Gaming en coding

Dr Didier Deses*

Inhoudsopgave

*Leerkracht wiskunde K. A. Koekelberg, medewerker aan het departement wiskunde van de VUB

Voorwoord

Waarschijnlijk is het eerste contact dat een kind heeft met moderne technologie één of andere game. Op school echter worden games vaak afgeschilderd als een boosdoener: van cyberpesten tot te weinig slaap om te studeren. Een uitgangspunt dat vaak onbesproken blijft is de techniek achter een spel. We zullen in dit cahier een aantal spelletjes zelf programmeren op de **TI-84 Plus Color** . Omdat een rekentoestel beperkter is dan een computer zullen soms wiskundige algoritmen gepast gebruikt moeten worden en soms zullen handige programmeertrucjes nodig zijn.

De spelletjes in dit cahier zijn in stijgende moeilijkheidsgraad opgenomen. **Level 1** bevat games die een goede kennismaking vormen met het programmeren voor de **TI-84 Plus Color** . Zowel de basiscommando's zoals `For...end, If...Then...Else...End, Lbl...Goto, Disp, Input, Text, GetKey, ...` als de nodige basistechnieken voor animaties komen aan bod.

Level 2 echter bevat geavanceerde spelletjes. Hierin worden bekende spelletjes in een versie voor de **TI-84 Plus Color** geprogrammeerd. De nadruk ligt hierbij op de gebruikte trucjes en technieken. Leerlingen zien hier dat de gekende spelletjes, al lijken ze eenvoudig, vaak programmeertechnieken benutten die uitblinken in originaliteit.

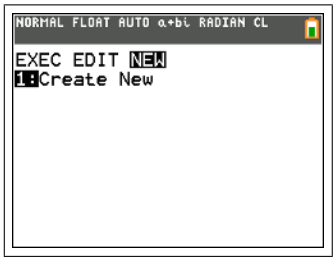
Level 3 zijn spelletjes waar het programma niet echt moeilijker is dan Level 2 maar wel iets langer en opgesplitst in enkele deelprogramma's. Het resultaat is echter wel telkens een uitdagend spel! De voorbeelden geven een duidelijk beeld over hoe het spel werkt, hoe het geprogrammeerd is en zelfs hoe een AI (Artificiële Intelligentie) te werk gaat om je te verslaan!

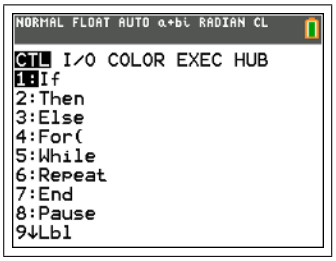
Inleiding: Hoe maak ik een programma?

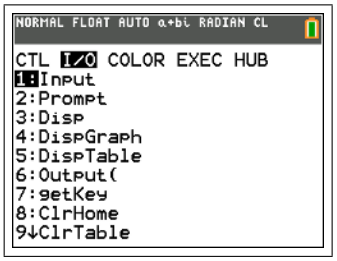
Wie reeds een programma heeft geschreven op de **TI-84 Plus Color** mag gerust deze paragraaf overslaan en beginnen programmeren.

De programmeertaal die in de **TI-84 Plus Color** zit, is TIBasic. Dit is een dialect van BASIC, een programmeertaal waarmee Bill Gates (Microsoft) zijn faam heeft verworven. BASIC (Beginners All-purpose Symbolic Instruction Code) werd ontworpen om ook de leek in staat te stellen om kleine programma's te schrijven. Het TIBasic-dialect is trouw aan deze filosofie: er is geen enkele programmeer-ervaring nodig om programma's te schrijven voor de **TI-84 Plus Color** . Enkel een beetje doorzettingsvermogen en zelfvertrouwen is nodig.

Om een programma te schrijven ga je naar `prgm`. Je kunt hier kiezen om een programma uit te voeren (`[exec]`), te veranderen (`[edit]`) of om een nieuw programma te schrijven (`[new]`). Indien je de laatste keuze maakt wordt er naar een naam gevraagd. Nadien kom je op de editor uit, waar je jouw programma kan invoeren. De commando's die met het programmeren te maken hebben, zitten nu onder `prgm`. We geven hier een kort overzicht van de nuttigste programmeerfuncties.







[ctl]	
[If]	Eerste vorm: `:If` *voorwaarde* `:` *commando*
[Then]	Tweede vorm: uitgebreide `if` structuur:

[Else]	`:If` *voorwaarde*
	`:Then`
	`:` *commando's*
	`:Else`
	`:` *commando's*
	`:End`

[For]	om lussen te maken
	`:For(` *var, beginwaarde, eindwaarde [, stapgrootte]* `)`
	`:` *commando's*
	`:End`

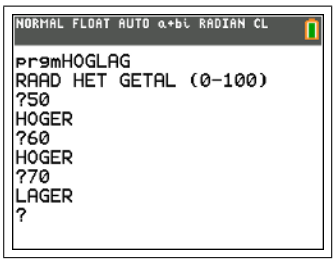
[End]	om bovenstaande blokken te eindigen
[Lbl]	`:Lbl` *getal* om een plaats in het programma aan te duiden
[Goto]	`:Goto` *getal* om maat een bepaald `Lbl` te gaan

[i/o]	
[Disp]	om een waarde/string op het scherm te printen
[Prompt]	om de waarde van een variabele te vragen aan de gebruiker
[Input]	om een tekstt op het scherm te tonen en een waarde te vragen aan de gebruiker `:Input` *"text", variabele*
[GetKey]	om een toetsaanslag in te lezen

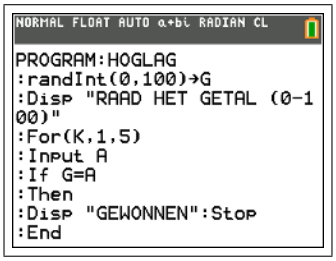
Nu we de nodige commando's kennen, kunnen we ons eerste spelletje maken.

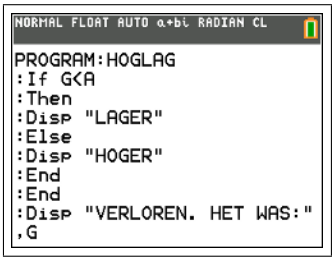
1 Level 1

1.1 Hoger-lager



Het spelletje hoger-lager is gemakkelijk programmeerbaar. De computer kiest een willekeurig getal tussen 1 en 100. Jij mag het trachten te raden!





