Verslag derde programmeeropdracht

Alex Keizer & Léon van Velzen

10 november 2017

1 Uitleg programma

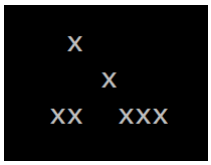
Life is origineel bedacht door John Conway. De "wereld"bestaat uit een 2-dimensionaal vlak van cellen, die levend of dood zijn. Een volgende generatie wordt d.m.v. twee simpele regels bepaald:

- 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

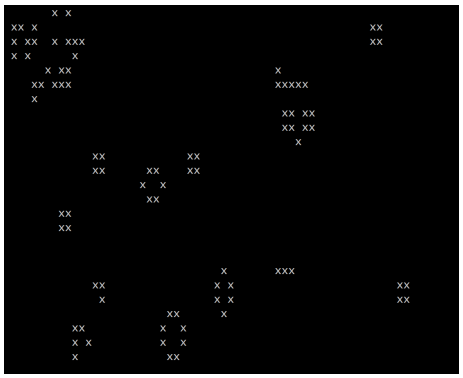
Waar de glidergun als het ware een geweer is dat continu gliders produceert, is de 'acorn'[2] het beste als een explosief te beschrijven. Hij begint simpel, de start configuratie bestaat uit slechts 7 cellen (zie Figuur 1).



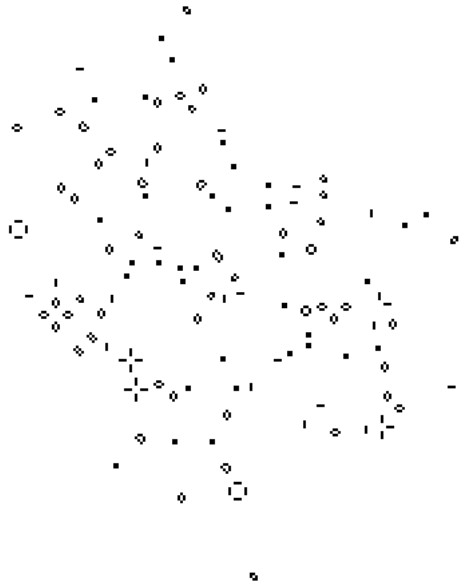
Figuur 1: De start configuratie van de acorn

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 Tijd

Tijd in uren besteed per week:

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

Laatste wijziging op 10/11/2017

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;

    // Absolute position of the cursor
    int cursor_x = 0;
    int cursor_y = 0;

    int generation = 0;

    // Parameters
    // Size of changes in view or cursor position
    int view_step_size = 1;
    int cursor_step_size = 1;
    // Characters for respectively dead or alive cells
    char dead_cell = ' ';
```

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

```
115         for(int y = 0; y < WORLD_SIZE; y++)
116         {
117             int count = countLiveNeighbours(x, y);
118 
119             if(isAlive(x, y)) {
120                 if(count == 2 || count == 3) temp_board[x][y] = true;
121                 else temp_board[x][y] = false;
122             }
123             else if(count == 3)
124             {
125                 temp_board[x][y] = true;
126             }
127             else
128             {
129                 temp_board[x][y] = false;
130             }
131         }
132 
133         generation++;
134         memcpy(board, temp_board, sizeof(bool) * WORLD_SIZE * WORLD_SIZE);
135     }
136 
137     void killAll() {
138         for(int x = 0; x < WORLD_SIZE; x++)
139         for(int y = 0; y < WORLD_SIZE; y++)
140         {
141             board[x][y] = false;
142         }
143     }
144 
145     void killView() {
146         for(int x = view_x; x < view_x + VIEW_WIDTH; x++)
147         for(int y = view_y; y < view_y + VIEW_HEIGHT; y++)
148         {
149             if(positionWithinWorld(x,y))
150                 board[x][y] = false;
151         }
152     }
153 
154     // Moves the view, won't go further than -1 or WORLD_SIZE
155     void moveView(int x, int y) {
156         int new_x = view_x + x*view_step_size;
157         int new_y = view_y + y*view_step_size;
158 
159         if(new_x < 0) view_x = -1;
160         else if(new_x > WORLD_SIZE - VIEW_WIDTH)
161             view_x = WORLD_SIZE - VIEW_WIDTH + 1;
162         else view_x = new_x;
163 
164         if(new_y < 0) view_y = -1;
165         else if(new_y > WORLD_SIZE - VIEW_HEIGHT)
166             view_y = WORLD_SIZE - VIEW_HEIGHT + 1;
167         else view_y = new_y;
168     }
169 
170     void resetCursor() {
171         cursor_x = view_x;
172         cursor_y = view_y;
173     }
174 
175     void moveCursor(int x, int y) {
```

