# Verslag derde programmeeropdracht

Alex Keizer & Léon van Velzen

10 november 2017

## 1 Uitleg programma

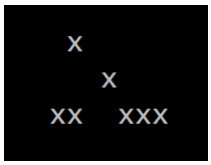
Life is origineel bedacht door John Conway. De "wereld"bestaat uit een 2-dimensionaal vlak van cellen, die levend of dood zijn. Een volgende generatie wordt d.m.v. twee simpele regels bepaald:

- 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 $1000 \times 1000$. 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 $80 \times 25$ 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

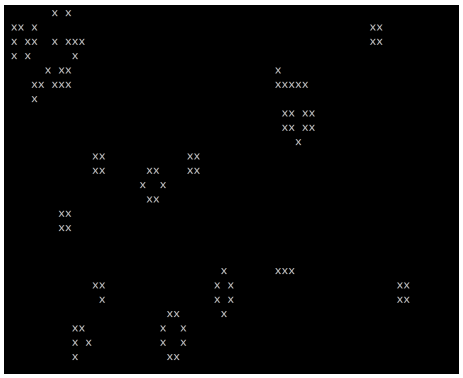
Waar de glidergun als het ware een geweer is dat continu gliders produceert, is de 'acorn'[2] het beste als een explosief te beschrijven. Hij begint simpel, de start configuratie bestaat uit slechts 7 cellen (zie Figuur 1).



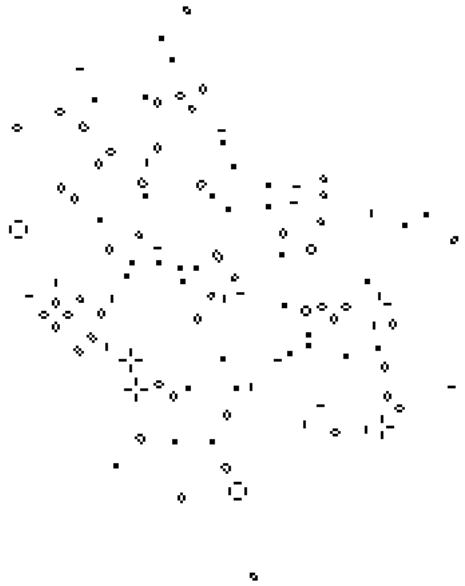
Figuur 1: De start configuratie van de acorn

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 $2 \times 2$ 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 Tijd