Bespreking:

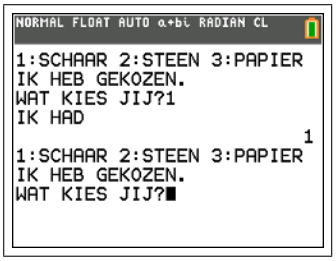
- Eerst wordt via `math`[prob][randint] een willekeurig geheel getal gekozen en opgeslagen met `sto` in de variabele G.
- Met `Disp` druk je een tekst af op het scherm zodat de speler weet wat hij of zij moet doen. Je vindt dit commando in het menu `prgm`[i/o].
- Daarna begint een `For`-lus (`prgm`[ctl]) om de speler vijf beurten te geven.
- Met `input` wordt een getal opgevraagd en in de variabele A onthouden.
- Via een `If...Then...End`-structuur (`prgm`[ctl]) wordt in het geval dat $G = A$ gemeld dat de speler gewonnen heeft en wordt het programma gestopt.
- Indien G en A verschillend zijn, wordt een `If...Then...Else...End`-structuur gestart om na te gaan of $G < A$ en afhangend hiervan "Hoger" of "Lager" af te drukken.

- De `For`-lus wordt tenslotte afgesloten zodat de volgende beurt kan beginnen.
- Wanneer de laatste beurt gedaan is, wordt het gekozen getal afgedrukt.

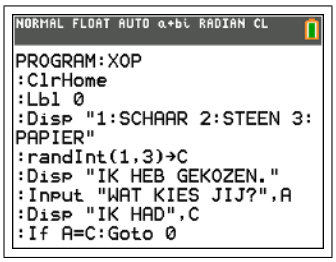
Uitbreiding:

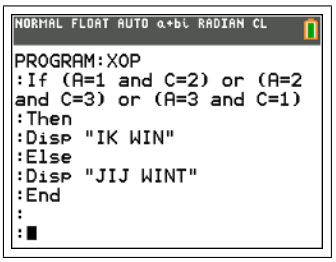
- Bij dit spel bestaat een optimale strategie om te winnen. Beschrijf deze. Hoeveel rondes heb je nodig om zeker altijd te winnen?
- Kan je het omgekeerde spel programmeren? Jij hebt een getal in gedachten en de computer raadt het.

1.2 Schaar-Steen-Papier



Onderstaand programma laat je het spel Schaar-Steen-Papier op de **TI-84 Plus Color** spelen.





Bespreking:

- Het scherm wordt leeggemaakt met `prgm` `[i/o]` `[ClrHome]` en een keuzemenu wordt afgedrukt.
- De **TI-84 Plus** kiest willekeurig één van de drie opties.
- De speler wordt dan om een keuze gevraagd en de keuze van de **TI-84 Plus** wordt kenbaar gemaakt.
- Een `If...:Goto ...`-structuur zorgt ervoor dat bij gelijk spel het programma opnieuw start.
- Indien er geen gelijk spel is wordt met een `If...Then...Else...End`-structuur nagegaan wie wint.

Uitbreiding:

- Kan jij de output mooier maken?
- Een goede oefening op meetkundige reeksen is het aantonen dat men een kans $\frac{1}{3}$ heeft om één spel te winnen maar dat, indien er bij gelijkspel wordt verdergespeeld, de kans stijgt tot $\frac{1}{2}$.
- In de tv-reeks *"The big bang theory"* komt de variant *"Rock, Paper, Scissors, Lizard, Spock "* voor:

Scissors cuts Paper

Paper covers Rock

Rock crushes Lizard

Lizard poisons Spock

Spock smashes Scissors

Scissors decapitates Lizard

Lizard eats Paper

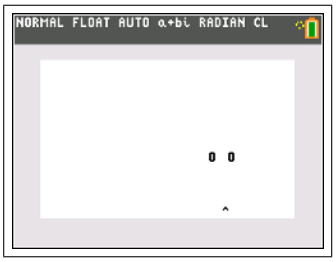
Paper disproves Spock

Spock vaporizes Rock

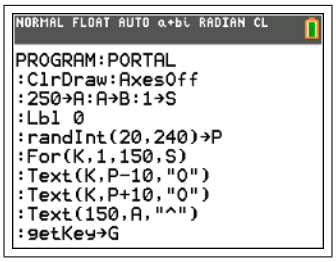
(and as it always has) Rock crushes Scissors

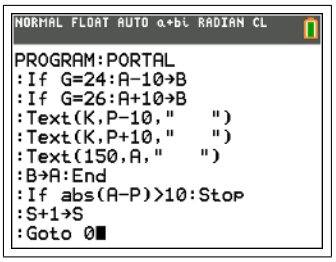
Kan je dit spel ook programmeren?

1.3 Portal



In dit spelletje moet je een ruimteschip doorheen zoveel mogelijk portalen leiden. We zullen zien hoe beweging in een spel gebracht kan worden en hoe de interactie met een speler efficiënt kan gedaan worden.





Bespreking:

- Eerst wordt het grafisch scherm leeggemaakt en de assen verwijderd. We gebruiken in dit spel het grafisch scherm dat een resolutie heeft van 265 pixels horizontaal en 165 verticaal.

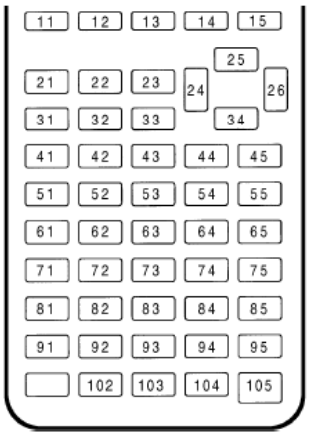
- De positie van het ruimteschip in de horizontale richting (voor en na beweging) wordt bijgehouden in A en B. Aanvankelijk begint het schip rechts op het scherm ($A = 250$). De snelheid wordt bijgehouden in S.

- Vervolgens wordt willekeurig met `randInt` de positie van het portaal gekozen (tussen 20 en 240).

- Er volgt nu een `for`-lus waarbij K de hoogte van het portaal is. Aanvankelijk op de eerste rij pixels en zo naar de 150ste rij, helemaal onderaan het scherm. De verticale schermcoördinaat wordt uitgedrukt in rijen pixels en is dus tegengesteld aan de traditionele y-as. De stap S bepaalt de snelheid waarmee de `for`-lus wordt doorlopen en dus ook de snelheid van het spel.

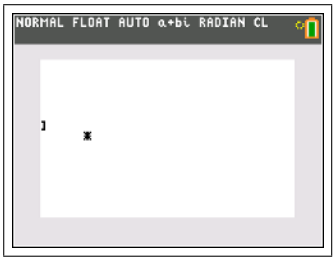
- Bij elke stap in de lus wordt het portaal getekend (20 pixels breed) alsook het ruimteschip op rij 150 onderaan het scherm.

- `getKey` wordt gebruikt om na te gaan of de speler een toets ingedrukt houdt.
- Indien de toets de waarde 24 had, ($\leftarrow$) wordt de nieuwe positie $A - 10$. Indien de toets de waarde 26 had, ($\rightarrow$) wordt de nieuwe positie $A + 10$. De waarden van de toetscodes vind je hieronder.

