Verslag derde programmeeropdracht

Alex Keizer & Léon van Velzen

10 november 2017

1 Uitleg programma

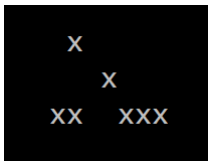
Life is origineel bedacht door John Conway. De "wereld"bestaat uit een 2-dimensionaal vlak van cellen, die levend of dood zijn. Een volgende generatie wordt d.m.v. twee simpele regels bepaald:

- Een levende cel overleeft alleen als het precies 2 of 3 levende buren heeft
- Een dode cel met 3 levende buren komt weer tot leven

Met buren worden de acht direct of diagonaal aanliggende cellen bedoeld. In de theorie is de wereld oneindig groot, wij hebben ons beperkt tot 1000×1000. Deze simpele regels zorgen voor vrij complex gedrag, zie bijvoorbeeld de glidergun[1] en de acorn besproken in Paragraaf 2
De gebruiker ziet slechts een 80×25 deel van de wereld tegelijk, de zogeheten "view". Hij kan zijn view bewegen m.b.v. de `wasd` toetsen. Verder is er onder de view een menu te vinden waarmee het programma bestuurd wordt.

2 Acorn

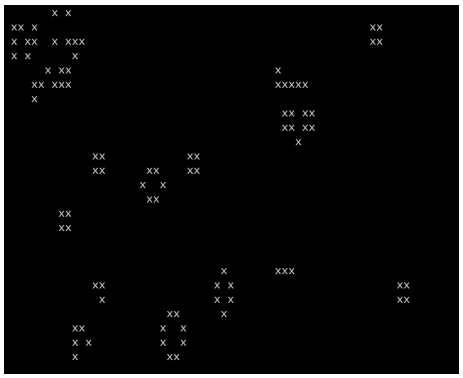
Waar de glidergun als het ware een geweer is dat continu gliders produceert, is de 'acorn'[2] het beste als een explosief te beschrijven. Hij begint simpel, de start configuratie bestaat uit slechts 7 cellen (zie Figuur 1).



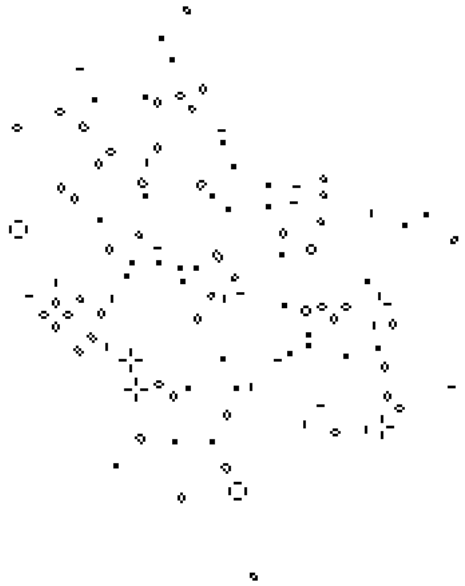
Figuur 1: De start configuratie van de acorn

In het begin breidt de vorm langzaam uit. Rond generatie 130 zien we dat de actie vooral naar links gebeurd, rechts zien we eigenlijk alleen 2 stabiele 2×2 blokken en een 'blinker' (zie Figuur 2). Deze vormen zien we rond generatie 300 nogsteeds, er zijn zelfs nog meer stabiele vormen verschenen (zie Figuur 3).
Op dit punt zijn de interessante elementen helaas niet meer in de grootte van de view te zien, uiteindelijk zal de acorn in generatie 5206 zijn volgroeid tot een stabiel patroon (zie Figuur 4.

Figuur 2: Generatie 130



Figuur 3: Generatie 300



Figuur 4: Generatie 5206, de stabiele staat [2]

3 Windows

Bij het continue draaien van Life hebben wij de `nanosleep()` functie gebruikt om leesbaarheid van een generatie te verbeteren. Helaas werkt deze functie niet op Windows en is er geen goed OS-onafhankelijk alternatief (`usleep()` wordt als `deprecated` gemarkeerd). De functie is echter niet noodzakelijk, op Windows zal het scherm slechts wat sneller verversen.

4 Tijd

Laatste wijziging gemaakt op 10/11/2017. Tijd in uren besteed per week:

Week	Alex Keizer	Léon van Velzen
42	1	?
43	0	?
44	1	?
45	5	?
Totaal	7	?