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`

# 4 Code

```
#include <iostream>
#include <fstream>
#include <cstring>
#include <ctime>
#include <unistd.h>


using namespace std;

// Random number generator
class RNG {
    private:
    static long random_number;

    static const long MULT = 29;
    static const long INC = 1;

    public:

    static const long MOD = 65536; // 2^16

    static void set_seed(long seed) {
        RNG::random_number = seed % MOD;
    }

    static long get_random_number() {
        return (RNG::random_number = (MULT * RNG::random_number + INC) % MOD);
    }
};

// Define storage for variable
long RNG::random_number;

class Life {
    const static int WORLD_SIZE = 1000;

    const int VIEW_WIDTH = 80;
    const int VIEW_HEIGHT = 25;

    // Position of the view
    int view_x = 0;
    int view_y = 0;

    // Position of the cursor relative
    // to the top left of the view
    int cursor_x = 0;
    int cursor_y = 0;

    int generation = 0;

    bool board[WORLD_SIZE][WORLD_SIZE] = { 0 };
    // Used to fill the actual board when computing
    // the next generation
    bool temp_board[WORLD_SIZE][WORLD_SIZE] = { 0 };

    static bool positionWithinWorld(int x, int y) {
        return x >= 0 && x < WORLD_SIZE
            && y >= 0 && y < WORLD_SIZE;
```

```
59      }
60
61      // Whether the position is on the edge of the world,
62      // the edge of the world is one character wide
63      static bool positionOnEdge(int x, int y) {
64          return ((x == -1 || x == WORLD_SIZE) && y >= -1 && y <= WORLD_SIZE) //
                Vertical edges
65              || ((y == -1 || y == WORLD_SIZE) && x >= -1 && x <= WORLD_SIZE); //
                    Horizontal edges
66
67      }
68
69      bool positionWithinView(int x, int y) {
70          return x >= view_x && x <= view_x + VIEW_WIDTH
71              && y >= view_y && y <= view_y + VIEW_HEIGHT;
72      }
73
74      bool isAlive(int x, int y) {
75          // Edges are dead by default
76          return Life::positionWithinWorld(x, y) && board[x][y];
77      }
78
79      int countLiveNeighbours(int x, int y) {
80          return isAlive(x - 1, y - 1)
81              + isAlive(x, y - 1)
82              + isAlive(x + 1, y - 1)
83              + isAlive(x - 1, y)
84              + isAlive(x + 1, y)
85              + isAlive(x - 1, y + 1)
86              + isAlive(x, y + 1)
87              + isAlive(x + 1, y + 1);
88      }
89
90      public:
91
92      // Parameters
93      // Size of changes in view or cursor position
94      int view_step_size = 1;
95      int cursor_step_size = 1;
96      // Characters for respectively dead or alive cells
97      char dead_cell = ' ';
98      char live_cell = 'x';
99      // The percentage (out of 100) of random cells picked and made alive
100     // when the view is filled randomly.
101     int random_percentage = 50;
102
103     Life() {
104         time_t rawtime;
105         time(&rawtime);
106
107         struct tm date_now = *localtime(&rawtime);
108
109         RNG::set_seed(date_now.tm_sec);
110     }
111
112     void nextGeneration() {
113         // Produce the next board according
114         // to the rules of Game of Life
115         for(int x = 0; x < WORLD_SIZE; x++)
116         for(int y = 0; y < WORLD_SIZE; y++)
117         {
```

```
118             int count = countLiveNeighbours(x, y);
119
120             if(isAlive(x, y)) {
121                 if(count == 2 || count == 3) temp_board[x][y] = true;
122                 else temp_board[x][y] = false;
123             }
124             else if(count == 3)
125             {
126                 temp_board[x][y] = true;
127             }
128         }
129
130         generation++;
131         memcpy(board, temp_board, sizeof(bool) * WORLD_SIZE * WORLD_SIZE);
132     }
133
134     void killAll() {
135         for(int x = 0; x < WORLD_SIZE; x++)
136         for(int y = 0; y < WORLD_SIZE; y++)
137         {
138             board[x][y] = false;
139         }
140     }
141
142     void killView() {
143         for(int x = view_x; x < view_x + VIEW_WIDTH; x++)
144         for(int y = view_y; y < view_y + VIEW_HEIGHT; y++)
145         {
146             board[x][y] = false;
147         }
148     }
149
150     void moveView(int x, int y) {
151         view_x += x*view_step_size;
152         view_y += y*view_step_size;
153     }
154
155     void resetCursor() {
156         cursor_x = view_x;
157         cursor_y = view_y;
158     }
159
160     void moveCursor(int x, int y) {
161         int new_x = cursor_x + x*cursor_step_size;
162         int new_y = cursor_y + y*cursor_step_size;
163
164         if(positionWithinView(new_x, new_y))
165         {
166             cursor_x = new_x;
167             cursor_y = new_y;
168         }
169     }
170
171     void toggleCursor() {
172         if(Life::positionWithinWorld(cursor_x, cursor_y)) {
173             board[cursor_x][cursor_y] = !board[cursor_x][cursor_y];
174         }
175     }
176
177     // Fills the view with random cells (determined by random_percentage)
178     void makeRandomAlive() {
```

```
179         int threshold = random_percentage * RNG::MOD / 100;
180 
181         // Then fill cells at random
182         for(int x=view_x; x<view_x+VIEW_WIDTH; x++)
183         for(int y=view_y; y<view_y+VIEW_HEIGHT; y++)
184         {
185             if(positionWithinWorld(x,y)){
186                 board[x][y] = RNG::get_random_number() < threshold;
187             }
188         }
189     }
190 
191     void printView(bool print_cursor) {
192 
193         cout << "Coordinates of view: (" << view_x << ", " << view_y << ")  ";