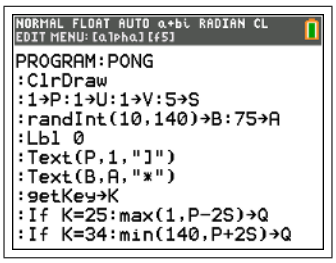


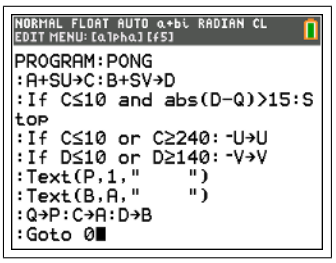
- Hierna worden het portaal en het ruimteschip weggeveegd. Dit gebeurt door spatie over de symbolen `O` en `^` heen af te drukken. Omdat echter een spatie smaller is dan `O` of `^` drukken we er vier af.
- De nieuwe positie wordt aangepast en de lus wordt gesloten.
- Wanneer de lus afgelopen is, bevindt het portaal zich onderaan het scherm. Als de afstand tussen het ruimteschip en het portaal te groot is, stopt het spel, anders wordt de snelheid verhoogd en een nieuwe level gestart.

1.4 Pong



Het spelletje Pong was één van de eerste computerspelletjes ooit. Een balletje botst heen en weer, je mag de bal niet mis slaan!





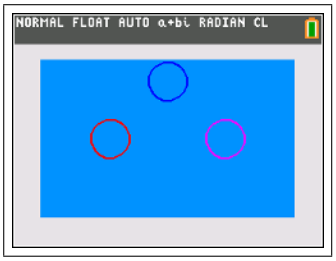
Bespreking:

- De positie van de speler wordt bijgehouden in P, die van de bal in (A, B). De richting van de bal wordt gegeven door de vector (U, V), de snelheid door S.
- Het scherm van de **TI-84 Plus Color** bestaat uit 265×165 pixels, om met ronde waarden te werken, gebruiken we een veld waarbij de lengte gaat van 10 tot 240 en de hoogte van 10 tot 140.
- K houdt bij of de speler een pijltjestoets indrukt.
- De nieuwe positie van de speler zit in Q, die van de bal in (C, D).
- Indien de bal bij de linkerrand komt en de speler te ver verwijderd is, verliest deze het spel.
- Als de nieuwe positie van de bal voorbij de rand van het veld komt, verandert de richting van de bal van teken. Dit zorgt voor de weerkaatsing van de bal.
- Voor het herstarten van de lus worden de bal en de speler weggeveegd en worden de posities aangepast.

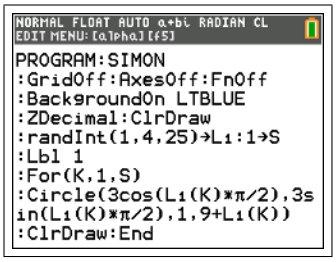
Uitbreiding:

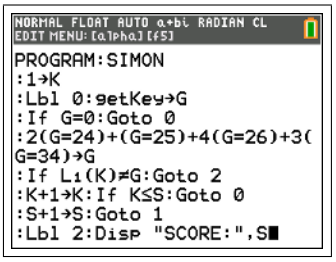
- Kan je de kleuren aanpassen?
- Maak een passend score-systeem.
- Zorg ervoor dat bij elke slag de snelheid toeneemt.

1.5 Simon says . . .



In dit klassiek spel krijgt de speler een steeds langere rij cirkels te zien. Daarna moet hij de rij kunnen reproduceren.



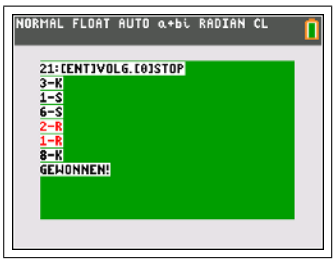


Bespreking:

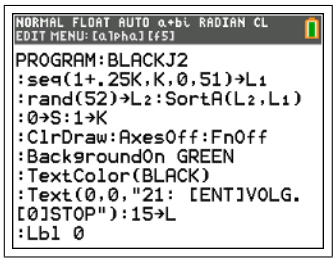
- In de eerste drie regels wordt het grafisch scherm aangepast voor het spelletje.
- Vervolgens wordt een willekeurige rij van 25 windrichtingen (1,2,3,4=N,O,Z,W) in L_1 gestoken.
- Een lus zorgt ervoor dat de deelrij van lengte S wordt getoond.
- Hierna wordt gewacht tot de speler aan de hand van de pijltjestoetsen de reeks ingeeft. Wanneer een toets wordt ingedrukt, wordt eerst de toetscode omgezet in één van de windrichtingen (1,2,3,4).
- Indien de gegeven richting niet overeenstemt met het Kde teken in de rij wordt naar `lbl 2` gesprongen om het spel te eindigen met de behaalde score.
- Indien de richting wel overeenstemt en de lengte S nog niet bereikt is, wordt K eentje groter en springt het programma terug naar `lbl 0` om de volgende input van de speler te krijgen.
- Indien de speler de volledige deelrij van lengte S correct heeft, wordt S eentje groter en begint het programma opnieuw de rij te tonen.

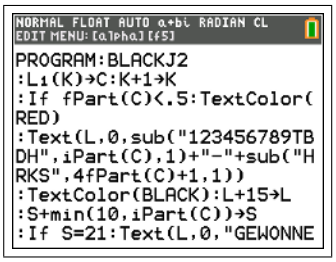
2 Level 2

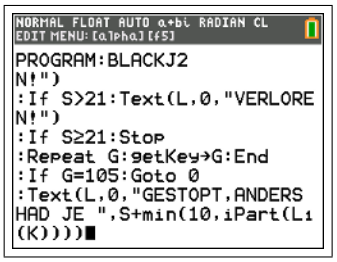
2.1 Blackjack



Met dit spelletje tonen we hoe een kartspel voorgesteld kan worden op de **TI-84 Plus Color** . Kaarten worden na elkaar getrokken, de bedoeling is om zo dicht mogelijk bij 21 te komen maar er nooit boven.







Bespreking:

- Een pak kaarten simuleren is lastig omdat elke kaart een waarde $(1, \ldots, 13)$ en een soort (Harten, Ruiten, Klaveren, Schoppen) heeft. Om deze beide elementen in één enkel getal te plaatsen gebruiken we volgende truc. De soorten komen overeen met de getallen na de komma $(0.0, 0.25, 0.5, 0.75)$, de waarde met het cijfer voor de komma. Zo komt het laatste getal (13.75) overeen met $H\spadesuit$. Het pak kaarten wordt aangemaakt in L_1.

- Nu worden de kaarten gemengd. Dit gebeurt door een tweede lijst van willekeurige getallen te plaatsen in L_2. Beide lijsten worden dan geordend afhankelijk van de waarden in L_2, de kaarten in L_1 staan daarna in een willekeurige volgorde. De variabele K is nu de eerste kaart en deze wordt de getrokken kaart C.