Referenties

[1] Gosper glider gun. Geraadpleegd op 9/11/2017, van http://www.conwaylife.com/w/index.php?title=Gosper_glider_gun

[2] Acorn. Geraadpleegd op 9/11/2017, van http://www.conwaylife.com/wiki/Acorn

5 Code

```
1  //  Tweede opdracht voor Programmeermethoden
2  //  Auteurs:
3  //        Alex Keizer (s2046253), eerstejaars Informatica (2017)
4  //        L on van Velzen (s2037033), eerstejaars Informatica (2017)
5  //
6  //  Gecompileerd onder:
7  //      Windows 10 met MinGW64 GCC versie 6.3.0
8  //      Debian GNU/Linux 9 GCC versie 6.3.0
9  //
10 //  Aan gewerkt in de periode 27/10/2017 tot 10/11/2017
11 //
12 //  Onze implementatie van John Conways Game Of Life
13
14
15 #include <iostream>
16 #include <fstream>
17 #include <cstring>
18 #include <ctime>
19 #include <unistd.h>
20
21
22 using namespace std;
23
24 // Random number generator
25 class RNG {
26     private:
27     static long random_number;
28
29     static const long MULT = 29;
30     static const long INC = 1;
31
32     public:
33
34     static const long MOD = 65536; // 2^16
35
36     static void set_seed(long seed) {
37         RNG::random_number = seed % MOD;
38     }
39
40     static long get_random_number() {
41         return (RNG::random_number = (MULT * RNG::random_number + INC) % MOD);
42     }
43 };
44
45 // Define storage for variable
46 long RNG::random_number;
47
48 class Life {
49     const static int WORLD_SIZE = 1000;
50
```

```
51      const int VIEW_WIDTH = 80;
52      const int VIEW_HEIGHT = 25;
53
54      // Position of the view
55      int view_x = 0;
56      int view_y = 0;
57
58      // Absolute position of the cursor
59      int cursor_x = 0;
60      int cursor_y = 0;
61
62      int generation = 0;
63
64      // Parameters
65      // Size of changes in view or cursor position
66      int view_step_size = 1;
67      int cursor_step_size = 1;
68      // Characters for respectively dead or alive cells
69      char dead_cell = ' ';
70      char live_cell = 'x';
71      // The percentage (out of 100) of random cells picked and made alive
72      // when the view is filled randomly.
73      int random_percentage = 50;
74
75      bool board[WORLD_SIZE][WORLD_SIZE] = { 0 };
76      // Used to fill the actual board when computing
77      // the next generation
78      bool temp_board[WORLD_SIZE][WORLD_SIZE] = { 0 };
79
80      static bool positionWithinWorld(int x, int y) {
81          return x >= 0 && x < WORLD_SIZE
82              && y >= 0 && y < WORLD_SIZE;
83      }
84
85      // Whether the position is on the edge of the world,
86      // the edge of the world is one character wide
87      static bool positionOnEdge(int x, int y) {
88          return ((x == -1 || x == WORLD_SIZE) && y >= -1 && y <= WORLD_SIZE) //
                Vertical edges
89              || ((y == -1 || y == WORLD_SIZE) && x >= -1 && x <= WORLD_SIZE); //
                    Horizontal edges
90
91      }
92
93      bool positionWithinView(int x, int y) {
94          return x >= view_x && x <= view_x + VIEW_WIDTH
95              && y >= view_y && y <= view_y + VIEW_HEIGHT;
96      }
97
98      bool isAlive(int x, int y) {
99          // Edges are dead by default
100         return Life::positionWithinWorld(x, y) && board[x][y];
101     }
102
103     int countLiveNeighbours(int x, int y) {
104         return isAlive(x - 1, y - 1)
105             + isAlive(x, y - 1)
106             + isAlive(x + 1, y - 1)
107             + isAlive(x - 1, y)
108             + isAlive(x + 1, y)
109             + isAlive(x - 1, y + 1)
```

```
110             + isAlive(x, y + 1)
111             + isAlive(x + 1, y + 1);
112     }
113
114     public:
115
116     Life() {
117         time_t rawtime;
118         time(&rawtime);
119
120         struct tm date_now = *localtime(&rawtime);
121
122         RNG::set_seed(date_now.tm_sec);
123     }
124
125     void nextGeneration() {
126         // Produce the next board according
127         // to the rules of Game of Life
128         for(int x = 0; x < WORLD_SIZE; x++)
129         for(int y = 0; y < WORLD_SIZE; y++)
130         {
131             int count = countLiveNeighbours(x, y);
132
133             if(isAlive(x, y)) {
134                 if(count == 2 || count == 3) temp_board[x][y] = true;
135                 else temp_board[x][y] = false;
136             }
137             else if(count == 3)
138             {
139                 temp_board[x][y] = true;
140             }
141             else
142             {
143                 temp_board[x][y] = false;
144             }
145         }
146
147         generation++;
148         memcpy(board, temp_board, sizeof(bool) * WORLD_SIZE * WORLD_SIZE);
149     }
150
151     void killAll() {
152         for(int x = 0; x < WORLD_SIZE; x++)
153         for(int y = 0; y < WORLD_SIZE; y++)
154         {
155             board[x][y] = false;
156         }
157     }
158
159     void killView() {
160         for(int x = view_x; x < view_x + VIEW_WIDTH; x++)
161         for(int y = view_y; y < view_y + VIEW_HEIGHT; y++)
