af STRATEGY er bunkenummeret indeholdt i den variable, actual group, og antallet i den variable, removed. Maskinen effektuerer nu fjernelsen:

```
GROUP[actual group] := GROUP[actual group] - removed;
```

og skriver talværdierne af actual group og removed med passende tekstforklaring. Hvis den berørte gruppe bliver helt tom, skriver den, at den har fjernet hele bunken uden at opgive antallet. Endelig skrives:

```
Bunkerne indeholder nu:
```

og proceduren PRINT GROUPS kaldes til udskrift af bunkeindholdet.

Ligesom ved afslutningen af MAN beregner vi her, om spillet er slut:

```
MACHINE := GROUP SUM = 0;
```

og vi sætter:

```
man wins := -, last wins;
```

6. PROGRAMMET

6.1. Programmets opbygning.

Selve ALGOL-programmet er vist i afsnit 8.1. Det er let at forstaa, fordi de vanskelige ting er gemt i procedurerne.

Først deklareres et vist antal boolean og integer variable, samt et integer array GROUP[1:10], som repræsenterer tændstikbunkerne. Derefter deklareres de 18 procedurer, som er omtalt i det foregaaende.

Derefter begynder det egentlige program med at sætte det resterende linieantal paa siden til 69. Proceduren **SELECT LANGUAGE** kaldes til valg af sproget, og START RANDOM til indsættelse af startværdi af moderrækken af tilfældige tal. Programmet udskriver saa to tekstlinier med programmets navn og med begyndelsen til forklaringen:

Program DEMON-3
Tændstikspillet NIM. Vi vælger først nogle tilfældige bunker af tændstikker:

Programmet sætter nu bunkeantallet til 3, vælger et tilfældigt indhold af hver bunke mellem 1 og 6, og udskriver indholdet saaledes:

Bunke nr:	1	2	3
Antal tændstikker:	5	2	5

Saa skriver programmet tre tekstlinier med vejledningen:

Vi skal nu skiftevis fjerne tændstikker fra bunkerne. Den, som fjerner den eller de sidste tændstikker, har vundet. Kun een bunke maa røres i hvert træk, og man skal fjerne mindst een tændstik fra den bunke.

Bemærk, hvorledes vi maa programmere:

```
linerest := linerest - 2;
```

fordi der er to vognretur i tekstlinierne, som ikke bliver talt med.

Nu indsætter programmet startværdier af de tre logiske variable:

```
first := last wins := true;
man next := false;
```

At first er sand, betyder, at vi faar de lange tekstforklaringer i proceduren MAN. At last wins er sand, betyder at den, der tager sidste tændstik, vinder spillet. Endelig betyder man next falsk, at maskinen begynder spillet. Det skriver den ogsaa:

Jeg begynder:

Hele spillet foregaar saa ved et kald af proceduren GAME med de to procedurer: MAN og MACHINE som aktuelle parametre.

Naar maskinen kommer tilbage fra GAME, er spillet afgjort, og udfaldet er gemt som den logiske variable: man wins. Er denne sand, skriver maskinen:

De har vundet. Tillykke.

ellers skriver den:

De har tabt.

Nu tilbyder maskinen et nyt spil med modspllleren ved et kald af proceduren QUESTION. Den spørger:

Skal vi prøve igen:

Hvis man svare nej, er programmet slut. Svarer man ja, stiller maskinen en række yderligere spørgsmaal. Først spørger den om, hvor mange bunker, der skal bruges:

```
groups := ASK NUMBER(‹Hvor mange bunker›), o.s.v.
```

Hvis modspilleren vælger groups < 2 eller groups > 10, retter programmet antallet henholdsvis op til 2 eller ned til 10 og skriver en bemærkning om, at dette er bedre.

Programmet spørger saa, om modspilleren ønsker selv at bestemme antallet af tændstikker i de enkelte bunker. Hvis han svarer ja, indlæses tallene med:

```
for g := 1 step 1 until groups do GROUP[g] := typein;
```

Hvis han ikke ønsker at bestemme antallet, skriver maskinen:

Her er bunkerne:

og beregner antallene med for-sætningen:

```
for g := 1 step 1 until groups do
group[g] := 1 + RANDOM INTEGER(2↑groups - 2);
```

Endelig udskrives bunkeindholdet med proceduren PRINT GROUPS.

Saa spørger maskinen, om modspilleren ønsker at begynde:

```
man next := QUESTION(⟨<Vil De begynde⟩, o.s.v.
```

Det sidste spørgsmaal lyder:

```
last wins := QUESTION(
⟨<Skal den, som tager sidste tændstik, vinde⟩, o.s.v.
```

Svarer han ja, faar vi det normale spil, medens nej giver det omvendte spil. Der hoppes da tilbage til AA, hvor GAME kaldes igen.