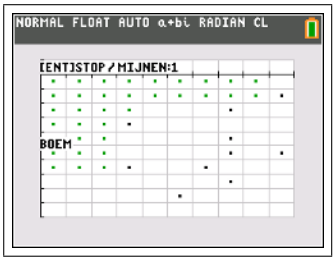
- Het afdrukken van een kaart gebeurt als volgt. L houdt bij op welke lijn er moet worden afgedrukt. Standaard staat de kleur op zwart. Indien `fPart` kleiner is dan 0.5 gaat het om een rode kaart en wordt de kleur veranderd. Met `iPart` wordt de waarde van de kaart bepaald. Deze gebruikt men dan om het juiste symbool uit `123456789TBDH` te

selecteren (we schrijven hier `T` voor 10 zodat elke kaart juist één symbool nodig heeft). Op analoge wijze wordt `fPart` gebruikt om de soort voor te stellen `HRKS` (Harten, Ruiten, Klaveren, Schoppen).

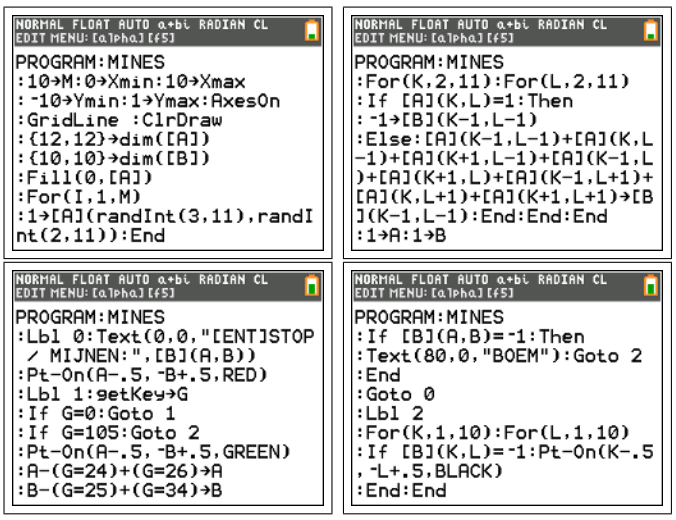
- De som van de getrokken kaarten wordt bijgehouden in S. Door gebruik te maken van `min(iPart(L1(K)),10)` tellen alle beeldjes voor 10, de azen tellen in deze versie voor 1. Er wordt getoetst of de speler gewonnen of verloren heeft, in dat geval wordt het programma gestopt.

- Anders wordt er gewacht op de reactie van de speler. Indien deze op de toets [enter] heeft gedrukt (toetscode 105) gaat het spel door (K wordt $K + 1$) anders wordt er gestopt en toont men de som die de speler zou hebben gehad.

2.2 Mijnenveger



In dit alombekende spel is het de bedoeling de mijnen in een mijnenveld te vinden.



Bespreking:

- Het veld is 10×10. `GridLine` en de `AxesOn` zorgen voor de afbakening. Om het mijnenveld te maken, beginnen we met een tijdelijke 12×12 matrix A (het speelveld plus een extra rand) die overal nul is. Willekeurig worden nu M mijnen geplaatst d.m.v. `[A](randint(3,11),randint(2,11))`. De ongelijke argumenten van `randint` zorgen ervoor dat de speler zeker altijd een eerste stap naar beneden kan zetten.

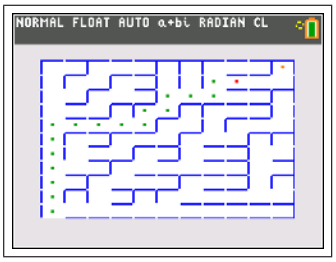
- Eenmaal dit gedaan is, gaan we een 10×10 matrix B opvullen. Indien $A_{K,L}$ een mijn bevat zal de overeenstemmende plaats in $B_{K-1,L-1}$ waarde -1 krijgen. Anders worden de mijnen in de acht buurcellen geteld en krijgt $B_{K-1,L-1}$ deze waarde. A zal hierna niet meer gebruikt worden.

- De beginpositie is $(A, B) = (1, 1)$. Nadat wordt afgedrukt hoeveel mijnen er in de buurt zijn wordt een lus gestart die wacht op input van de speler. Als de ingedrukte toets `enter` was (toetscode=105) springt het programma naar `lbl 2` om te eindigen. Anders wordt de positie van de speler als groen (veilig) aangeduid in het midden van de cel (coördinaten $(A - 0.5, -B + 0.5)$ (het veld ligt onder de x-as dus negatieve y-waarde)).

- De nieuwe positie wordt bepaald door het handige feit dat een vergelijking $G = 24$ de waarde 1 of 0 krijgt (waar/vals). In twee regels kan men dus controleren of een pijltjestoets (toetscodes $\leftarrow$: $24, \rightarrow$: $26, \uparrow$: $25, \downarrow$: 34) werd ingedrukt en tegelijk ook de nieuwe positie berekenen. Indien de speler op een mijn belandt, springt het programma naar `lbl 2` om te eindigen, anders herbegint de input-lus vanaf `lbl 0`.

- Het einde van het programma plaatst alle mijnen in het zwart. Het programma voorziet geen controle op de plaatsing van de mijnen ('eilandjes') noch een test om te zien hoe ver een speler staat.

2.3 Doolhof



In dit programma gaan we een doolhof maken. Er bestaan veel algoritmen om dit te doen, de meeste echter zijn ingewikkeld en steunen op een meer grondige kennis van wiskunde (bijvoorbeeld grafentheorie). Onderstaande methode is echter zeer eenvoudig en produceert enkelvoudig samenhangende doolhoven, dit zijn doolhoven waavoor elke twee punten verbonden zijn door een unieke weg.

```
NORMAL FLOAT AUTO a+bi RADIAN CL
EDIT MENU: [α]Pha[α] [f5]
PROGRAM:MAZEBIN
:100→N
:Lbl 0:ClrDraw
:0→Xmin:N→Xmax
:0→Ymin:N→Ymax
:For(K,0,N,10)
:Horizontal K:Vertical K
:End
:For(K,0,N-10,10)
:For(L,0,N-10,10)
```

```
NORMAL FLOAT AUTO a+bi RADIAN CL
EDIT MENU: [α]Pha[α] [f5]
PROGRAM:MAZEBIN
:randInt(0,1)→R
:If L=0:1→R
:If K=0:0→R
:If R=1:Line(K,L,K,L+10,0)

:If R=0:Line(K,L,K+10,L,0)

:End:End
:Pt-On(N-5,N-5,ORANGE)
```

```
NORMAL FLOAT AUTO a+bi RADIAN CL
EDIT MENU: [α]Pha[α] [f5]
PROGRAM:MAZEBIN
:5→A:5→B
:Lbl 1:Pt-On(A,B,RED)
:Lbl 2:getKey→G
:If G=0:Goto 2
:Pt-On(A,B,GREEN)
:A-10(G=24)+10(G=26)→A
:B-10(G=34)+10(G=25)→B
:If A≠N-5 or B≠N-5:Goto 1
:N+10→N:Goto 0
```

Bespreking:

- We beginnen met een raster van 10×10 cellen. Elke cel is 10 punten breed, de assen van het grafisch scherm gaan dus van 0 tot $N = 100$. Voor elke cel (K, L) kiezen we willekeurig $R = 0$ of 1. In het eerste geval verwijderen we de linkermuur van de cel, anders de onderste (het commando `Line(...,0)` duidt aan dat we een lijn willen verwijderen). We letten er wel op dat in de eerste kolom en de onderste rij nooit een buitenmuur wordt weggehaald. Eens de doolhof getekend is, markeren we het eindpunt in het oranje.