162         {
163             if(positionWithinWorld(x,y))
164                 board[x][y] = false;
165         }
166     }
167
168     // Moves the view, won't go further than -1 or WORLD_SIZE
169     void moveView(int x, int y) {
170         int new_x = view_x + x*view_step_size;
```

```
171         int new_y = view_y + y*view_step_size;
172 
173         if(new_x < 0) view_x = -1;
174         else if(new_x > WORLD_SIZE - VIEW_WIDTH)
175             view_x = WORLD_SIZE - VIEW_WIDTH + 1;
176         else view_x = new_x;
177 
178         if(new_y < 0) view_y = -1;
179         else if(new_y > WORLD_SIZE - VIEW_HEIGHT)
180             view_y = WORLD_SIZE - VIEW_HEIGHT + 1;
181         else view_y = new_y;
182     }
183 
184     void resetCursor() {
185         cursor_x = view_x;
186         cursor_y = view_y;
187     }
188 
189     void moveCursor(int x, int y) {
190         int new_x = cursor_x + x*cursor_step_size;
191         int new_y = cursor_y + y*cursor_step_size;
192 
193         if(positionWithinView(new_x, new_y))
194         {
195             cursor_x = new_x;
196             cursor_y = new_y;
197         }
198     }
199 
200     void toggleCursor() {
201         if(Life::positionWithinWorld(cursor_x, cursor_y)) {
202             board[cursor_x][cursor_y] = !board[cursor_x][cursor_y];
203         }
204     }
205 
206     // Fills the view with random cells (determined by random_percentage)
207     void makeRandomAlive() {
208         int threshold = random_percentage * RNG::MOD / 100;
209 
210         // Then fill cells at random
211         for(int x=view_x; x<view_x+VIEW_WIDTH; x++)
212         for(int y=view_y; y<view_y+VIEW_HEIGHT; y++)
213         {
214             if(positionWithinWorld(x,y)){
215                 board[x][y] = RNG::get_random_number() < threshold;
216             }
217         }
218     }
219 
220     void printView(bool print_cursor) {
221 
222         cout << "Coordinates of view: (" << view_x << ", " << view_y << ")  ";
223         cout << "Gen: " << generation << "    ";
224         cout << "Stepsize: " << view_step_size << "/" << cursor_step_size << ", ";
225         cout << "Alive: '" << live_cell << "', ";
226         cout << "Dead: '" << dead_cell << "'" << endl;
227 
228         if(print_cursor)
229         {
230             for(int x = view_x; x < cursor_x && x < view_x + VIEW_WIDTH; x++)
231             {
```

```
232                 cout << " ";
233             }
234             cout << " |" << endl;
235         }
236 
237 
238         for(int y = view_y; y < view_y + VIEW_HEIGHT; y++)
239         {
240             if(print_cursor)
241             {
242                 if(y == cursor_y) cout << "-";
243                 else cout << " ";
244             }
245 
246             for(int x = view_x; x < view_x + VIEW_WIDTH; x++)
247             {
248                 if(Life::positionWithinWorld(x, y) && board[x][y])
249                     cout << live_cell;
250                 else if(Life::positionOnEdge(x, y)) cout << "#";
251                 else cout << dead_cell;
252             }
253 
254             cout << endl;
255         }
256     }
257 
258     void fillViewFromFile(string filename)
259     {
260         int symbol;
261 
262         int x = view_x;
263         int y = view_y;
264 
265         ifstream file(filename);
266 
267         killAll();
268 
269         while((symbol = file.get()) != EOF)
270         {
271             if(symbol == '\n')
272             {
273                 x = view_x;
274                 y += 1;
275             }
276             else
277             {
278                 if(Life::positionWithinWorld(x, y))
279                 {
280                     if(symbol == 'x') board[x][y] = true;
281                     else board[x][y] = false;
282                 }
283 
284                 x += 1;
285             }
286         }
287     }
288 
289     int get_maximum_cursor_step_size() {
290         return (
291             Life::VIEW_HEIGHT < Life::VIEW_WIDTH ?
292                                 Life::VIEW_HEIGHT
```

```
293                                     : Life::VIEW_WIDTH
294         ) - 1;
295     }
296
297     // Parameter getters and setters
298
299     int get_view_step_size() {return view_step_size;}
300     int get_cursor_step_size() {return cursor_step_size;}
301     char get_dead_cell() {return dead_cell;}
302     char get_live_cell(){return live_cell;}
303     int get_random_percentage(){return random_percentage;}
304
305     void set_view_step_size(int new_size)
306     {
307         if(0 <= new_size && new_size <= 500)
308             view_step_size = new_size;
309     }
310
311     void set_cursor_step_size(int new_size)
312     {
313         if(0 <= new_size && new_size <= get_maximum_cursor_step_size())
314             cursor_step_size = new_size;
315     }
316
317     void set_dead_cell(char new_char)
318     {
319         if(new_char != '\t' && new_char != '\n')
320             dead_cell = new_char;
321     }
322
323     void set_live_cell(char new_char)