194         cout << "Gen: " << generation << "    ";
195         cout << "Stepsize: " << view_step_size << "/" << cursor_step_size << ", ";
196         cout << "Random Filling: " << random_percentage << "%, ";
197         cout << "Live cells: '" << live_cell << "', ";
198         cout << "Dead cells: '" << dead_cell << "'" << endl;
199 
200         if(print_cursor)
201         {
202             for(int x = view_x; x < cursor_x && x < VIEW_WIDTH; x++)
203             {
204                 cout << " ";
205             }
206             cout << " |" << endl;
207         }
208 
209 
210         for(int y = view_y; y < view_y + VIEW_HEIGHT; y++)
211         {
212             if(print_cursor)
213             {
214                 if(y == cursor_y) cout << "-";
215                 else cout << " ";
216             }
217 
218             for(int x = view_x; x < view_x + VIEW_WIDTH; x++)
219             {
220                 if(Life::positionWithinWorld(x, y) && board[x][y])
221                     cout << live_cell;
222                 else if(Life::positionOnEdge(x, y)) cout << "#";
223                 else cout << dead_cell;
224             }
225 
226             cout << endl;
227         }
228     }
229 
230     void fillViewFromFile(string filename)
231     {
232         int symbol;
233 
234         int x = view_x;
235         int y = view_y;
236 
237         ifstream file(filename);
238 
239         killAll();
```

```
240
241         while((symbol = file.get()) != EOF)
242         {
243             if(symbol == '\n')
244             {
245                 x = view_x;
246                 y += 1;
247             }
248             else
249             {
250                 if(Life::positionWithinWorld(x, y))
251                 {
252                     if(symbol == 'x') board[x][y] = true;
253                     else board[x][y] = false;
254                 }
255
256                 x += 1;
257             }
258         }
259     }
260
261     int get_maximum_cursor_step_size() {
262         return (
263             Life::VIEW_HEIGHT < Life::VIEW_WIDTH ?
264                                 Life::VIEW_HEIGHT
265                                 : Life::VIEW_WIDTH
266         ) - 1;
267     }
268
269 };
270
271 class Menu {
272
273     private:
274
275     // Gives the first non-newline character on stdin
276     static char read_character()
277     {
278         char kar;
279
280         while((kar = cin.get()) == '\n'){};
281
282         return kar;
283     }
284
285     // Reads a number from stdin, stops when number would exceed maximum
286     static int read_number(int maximum)
287     {
288         int num = 0;
289
290         char kar = read_character();
291         do {
292             if ('0' <= kar && kar <= '9')
293             {
294                 int new_num = num * 10;
295                 new_num += kar - '0';
296                 if(new_num <= maximum)
297                 {
298                     num = new_num;
299                 } else {
300                     // Prevent extra numbers from being seen as menu-input
```

```
301                     while(cin.get() != '\n'){}
302                     return num;
303                 }
304             }
305             kar = cin.get();
306         } while(kar != '\n');
307         return num;
308     }
309
310     static void print_cursor_menu()
311     {
312         cout << "Use w,a,s,d to move cursor around" << endl;
313         cout << "Press space to toggle cell" << endl;
314         cout << "Press b to return to menu" << endl;
315     }
316
317     // handles input, returns whether screen should be redrawn
318     static bool input_cursor_menu(Life *game, char input)
319     {
320         switch(input)\
321         {
322             case 'w':
323                 game->moveCursor(0, -1);
324                break;
325             case 'a':
326                 game->moveCursor(-1, 0);
327                 break;
328             case 's':
329                 game->moveCursor(0, 1);
330                 break;
331             case 'd':
332                 game->moveCursor(1, 0);
333                 break;
334
335             case ' ':
336                 game->toggleCursor();
337                 break;
338             case 'b':
339                 Menu::current_menu = Menu::MAIN;
340                 break;
341         }
342         return false;
343     }
344
345     static void print_param_menu(Life *game) {
346         cout << "Change parameters or press b to go back" << endl;
347         cout << "Numbers will be read as long as they don't exceed the given
                maximum,"
348             << " further digits are ignored" << endl;
349         cout << "1.View Step Size (" << game->view_step_size << ") ";
350         cout << "2.Cursor Step Size (" << game->cursor_step_size << ") ";
351         cout << "3.Random Filling Percentage (" << game->random_percentage << ") "
                ;
352         cout << "4.Alive Char ('" << game->live_cell << "') ";
353         cout << "5.Dead Char ('" << game->dead_cell << "') " << endl;
354     }
355
356     // handles input, returns whether screen should be redrawn
357     static bool input_param_menu(Life *game, char input){
358         switch(input){
359             case '1':
```

```
360                 game->printView(false);
361                 cout << endl << endl;
362                 cout << "Give a new step size (0-500) for view changes:"
363                     << endl;
364                 game->view_step_size = read_number(500);
365                 break;
366 
367             case '2':
368                 game->printView(false);