- De speler (rood) begint linksonder (positie $(A, B) = (5, 5)$) en moet zijn weg zoeken tot de rechter bovenhoek. Een lus wordt gestart tot een toets wordt ingedrukt. De positie van de speler wordt groen.

- De nieuwe positie wordt bepaald door het handige feit dat een vergelijking $G = 24$ de waarde 1 of 0 krijgt (waar/vals). In twee regels kan de **TI-84 Plus Color** controleren of een pijltjestoets (toetscodes $\leftarrow$: 24, $\rightarrow$: 26, $\uparrow$: 25, $\downarrow$: 34) werd ingedrukt en tegelijk ook de nieuwe positie berekenen (een stap van lengte 10 in de gegeven richting).

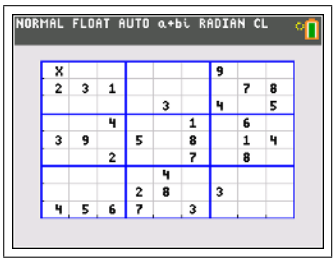
- Indien de nieuwe positie niet de eindpositie is, wordt de lus gesloten. Als de speler wel het eindpunt heeft bereikt, wordt het spel opnieuw gestart met een grotere N (één cel meer in beide richtingen).

Uitbreiding:

- De doolhoven vormen een binaire boomstructuur en zijn eenvoudig op te lossen. Hoe doe je dit? Verklaar nu de naam binaire boom.
- De speler kan in deze versie gerust doorheen de muren, er is geen wall-checking algoritme. Dit kan gedaan worden maar is een stukje moeilijker en vertraagt het programma enorm.
- Bij het tekenen van het raster kan je een extra parameter plaatsen: `Horizontal K,1:Vertical K,1`. Wat doet dit? Wat heeft dit als gevolg?

3 Level 3

3.1 Sudoku's



Inleiding:

Een sudoku bestaat uit 9×9 vakjes die gegroepeerd zijn als 9 blokken van 3×3 vakjes. De cijfers 1 t.e.m. 9 moeten ingevuld worden zodat in elke rij en in elke kolom en in elk blok elk cijfer juist één keer voorkomt. In een aantal vakjes zijn de cijfers al ingevuld in het begin.

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9

=blokrij

Algoritmen om sudoku's te genereren zijn ingewikkeld en te tijdrovend voor een rekentoestel. Als we vertrekken van een bestaande sudoku kunnen we deze wel omvormen tot een nieuwe:

1. Twee blokrijen mag men wisselen.
2. In een blokrij mag men twee rijen wisselen.
3. Men mag de sudoku transponeren.
4. Men mag de getallen herbenoemen (vb: alle 1'en worden 9 en omgekeerd).

Het programma bestaat uit een subprogramma `SUDOMK` dat een sudoku aanmaakt en een hoofdprogramma `SUDOKU` dat de gebruikersinterface bevat.

Programma `SUDOMK`**:**

```
NORMAL FLOAT AUTO a+bi RADIAN CL
EDIT MENU: [alpha] [f5]
PROGRAM:SUDOMK
:[[-5,-3,4,6,-7,8,9,1,2][-
6,7,2,-1,-9,-5,3,4,8][1,-9
,-8,3,4,2,5,-6,7][-8,5,9,7
,-6,1,4,2,-3][-4,2,6,-8,5,
-3,7,9,-1][-7,1,3,9,-2,4,8
,5,-6][9,-6,1,5,3,7,-2,-8,
4][2,8,7,-4,-1,-9,6,3,-5][
3,4,5,2,-8,6,1,-7,-9]]→[H]
```

```
NORMAL FLOAT AUTO a+bi RADIAN CL
PROGRAM LINE 8
PROGRAM:SUDOMK
:For(I,1,2)
:randInt(0,2)→K
:randInt(0,2)→L
:For(J,1,3)
:rowSwap([H],3K+J,3L+J)→[H
]:End
:For(K,0,2)
:rowSwap([H],3K+randInt(1,
3),3K+randInt(1,3))→[H]
```

```
NORMAL FLOAT AUTO a+bi RADIAN CL
EDIT MENU: [alpha] [f5]
PROGRAM:SUDOMK
:End
:[H]ᵀ→[H]
:End
:If rand<.5:[H]ᵀ→[H]
:randIntNoRep(1,9,9)→L₁
:For(K,1,9):For(L,1,9)
:[H](K,L)→T:abs(T)/T*L₁(ab
s(T))→[H](K,L)
:End:End
```

Bespreking:

Eerst wordt een opgeloste sudoku als 9×9 matrix A ingegeven. De zichtbare getallen geven we in als negatieve waarden. Via `2nd` `[rcl]` `[A]` plaatst men de sudoku in het programma in de matrixvariabele `[H]`. Daarna worden de transformaties (1-2) toegepast. De lus over I zorgt ervoor dat dit tweemaal gebeurt, voor H en $H^\top$. Nadien wordt de sudoku eventueel nogmaals getransponeerd en worden de getallen herbenoemd.

Programma `SUDOKU`**:**

```
NORMAL FLOAT AUTO a+bi RADIAN CL
EDIT MENU: [alpha] [f5]
PROGRAM:SUDOKU
:Input "NEW (1/0)?",G
:If G:prgmSUDOMK
:0→Xmin:9→Xmax:1→Xscl
:0→Ymin:9→Ymax:1→Yscl
:GridLine :ClrDraw
:For(K,0,9,3)
:Vertical K:Horizontal K
:End
:For(K,1,9):For(L,1,9)
```

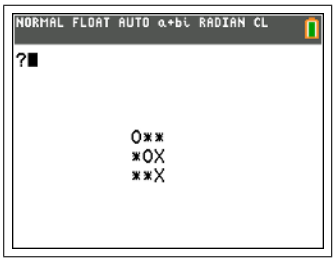
```
NORMAL FLOAT AUTO a+bi RADIAN CL
EDIT MENU: [alpha] [f5]
PROGRAM:SUDOKU
:If [H](K,L)<0:Text(3+18(K
-1),16+28(L-1),abs([H](K,L
)))
:End:End:1→A:1→B:3→U:16→V
:Lbl 0:Text(U,V,"X")
:Repeat G:getKey→G:End
:Text(U,V,"    ")
:If [H](A,B)<0:Text(U,V,ab
s([H](A,B)))
```

```
NORMAL FLOAT AUTO a+bi RADIAN CL
EDIT MENU: [alpha] [f5]
PROGRAM:SUDOKU
:A-(G=25)+(G=34)→A
:B-(G=24)+(G=26)→B
:3+18(A-1)→U:16+28(B-1)→V
:If G=105:Then:Prompt N
:If [H](A,B)≠N:Stop
:Text(U,V,[H](A,B))
:-[H](A,B)→[H](A,B)
:If -abs([H])=[H]:Stop
:End:Goto 0
```