324     {
325         if(new_char != '\t' && new_char != '\n')
326             live_cell = new_char;
327     }
328
329     void set_random_percentage(int new_percentage)
330     {
331         if(0 <= new_percentage && new_percentage <= 100)
332             random_percentage = new_percentage;
333     }
334
335 };
336
337 class Menu {
338
339     private:
340
341     // Gives the first non-newline character on stdin
342     static char read_character()
343     {
344         char kar;
345
346         while((kar = cin.get()) == '\n'){};
347
348         return kar;
349     }
350
351     // Reads a number from stdin, stops when number would exceed maximum
352     static int read_number(int maximum)
353     {
```

```
354         int num = 0;
355 
356         char kar = read_character();
357         do {
358             if ('0' <= kar && kar <= '9')
359             {
360                 int new_num = num * 10;
361                 new_num += kar - '0';
362                 if(new_num <= maximum)
363                 {
364                     num = new_num;
365                 } else {
366                     // Prevent extra numbers from being seen as menu-input
367                     while(cin.get() != '\n'){}
368                     return num;
369                 }
370             }
371             kar = cin.get();
372         } while(kar != '\n');
373         return num;
374     }
375 
376     static void print_cursor_menu()
377     {
378         cout << "Use w,a,s,d to move cursor around" << endl;
379         cout << "Press space to toggle cell" << endl;
380         cout << "Press b to return to menu" << endl;
381     }
382 
383     // handles input, returns whether screen should be redrawn
384     static bool input_cursor_menu(Life *game, char input)
385     {
386         switch(input)\
387         {
388             case 'w':
389                 game->moveCursor(0, -1);
390                break;
391             case 'a':
392                 game->moveCursor(-1, 0);
393                 break;
394             case 's':
395                 game->moveCursor(0, 1);
396                 break;
397             case 'd':
398                 game->moveCursor(1, 0);
399                 break;
400 
401             case ' ':
402                 game->toggleCursor();
403                 break;
404             case 'b':
405                 Menu::current_menu = Menu::MAIN;
406                 break;
407         }
408         return false;
409     }
410 
411     static void print_param_menu(Life *game) {
412         cout << endl;
413         cout << "Change parameters or press b to go back" << endl;
414         cout << "1. View Step Size (" << game->get_view_step_size() << ") ";
```

```
415         cout << "2.Cursor Step Size (" << game->get_cursor_step_size() << ") ";
416         cout << "3.Random Filling Percentage (" << game->get_random_percentage()
417             << ") ";
418         cout << "4.Alive Char ('" << game->get_live_cell() << "') ";
419         cout << "5.Dead Char ('" << game->get_dead_cell() << "') " << endl;
420     }
421
422     // handles input, returns whether screen should be redrawn
423     static bool input_param_menu(Life *game, char input){
424         switch(input){
425             case '1':
426                 game->printView(false);
427                 cout << endl;
428                 cout << "Numbers will be read as long as they don't exceed the "
429                     << "given maximum, further digits are ignored" << endl;
430                 cout << "Give a new step size (0-500) for view changes:"
431                     << endl;
432                 game->set_view_step_size( read_number(500) );
433                 break;
434
435             case '2':
436                 game->printView(false);
437                 cout << endl;
438                 cout << "Numbers will be read as long as they don't exceed the "
439                     << "given maximum, further digits are ignored" << endl;
440                 cout << "Give a new step size (0-" << game->
                        get_maximum_cursor_step_size()
441                     <<") for cursor changes:" << endl;
442                 game->set_cursor_step_size(
443                         read_number( game->get_maximum_cursor_step_size() )
444                     );
445                 break;
446
447             case '3':
448                 game->printView(false);
449                 cout << endl;
450                 cout << "Numbers will be read as long as they don't exceed the "
451                     << "given maximum, further digits are ignored" << endl;
452                 cout << "Give a percentage (0-100) for random filling:" << endl;
453                 game->set_random_percentage( read_number(100) );
454                 break;
455
456             case '4':
457                 game->printView(false);
458                 cout << endl << endl;
459                 cout << "Give a new character for live cells: "
460                     << "(Tab is not allowed)" << endl;