6.2. Eksempel paa et spil.

I afsnit 8.2. er vist et typisk eksempel paa en serie af spil.

I det første spil faar vi bunkerne 5, 2 og 5. Da maskinen begynder og positionen er usikker, vinder den.

I andet spil faar vi bunkerne 2, 2 og 2. Positionen er ogsaa usikker og modspilleren vælger at begynde. Han fjerner den sidste bunke, hvorved situationen (2, 2, 0) bliver sikker for ham. Maskinen fjerner kun een tændstik ad gangen og slutter med at tabe.

I tredje spil faar vi bunkerne 3, 3 og 6. Modspilleren begynder ogsaa korrekt her med at fjerne hele tredje bunke. Vi faar maskinens kommentar:

Hvis De spiller rigtigt, kan De vinde dette spil.

Efter maskinens træk er bunkerne : 2, 3, 0. Nu skulle modspilleren fjerne 1 fra bunke 2, men han tager to tændstikker. Dette kommenteres straks af maskinen:

Det var forkert, nu kan De ikke vinde.

og maskinen vinder hurtigt. Bemærk her kontrollen paa valg af tomme eller ikke-eksisterende bunker.

I fjerde spil vælger modspilleren bunkerne: 4, 1 og 3. Han vælger selv at begynde, samt at spille omvendt spil. Han følger den korrekte strategi og vinder.

Endelig har vi det femte spil, hvor der er 6 bunker med indholdet:

4 29 29 49 27 54

Maskinen begynder, og der spilles normalt spil. Nedskrevet i totalsystemet ser bunkeindholdene og summen modulo 2 saaledes ud:

	Indhold	
Bunke nr:	Tital	Total
1	4	100
2	29	11101
3	29	11101
4	49	110001
5	27	11011
6	54	110110
	Sum	011000

Summen er 24, opfattet som tal. Da ingen bunke indeholder nøjagtigt 24 tændstikker, fjerner maskinen i stedet et mindre antal fra den sidste bunke, som har en to-potens sammenfaldende med den højeste to-potens i summen, her 16. Det er bunke nr. 6, hvor maskinen fjerner 16 og tilføjer 8, altsaa fjerner 8. Modspilleren opgiver hurtigt spillet.

7. REFERENCER

Bouton, C.L.: Nim, a game with a complete mathematical theory. Annals of Mathematics, Series 2, Vol. 3, pag. 35-39 (1901 - 1902).

Hansson, L.: Random, Risø procedure SA-51 (1963).

Kjær, J: Programmering af kemiske beregninger, Bind 1, ALGOL, Haldor Topsøe (1964).