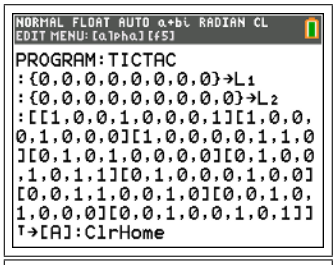
Bespreking:

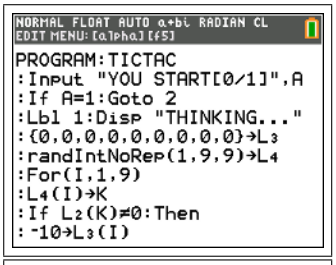
Het hoofdprogramma vraagt eerst of men een nieuwe sudoku wil (`SUDOMK` aanroepen) of verder wil gaan met de vorige. Via een passend grafisch venster wordt het raster getekend. Met de formules $3 + 18(K - 1)$ en $16 + 28(L - 1)$ komen de zichtbare waarden van $H_{K,L}$ op de juiste plaats. Nadien begint een lus waarmee een cursor `X` kan bewegen. Wanneer `enter` werd gedrukt (code 105), wordt een waarde gevraagd. Als deze niet correct is, wordt er gestopt anders wordt ze zichtbaar gemaakt. Met $-abs(H) = H$ wordt gekeken of de sudoku volledig opgelost is.

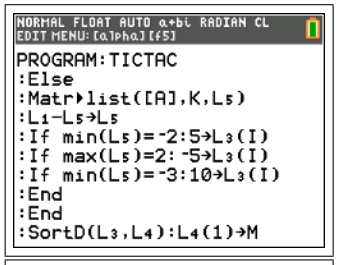
3.2 Tic Tac Toe

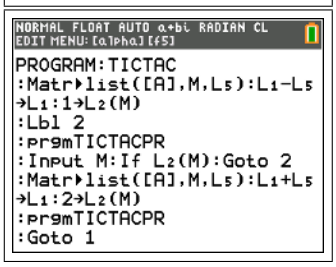


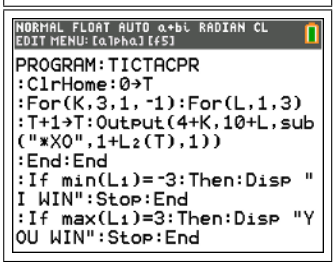
Spelers plaatsen om beurt hun teken (O of X) op een 3×3 veld. De eerste met drie op een rij wint. Ons programma bevat een kleine AI (artificiële intelligentie), het hoofdprogramma `TICTAC` bevat het spel, het subprogramma `TICTACPR` (2 laatste schermen) zorgt voor het afdrukken op het scherm.

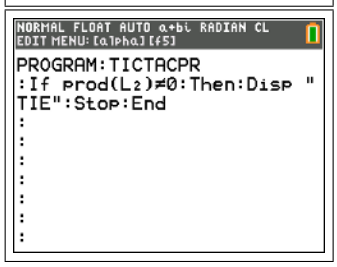












Bespreking:

- Het veld wordt voorgesteld door de 9 cijfers op het klavier en wordt bijgehouden in de lijst L_2. Het spel wordt gewonnen indien een rij $(H1, H2, H3)$, een kolom $(V1, V2, V3)$ of een diagonaal $(D1, D2)$ eenzelfde teken bevat. We houden dit bij in de lijst

  $$L_1 = [H1, H2, H3, V1, V2, V3, D1, D2]$$

  Wanneer een 1 wordt ingevuld zal dit een bijdrage leveren aan $H1, V1$ en $D2$. De zet 1 komt overeen met een bijdrage $[1, 0, 0, 1, 0, 0, 0, 1]$. De bijdragen van de zetten vormen een 9×8 matrix, die we zullen transponeren omdat `matr>list` kolommen omzet naar lijsten en geen rijen.

	$V1$	$V2$	$V3$		
$H3$	7	8	9		$1 = [1,0,0,1,0,0,0,1]$
$H2$	4	5	6		$2 = [1,0,0,0,1,0,0,0]$
$H1$	1	2	3		$3 = [1,0,0,0,0,1,1,0]$
$D2$				$D1$	$4 = [0,1,0,0,1,0,0,0]$
					$\ldots$

- Een zet van de speler zal een positieve bijdrage geven aan L_1, een zet van de AI zal negatief meetellen. In L_2 wordt de AI voorgesteld door 1 (X), de speler door 2 (O). De speler zal dus winnen indien L_1 een 3 bevat, de AI indien -3 in L_1 zit. Indien L_1 de waarde 2 bevat, heeft de speler nog maar één zet nodig om te winnen. De AI zal dit gebruiken om haar zetten te bepalen.

- Voor dit spel bestaat er een optimale strategie. Deze kan je programmeren (ingewikkeld!), maar dan zal de AI altijd winnen of gelijk spel spelen. We kiezen hier dus voor andere strategie (vanaf `lbl 1`). Via L_4 en K worden alle zetten (1-9) in willekeurige volgorde afgegaan, in L_3 wordt de score van elke zet bijgehouden. Indien de zet reeds gebruikt is, wordt de score -10, anders wordt de zet uitgevoerd en bekijkt de AI de nieuwe situatie L_5:

  1. als de AI 2 op een rij ($min(L_5) = -2$) heeft, is de score 5
  2. tenzij de speler 2 op een rij heeft ($max(L_5) = 2$), dan is de score -5
  3. tenzij de de AI wint, met score 10.

- Met `SortD` bepaalt de AI de zet M met de hoogste score en voert deze uit (L_1 en L_2 worden aangepast). Hierna (`lbl 2`) wordt de nieuwe toestand afgedrukt en is het de beurt aan de speler. Nadien weer aan de AI (`lbl 1`).

- Het programma `tictacpr` drukt het veld af en gaat na of er een winnaar of gelijk spel is.

3.3 Pirates

In vele spelletjes dwaal je rond in een zeer grote en soms zelf eindeloze wereld. Hoe kan een computer of in dit geval een **TI-84 Plus Color** zo'n wereld simuleren? We geven hier het voorbeeld van een wereld met 25 eilanden waar een schat verborgen is.

De truc is het gebruik van de pseudorandomgenerator `rand`. `rand` is een speciale variabele dat een getal tussen 0 en 1 bevat. Wanneer het wordt aangeroepen, gebruikt de **TI-84 Plus** dit om een willekeurig getal te genereren. `rand` krijgt dan die waarde. Herhaaldelijk aanroepen geeft een rij van *pseudorandomgetallen*.