461                 game->set_live_cell( read_character() );
462                 break;
463
464             case '5':
465                 game->printView(false);
466                 cout << endl << endl;
467                 cout << "Give a new character for dead cells: "
468                     << "(Tab is not allowed)" << endl;
469                 game->set_dead_cell( read_character() );
470                 break;
471
472             case 'b':
473                 current_menu = MAIN;
474                 return false;
```

```
475
476             default:
477                 return false;
478         }
479         return true;
480     };
481
482     static void print_main_menu()
483     {
484         cout << "1. Exit ";
485         cout << "2. Clean world ";
486         cout << "3. Clean view ";
487         cout << "4. Change parameters ";
488         cout << "5. Random ";
489         cout << "6. Toggle using cursor ";
490         cout << "7. Load glidergun.txt";
491         cout << " 8. Compute one generation ";
492         cout << "9. Run Game of Life " << endl;
493         cout << "Use w,a,s,d to move view around" << endl;
494     }
495
496     // handles input, returns whether screen should be redrawn
497     static bool input_main_menu(Life *game, char input)
498     {
499         switch(input)
500         {
501             // Movements
502             case 'w':
503                 game->moveView(0, -1);
504                 break;
505             case 'a':
506                 game->moveView(-1, 0);
507                 break;
508             case 's':
509                 game->moveView(0, 1);
510                 break;
511             case 'd':
512                 game->moveView(1, 0);
513                 break;
514
515             // Menu
516             case '1': exit(0);
517
518             case '2':
519                 game->killAll();
520                 break;
521             case '3':
522                 game->killView();
523                 break;
524             case '4':
525                 current_menu = PARAM;
526                 break;
527             case '5':
528                 game->makeRandomAlive();
529                 break;
530             case '6':
531                 game->resetCursor();
532                 current_menu = CURSOR;
533                 break;
534             case '7':
535                 game->fillViewFromFile("gliderGun.txt");
```

```
536                 break;
537             case '8':
538                 game->nextGeneration();
539                 break;
540             case '9':
541                 for(;;)
542                 {
543                     game->nextGeneration();
544                     game->printView(false);
545                     cout << endl << endl << endl;
546 
547                     // Sleep for 50ms (POSIX only)
548                     struct timespec sleep = {0};
549                     sleep.tv_sec = 0;
550                     sleep.tv_nsec = 50000000L;
551                     nanosleep(&sleep, (struct timespec *)NULL);
552                 }
553                 break;
554         }
555         return false;
556     }
557 
558     public:
559 
560     static int current_menu;
561 
562     static const int MAIN = 0;
563     static const int CURSOR = 1;
564     static const int PARAM = 2;
565 
566     static void print_menu(Life *game)
567     {
568         switch(current_menu){
569             case CURSOR:
570                 print_cursor_menu();
571                 break;
572 
573             case PARAM:
574                 print_param_menu(game);
575                 break;
576 
577             case MAIN:
578             default:
579                 print_main_menu();
580                 break;
581         }
582     }
583 
584     // Handles input for the active menu
585     // Returns whether the screen should be redrawn
586     static bool handle_input(Life *game, char input){
587         if(input == '\n')
588         {
589             return true;
590         } else {
591             switch(current_menu){
592                 case CURSOR:
593                     return input_cursor_menu(game, input);
594                     break;
595 
596                 case PARAM:
```

```
597                     return input_param_menu(game, input);
598                     break;
599
600                 case MAIN:
601                 default:
602                     return input_main_menu(game, input);
603                     break;
604             }
605         }
606     }
607 };
608
609 int Menu::current_menu = Menu::MAIN;
610
611 int main()
612 {
613     Life *game = new Life();
614     Menu::current_menu = Menu::MAIN;
615
616     game->printView(false);
617     Menu::print_menu(game);
618
619     for(;;) {
620         char input = cin.get();
621         if(Menu::handle_input( game, input) ) {
622             game->printView( Menu::current_menu == Menu::CURSOR );
623             Menu::print_menu(game);
624         };
625     }
626
627     return 0;
628 }
```