8. APPENDIKS

8.1. Algolprogrammet.

```
program DEMON-3, NIM,
begin
    boolean first, man wins, last wins, newsafe, oldsafe, boole sum, group boole,
    man next, decided, odd ones;
    integer interest, librand, lang, groups, actual group, removed, draw, ones,
    [illegible], max group, number, biggest group, best group, max possible, possible,
    [illegible], comb, n, bit, g;
    integer array GROUP[1:10];
    comment library ASK NUMBER;
    comment library GAME;
    comment library GROUP SUM;
    comment library LINE;
    comment library MACHINE;
    comment library MAN;
    comment library NEW PAGE;
    procedure NONE;;
    comment library NORMAL TEST;
    comment library PRINT GROUPS;
    comment library QUESTION;
    comment library RANDOM INTEGER;
    comment library SAFE;
    comment library SELECT LANGUAGE;
    comment library SPECIAL TEST;
    comment library START RANDOM;
    comment library STRATEGY;
    comment library WRITE TEXT;
    interest := 69;
    SELECT LANGUAGE;
    LINE;
    START RANDOM;
    LINE;
    WRITE TEXT(
    ¢<PROGRAM DEMON-3>¢, ¢<PROGRAM DEMON-3>¢,
    ¢<PROGRAMME DEMON-3>¢, ¢<PROGRAMM DEMON-3>¢);
comment
```

```
LINE;
WRITE TEXT(
{<Tændstikspillet NIM. Vi vælger først nogle tilfældige bunker af tændstikker:},
{<The match game NIM. We first select some random groups of matches:},
{<Le jeu de NIM.  Nous choisissons d abord quelques groupes d allumettes:},
{<Das Spiel NIM. Wir waehlen zuerst einigen Haufen von Zuendhoelzern:});
LINE;
LINE;
WRITE TEXT(
{<Bunke nr.:          }, {<Group no.:          },
{<Groupe no.:          }, {<Haufen Nr.:      });
groups := 3;
for g := 1 step 1 until groups do write({-ndd}, g);
LINE;
WRITE TEXT(
{<Antal tændstikker: }, {<Number of matches: },
{<Nombre d allumettes: }, {<Anzahl Hoelzer: });
for g := 1 step 1 until groups do
GROUP[g] := 1 + RANDOM INTEGER(6);
PRINT GROUPS;
LINE;
LINE;
WRITE TEXT(
{<Vi skal nu skiftevis fjerne tændstikker fra bunkerne. Den, som fjerner den
eller de sidste tændstikker, har vundet. Kun een bunke maa røres i hvert
træk, og man skal fjerne mindst een tændstik fra den bunke.},
{<We shall now alternatingly remove matches from the groups. He who removes
the last match (or matches) has won. In each move only one group must be
touched, and at least one match must be removed from that group.},
{<Nous allons alternativement enlever des allumettes des groupes. Celui qui
prend la derniere allumette est vainqueur. Dans chaque coup il faut toucher
un groupe seulement et prendre au moins une allumette.},
{<Wir sollen jetzt abwechselnd Zuendhoelzer von den Haufen wegnehmen. Derjenige,
der das letzte nimmt, hat gewonnen. In jedem Zug darf man nur einen Haufen
ruehren, und man soll mindestens ein Zuendholz wegnehmen.});
linerest := linerest - 2;
comment
```

```
    NE
    NE;
    := last wins := true;
    man next := false;
    WRITE TEXT(
    {<Jeg begynder:}, {<I begin:},
    {<Je commence:}, {<Ich fange an:});
AA: LINE;
    (AME MAN, MACHINE; man next);
    LINE;
    if man wins then WRITE TEXT(
    {<De har vundet. Tillykke.},
    {<You have won. Congratulations.},
    {<Vous avez gagné. Mes félicitations.},
    {<Sie haben gewonnen. Ich gratuliere Ihnen.})
    else
    WRITE TEXT(
    {<De har tabt.}, {<You have lost.},
    {<Vous avez perdu.}, {<Sie haben verloren.});
    if QUESTION(
    {<Skal vi prøve igen}, {<Shall we try again},
    {<Voulez vous jouer encore}, {<Sollen wir nochmals spielen}) then
    begin
        groups := ASK NUMBER(
        {<Hvor mange bunker}, {<How many groups},
        {<Combien de groupes}, {<Wieviel Haufen});
        if groups < 2 ∨ groups > 10 then
        begin
            groups := if groups < 2 then 2 else 10;
            LINE;
            write({<nd}, groups);
            WRITE TEXT(
            {< er bedre.}, {< is better.},
            {< va mieux.}, {< ist besser.})
        end if out of range;
comment;
```

```
      if QUESTION(
      {<Ønsker De selv at bestemme antallet af tændstikker},
      {<Do you want to specify the number of matches},
      {<Voulez vous specifier le nombre des allumettes},
      {<Wollen Sie die Anzahl der Zuendhoelzer angeben})then
      begin
         LINE;
         WRITE TEXT(
         {<Skriv dem her}, {<Write them here},
         {<Ecrivez les ici}, {<Schreiben Sie ihnen hier});
         LINE;
         for g := 1 step 1 until groups do GROUP[g] := typein
      end
      else
      begin
         LINE;
         WRITE TEXT(
         {<Her er bunkerne:}, {<Here are the groups:},
         {<Voici les groupes:}, {<Hier sind die Haufen:});
         LINE;
         for g := 1 step 1 until groups do
         GROUP[g] := 1 + RANDOM INTEGER(2↑groups - 2);
         PRINT GROUPS
      end;
      LINE;
      man next := QUESTION(
      {<Vil De begynde}, {<Do you want to begin},
      {<Voulez vous commencer}, {<Wollen Sie anfangen});
      last wins := QUESTION(
      {<Skal den, som tager sidste tændstik, vinde},
      {<Must he who takes the last match win},
      {<Celui qui prend la derniere allumette, est-il vainqueur},
      {<Hat derjenige, der das letzte nimmt, gewonnen});
      go to AA
   end if more games
end program;
```