Deze rij getallen heet *random* omdat ze uniform verdeeld zijn, elk getal heeft evenveel kans om voor te komen. De rij is echter maar *pseudorandom* want je kan een *seed* (beginwaarde) geven aan `rand`: *seed*→`rand`. De rij die daarna gemaakt wordt, is altijd dezelfde (maar wel nog steeds uniform verdeeld). Hoe dit werkt kan je zelf opzoeken. Uit `rand` zijn een aantal andere commando's afgeleid. Wij gaan deze gebruiken om een wereld vol eilanden te genereren.

```
NORMAL FLOAT AUTO a+bi RADIAN CL
EDIT MENU: [α]Pha] [f5]
PROGRAM:PIRATES
:ClrHome
:Input "MAP NR:",M
:Disp "YOU ARE IN POSSESIO
N OF A"
:Disp "MAP OF 25 ISLANDS.
ON ONE"
:Disp "A TREASURE IS HIDDE
N. YOUR"
:Disp "SHIP HAS FOOD FOR 9
```

```
NORMAL FLOAT AUTO a+bi RADIAN CL
EDIT MENU: [α]Pha] [f5]
PROGRAM:PIRATES
0 DAYS.":Disp ""
:Disp "LETS FIND TREASURE
ISLAND!"
:Repeat G:getKey→G:End
:M→rand:25→N:1→A:1→B:90→F
:0→Xmin:0→Ymin:1→ΔX:1→ΔY
:BackgroundOn LTBLUE
:randIntNoRep(1,250,N)→L1
:randInt(1,150,N)→L2
```

```
NORMAL FLOAT AUTO a+bi RADIAN CL
EDIT MENU: [α]Pha] [f5]
PROGRAM:PIRATES
:randInt(1,150,N)→L2
:L1(1)→T:SortA(L1)
:Lbl 1
:Plot1(Scatter,L1,L2,▫,GRE
EN):Pt-On(A,B,YELLOW)
:Circle(A,B,F/2,ORANGE)
:Circle(A,B,F,RED):Trace
:int(√((X-A)²+(Y-B)²))→D
:X→A:Y→B:F-D→F
```

```
NORMAL FLOAT AUTO a+bi RADIAN CL
EDIT MENU: [α]Pha] [f5]
PROGRAM:PIRATES
:If F<0:Then
:Text(60,0,"YOU STARVED AT
 SEA"):Stop:End
:prgmISLANDS
:If A=T:Then
:Text(105,0,"YOU FOUND TRE
ASURE ISLAND"):Stop:End
:Repeat G:getKey→G:End
:Goto 1
```

```
NORMAL FLOAT AUTO a+bi RADIAN CL
EDIT MENU: [α]Pha] [f5]
PROGRAM:ISLANDS
:PlotsOff :ClrDraw
:TextColor(BROWN)
:Text(0,0,"DIST:",D," DAYS
")
:Text(15,0,"FOOD:",F," DAY
S LEFT")
:Text(30,0,"EXPLORING ISLA
ND...")
:A→rand:randInt(80,200)→S
```

```
NORMAL FLOAT AUTO a+bi RADIAN CL
EDIT MENU: [α]Pha] [f5]
PROGRAM:ISLANDS
:randInt(0,2)→R
:160→U:60→V:For(K,1,S)
:U+3randInt(-1,1)→U
:V+3randInt(-1,1)→V
:Pt-On(U,V,2,GREEN):End
:Text(45,0,"SIZE:",S,"KM²"
)
:Text(60,0,"TYPE:",sub("TR
EESGRASSROCKS",1+5R,5))
```

```
NORMAL FLOAT AUTO a+bi RADIAN CL
EDIT MENU: [α]Pha] [f5]
PROGRAM:ISLANDS
:If rand<.1:Then
:Text(75,0,"YOU FOUND FOOD
"):80→F:End
:If rand<.05:Then
:If T<A:Text(90,0,"TREASUR
E ISLAND <- WEST")
:If T>A:Text(90,0,"TREASUR
E ISLAND -> EAST")
:End
```

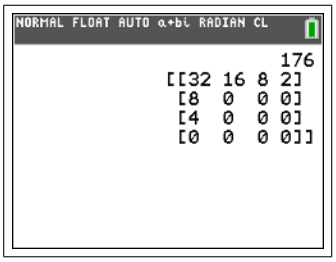
Bespreking:

- Eerst wordt een kaartnummer gevraagd, dit zal de *seed* vastleggen en dus ook de kaart.
- Een intro wordt afgedrukt om het spel uit te leggen.
- De code `Repeat G:GetKey->G:End` zorgt ervoor dat er wordt gewacht tot de speler een toets indrukt om verder te gaan.
- De kaart bestaat uit de coördinaten van 25 eilanden (L_1 en L_2). Met `radnIntNoRep` wordt ervoor gezorgd dat de$x$- coördinaten altijd verschillend zijn. Eentje ervan is schatteneiland T. Daarna worden de eilanden gesorteerd naar de x-coördinaten.
- Met een scatterplot wordt de kaart getekend. De nodige commando's vind je onder 2nd `[stat plot]`. Twee cirkels duiden aan hoe ver de boot kan reizen en tot waar een terugreis mogelijk is.
- We gebruiken `trace` om de bestemming van de speler op te vragen.
- Nadat de afstand en de overgebleven voedselvooraad F werd uitgerekend wordt nagegaan of de reis niet te lang was. Indien niet, dan gaat het spel verder.
- Het subprogramma `islands` genereert een eiland. De unieke x-coördinaat wordt als `seed` gebruikt. Vanaf nu kan `rand` dus gebruikt worden om de eigenschappen van dit eiland te bepalen.
- Aan de hand van een *randomwalk* wordt het eiland getekend. De grootte S van het eiland bepaalt de lengte van de wandeling. Daarna gebruiken we weer randomgetallen om een korte beschrijving te gegeven.
- In 10% van de gevallen wordt de voedselvooraad opnieuw gevuld en soms wordt een hint over schatteneiland gevonden.
- Na terugkeer naar het hoofdprogramma wordt nagegaan of dit het schatteneilend is, anders keert de speler terug naar de kaart.

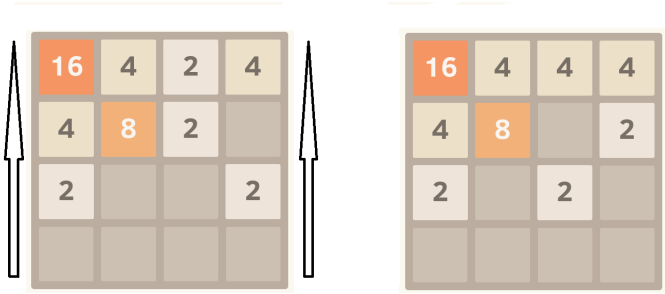
Uitbreiding:

Je kan zelf extra's toevoegen aan een eiland en tijdens de boottocht kan je misschien een zeegevecht inlassen.