369                 cout << endl << endl;
370                 cout << "Give a new step size (0-" << game->
                        get_maximum_cursor_step_size()
371                     <<") for cursor changes:" << endl;
372                 game->cursor_step_size = read_number( game->
                        get_maximum_cursor_step_size() );
373                 break;
374 
375             case '3':
376                 game->printView(false);
377                 cout << endl << endl;
378                 cout << "Give a percentage (0-100) for random filling:" << endl;
379                 game->random_percentage = read_number(100);
380                 break;
381 
382             case '4':
383                 game->printView(false);
384                 cout << endl << endl;
385                 cout << "Give a new character for live cells:" << endl;
386                 game->live_cell = read_character();
387                 break;
388 
389             case '5':
390                 game->printView(false);
391                 cout << endl << endl;
392                 cout << "Give a new character for dead cells:" << endl;
393                 game->dead_cell = read_character();
394                 break;
395 
396             case 'b':
397                 current_menu = MAIN;
398                 return false;
399 
400             default:
401                 return false;
402         }
403         return true;
404     };
405 
406     static void print_main_menu()
407     {
408         cout << "1. Exit ";
409         cout << "2. Clean world ";
410         cout << "3. Clean view ";
411         cout << "4. Change parameters ";
412         cout << "5. Random ";
413         cout << "6. Toggle using cursor ";
414         cout << "7. Load glidergun.txt " << endl;
415         cout << " 8. Compute one generation ";
416         cout << "9. Run Game of Life " << endl;
417         cout << "Use w,a,s,d to move view around" << endl;
418     }
```

```
419
420     // handles input, returns whether screen should be redrawn
421     static bool input_main_menu(Life *game, char input)
422     {
423         switch(input)
424         {
425             // Movements
426             case 'w':
427                 game->moveView(0, -1);
428                 break;
429             case 'a':
430                 game->moveView(-1, 0);
431                 break;
432             case 's':
433                 game->moveView(0, 1);
434                 break;
435             case 'd':
436                 game->moveView(1, 0);
437                 break;
438
439             // Menu
440             case '1': exit(0);
441
442             case '2':
443                 game->killAll();
444                 break;
445             case '3':
446                 game->killView();
447                 break;
448             case '4':
449                 current_menu = PARAM;
450                 break;
451             case '5':
452                 game->makeRandomAlive();
453                 break;
454             case '6':
455                 game->resetCursor();
456                 current_menu = CURSOR;
457                 break;
458             case '7':
459                 game->fillViewFromFile("gliderGun.txt");
460                 break;
461             case '8':
462                 game->nextGeneration();
463                 break;
464             case '9':
465                 for(;;)
466                 {
467                     game->nextGeneration();
468                     game->printView(false);
469                     cout << endl << endl << endl;
470                     usleep(200000); // Sleep for 50ms
471                 }
472                 break;
473         }
474         return false;
475     }
476
477     public:
478
479     static int current_menu;
```

```
480
481     static const int MAIN = 0;
482     static const int CURSOR = 1;
483     static const int PARAM = 2;
484
485     static void print_menu(Life *game)
486     {
487         switch(current_menu){
488             case CURSOR:
489                 print_cursor_menu();
490                 break;
491
492             case PARAM:
493                 print_param_menu(game);
494                 break;
495
496             case MAIN:
497             default:
498                 print_main_menu();
499                 break;
500         }
501     }
502
503     // Handles input for the active menu
504     // Returns whether the screen should be redrawn
505     static bool handle_input(Life *game, char input){
506         if(input == '\n')
507         {
508             return true;
509         } else {
510             switch(current_menu){
511                 case CURSOR:
512                     return input_cursor_menu(game, input);
513                     break;
514
515                 case PARAM:
516                     return input_param_menu(game, input);
517                     break;
518
519                 case MAIN:
520                 default:
521                     return input_main_menu(game, input);
522                     break;
523             }
524         }
525     }
526 };
527
528 int Menu::current_menu = Menu::MAIN;
529
530 int main()
531 {
532     Life *game = new Life();
533     Menu::current_menu = Menu::MAIN;
534
535     game->printView(false);
536     Menu::print_menu(game);
537
538     for(;;) {
539         char input = cin.get();
540         if(Menu::handle_input( game, input) ) {
```

```
541                 game->printView( Menu::current_menu == Menu::CURSOR );
542                 Menu::print_menu(game);
543             };
544         }
545 
546         return 0;
547     }
```