8.2. Udskrift fra et spil.

```
Select language: d: danish, e: english, f: french, g: german.: d
Dansk


Skriv Deres initialer her og slut med mellemrum: jk
PROGRAM DEMON-3
Tændstikspillet NIM. Vi vælger først nogle tilfældige bunker af tændstikker:

Bunke nr.:          1   2   3
Antal tændstikker:  5   2   5

Vi skal nu skiftevis fjerne tændstikker fra bunkerne. Den, som fjerner den
eller de sidste tændstikker, har vundet. Kun een bunke maa røres i hvert
træk, og man skal fjerne mindst een tændstik fra den bunke.

Jeg begynder:

Jeg fjerner nu hele bunke nr.  2. Bunkerne indeholder nu:

  5  0  5
Skriv nummeret paa den bunke, fra hvilken De vil fjerne tændstikker: 3
Og antallet af tændstikker, De vil fjerne: 3
De kan ikke vinde dette spil.
Jeg fjerner nu  3 fra bunke nr.  1. Bunkerne indeholder nu:

  2  0  2
Vælg Deres bunke: 1
Og antallet: 2
Giver De fortabt: nej
Jeg fjerner nu hele bunke nr.  3. Bunkerne indeholder nu:

  0  0  0
De har tabt.
Skal vi prøve igen: ja
Hvor mange bunker: 3
Ønsker De selv at bestemme antallet af tændstikker: nej
Her er bunkerne:
  2  2  2
```

```
Vil De begynde: ja
Skal den, som tager sidste tændstik, vinde: ja

Vælg Deres bunke: 3
Og antallet: 2
Hvis De spiller rigtigt, kan De vinde dette spil.
Jeg fjerner nu  1 fra bunke nr.  1. Bunkerne indeholder nu:

  1  2  0
Vælg Deres bunke: 2
Og antallet: 1
Jeg fjerner nu hele bunke nr.  1. Bunkerne indeholder nu:

  0  1  0
Vælg Deres bunke: 2
De har vundet. Tillykke.
Skal vi prøve igen: ja
Hvor mange bunker: 3
Ønsker De selv at bestemme antallet af tændstikker: nej
Her er bunkerne:
  3  3  6

Vil De begynde: ja
Skal den, som tager sidste tændstik, vinde: ja

Vælg Deres bunke: 3
Og antallet: 6
Hvis De spiller rigtigt, kan De vinde dette spil.
Jeg fjerner nu  1 fra bunke nr.  1. Bunkerne indeholder nu:

  2  3  0
Vælg Deres bunke: 2
Og antallet: 2
Det var forkert. Nu kan De ikke vinde.
Giver De fortabt: nej
Jeg fjerner nu  1 fra bunke nr.  1. Bunkerne indeholder nu:

  1  1  0
Vælg Deres bunke: 5
Undskyld, men tallet er for stort.
```

```
Vælg Deres bunke: 3
Undskyld, men denne bunke er tom.
Vælg Deres bunke: 2
Jeg fjerner nu hele bunke nr.  1. Bunkerne indeholder nu:

   0   0   0
De har tabt.
Skal vi prøve igen: ja
Hvor mange bunker: 3
Ønsker De selv at bestemme antallet af tændstikker: ja
Skriv dem her
 4   1   3

Vil De begynde: ja
Skal den, som tager sidste tændstik, vinde: nej

Vælg Deres bunke: 1
Og antallet: 2
Hvis De spiller rigtigt, kan De vinde dette spil.
Jeg fjerner nu  1 fra bunke nr.  3. Bunkerne indeholder nu:

   2   1   2
Vælg Deres bunke: 2
Jeg fjerner nu  1 fra bunke nr.  1. Bunkerne indeholder nu:

   1   0   2
Vælg Deres bunke: 3
Og antallet: 2
Jeg fjerner nu hele bunke nr.  1. Bunkerne indeholder nu:

   0   0   0
De har vundet. Tillykke.
Skal vi prøve igen: ja
Hvor mange bunker: 6
Ønsker De selv at bestemme antallet af tændstikker: nej
Her er bunkerne:
   4  29  29  49  27  54
```

Vil De begynde: nej
Skal den, som tager sidste tændstik, vinde: ja

Jeg fjerner nu 8 fra bunke nr. 6. Bunkerne indeholder nu:

4 29 29 49 27 46
Vælg Deres bunke: 1
Og antallet: 17
Saa mange er der ikke i bunken. De fjerner altsaa hele bunken.
De kan ikke vinde dette spil.
Jeg fjerner nu 4 fra bunke nr. 6. Bunkerne indeholder nu:

0 29 29 49 27 42
Vælg Deres bunke: 2
Og antallet: 29
Giver De fortabt: ja
De har tabt.
Skal vi prøve igen: nej
end