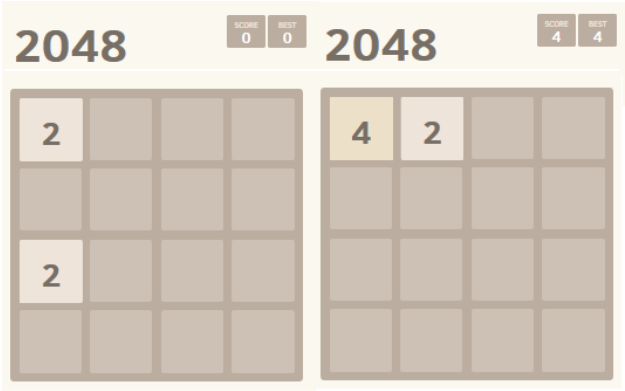
3.4 2048



Dit spel is tegenwoordig zeer populair op pc of smartphones. Het bevat een 4×4 bord met getallen. De speler kan kiezen of hij de getallen naar boven, onder, links of rechts schuift. Wanneer twee dezelfde getallen op elkaar schuiven, worden ze opgeteld. Daarna komt er een 2 of een 4 bij. Het doel van het spel is om 2048 of een hoger getal te maken.

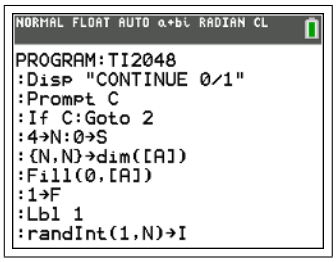


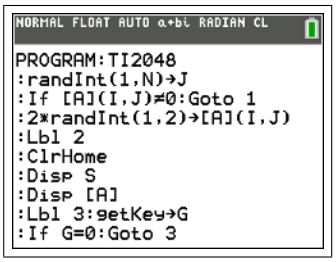
Wanneer je een nieuw getal maakt, wordt deze bij je huidige score opgeteld. Als we dus in onderstaand geval de twee 2's bij elkaar schuiven, krijgen we een 4, vier punten dus. Daarna verschijnt een nieuw getal.

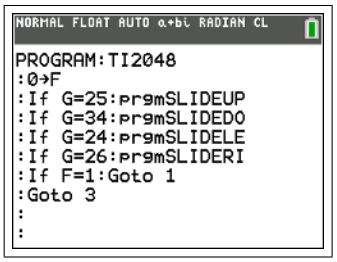


Het spel werd geschreven door de 19-jarige *Gabriele Cirulli* als programmeeroefening om te zien of hij in één weekend tijd een computerspel kon programmeren. Het werd een echte hype.

De implementatie op de **TI-84 Plus Color** maakt gebruik van één hoofdprogramma `TI2048` en van 4 subprogramma's die het verschuiven in een bepaalde richting uitvoeren.







Bespreking:

- Eerst wordt de keuze gegeven om verder te gaan met een vorig spel.
- Indien er een nieuw spel wordt begonnen, maakt de **TI-84 Plus** een 4×4 nulmatix A aan. Op een willekeurige positie A_{ij} waar een 0 staat wordt met `2*randInt(1,2)` een 2 of een 4 geplaatst.
- Vervolgens wordt het scherm leeggemaakt en wordt de score S en de matrix A afgedrukt.
- Hierna wordt gewacht op input van de speler.
- De vlag F houdt bij of de zet iets verandert aan de matrix A, deze wordt op 0 gezet. Afhangend van input (via de pijltjestoetsen) wordt het gepaste subprogramma aangeroepen.
- De subprogramma's voeren de zet uit en passen de score aan. Indien de situatie verandert (geldige zet), wordt de vlag F gelijk aan 1.
- Indien $F = 1$ keert het programma terug naar `Lbl 1` en wordt opnieuw een 2 of een 4 op een lege plaats toegevoegd. Anders gaat het programma naar `Lbl 3` waar wordt gewacht op een geldige zet.

De subprogramma's die voor het uitvoeren van de zetten zorgen zijn:

```
NORMAL FLOAT AUTO a+bi RADIAN CL

PROGRAM:SLIDEUP
:For(J,1,N):For(I,1,N)
:For(K,I+1,N)
:If [A](K,J)≠0
:Then
:If [A](I,J)=0
:Then
:[A](K,J)→[A](I,J):0→[A](K
,J):1→F
:Else
```

```
NORMAL FLOAT AUTO a+bi RADIAN CL

PROGRAM:SLIDEUP
:If [A](I,J)=[A](K,J)
:Then
:[A](I,J)+[A](K,J)→[A](I,J
):S+[A](I,J)→S:0→[A](K,J):
1→F:End
:N→K
:End
:End:End
:End:End
```

```
NORMAL FLOAT AUTO a+bi RADIAN CL

PROGRAM:SLIDEDO
:For(J,1,N):For(I,N,1,-1)
:For(K,I-1,1,-1)
:If [A](K,J)≠0
:Then
:If [A](I,J)=0
:Then
:[A](K,J)→[A](I,J):0→[A](K
,J):1→F
:Else
```

```
NORMAL FLOAT AUTO a+bi RADIAN CL

PROGRAM:SLIDEDO
:If [A](I,J)=[A](K,J)
:Then
:[A](I,J)+[A](K,J)→[A](I,J
):S+[A](I,J)→S:0→[A](K,J):
1→F:End
:1→K
:End
:End:End
:End:End
```

```
NORMAL FLOAT AUTO a+bi RADIAN CL

PROGRAM:SLIDELE
:For(I,1,N):For(J,1,N)
:For(K,J+1,N)
:If [A](I,K)≠0
:Then
:If [A](I,J)=0
:Then
:[A](I,K)→[A](I,J):0→[A](I
,K):1→F
:Else
```

```
NORMAL FLOAT AUTO a+bi RADIAN CL

PROGRAM:SLIDELE
:If [A](I,J)=[A](I,K)
:Then
:[A](I,J)+[A](I,K)→[A](I,J
):S+[A](I,J)→S:0→[A](I,K):
1→F:End
:N→K
:End
:End:End
:End:End
```

```
NORMAL FLOAT AUTO a+bi RADIAN CL

PROGRAM:SLIDERI
:For(I,1,N):For(J,N,1,-1)
:For(K,J-1,1,-1)
:If [A](I,K)≠0
:Then
:If [A](I,J)=0
:Then
:[A](I,K)→[A](I,J):0→[A](I
,K):1→F
:Else
```

```
NORMAL FLOAT AUTO a+bi RADIAN CL

PROGRAM:SLIDERI
:If [A](I,J)=[A](I,K)
:Then
:[A](I,J)+[A](I,K)→[A](I,J
):S+[A](I,J)→S:0→[A](I,K):
1→F:End
:1→K
:End
:End:End
:End:End
```

Je hoeft deze programma's zeker niet allemaal in te tikken. Geef het eerste programma `SLIDEUP` in. Maak dan een nieuw programma `SLIDEDO`. Eenmaal in de editor gebruik je (2nd)[rcl](prgm)[exec][SLIDEUP] om met `rcl prgmSLIDEUP` het programma te kopiëren. Maak tenslotte de nodige aanpassingen en herhaal voor de andere programma's.