```
176         int new_x = cursor_x + x*cursor_step_size;
177         int new_y = cursor_y + y*cursor_step_size;
178 
179         if(positionWithinView(new_x, new_y))
180         {
181             cursor_x = new_x;
182             cursor_y = new_y;
183         }
184     }
185 
186     void toggleCursor() {
187         if(Life::positionWithinWorld(cursor_x, cursor_y)) {
188             board[cursor_x][cursor_y] = !board[cursor_x][cursor_y];
189         }
190     }
191 
192     // Fills the view with random cells (determined by random_percentage)
193     void makeRandomAlive() {
194         int threshold = random_percentage * RNG::MOD / 100;
195 
196         // Then fill cells at random
197         for(int x=view_x; x<view_x+VIEW_WIDTH; x++)
198         for(int y=view_y; y<view_y+VIEW_HEIGHT; y++)
199         {
200             if(positionWithinWorld(x,y)){
201                 board[x][y] = RNG::get_random_number() < threshold;
202             }
203         }
204     }
205 
206     void printView(bool print_cursor) {
207 
208         cout << "Coordinates of view: (" << view_x << ", " << view_y << ")  ";
209         cout << "Gen: " << generation << "    ";
210         cout << "Stepsize: " << view_step_size << "/" << cursor_step_size << ", ";
211         cout << "Alive: '" << live_cell << "', ";
212         cout << "Dead: '" << dead_cell << "'" << endl;
213 
214         if(print_cursor)
215         {
216             for(int x = view_x; x < cursor_x && x < view_x + VIEW_WIDTH; x++)
217             {
218                 cout << " ";
219             }
220             cout << " |" << endl;
221         }
222 
223 
224         for(int y = view_y; y < view_y + VIEW_HEIGHT; y++)
225         {
226             if(print_cursor)
227             {
228                 if(y == cursor_y) cout << "-";
229                 else cout << " ";
230             }
231 
232             for(int x = view_x; x < view_x + VIEW_WIDTH; x++)
233             {
234                 if(Life::positionWithinWorld(x, y) && board[x][y])
235                     cout << live_cell;
236                 else if(Life::positionOnEdge(x, y)) cout << "#";
```

```
237                 else cout << dead_cell;
238             }
239 
240             cout << endl;
241         }
242     }
243 
244     void fillViewFromFile(string filename)
245     {
246         int symbol;
247 
248         int x = view_x;
249         int y = view_y;
250 
251         ifstream file(filename);
252 
253         killAll();
254 
255         while((symbol = file.get()) != EOF)
256         {
257             if(symbol == '\n')
258             {
259                 x = view_x;
260                 y += 1;
261             }
262             else
263             {
264                 if(Life::positionWithinWorld(x, y))
265                 {
266                     if(symbol == 'x') board[x][y] = true;
267                     else board[x][y] = false;
268                 }
269 
270                 x += 1;
271             }
272         }
273     }
274 
275     int get_maximum_cursor_step_size() {
276         return (
277             Life::VIEW_HEIGHT < Life::VIEW_WIDTH ?
278                                 Life::VIEW_HEIGHT
279                                 : Life::VIEW_WIDTH
280         ) - 1;
281     }
282 
283     // Parameter getters and setters
284 
285     int get_view_step_size() {return view_step_size;}
286     int get_cursor_step_size() {return cursor_step_size;}
287     char get_dead_cell() {return dead_cell;}
288     char get_live_cell(){return live_cell;}
289     int get_random_percentage(){return random_percentage;}
290 
291     void set_view_step_size(int new_size)
292     {
293         if(0 <= new_size && new_size <= 500)
294             view_step_size = new_size;
295     }
296 
297     void set_cursor_step_size(int new_size)
```

```
298     {
299         if(0 <= new_size && new_size <= get_maximum_cursor_step_size())
300             cursor_step_size = new_size;
301     }
302 
303     void set_dead_cell(char new_char)
304     {
305         if(new_char != '\t' && new_char != '\n')
306             dead_cell = new_char;
307     }
308 
309     void set_live_cell(char new_char)
310     {
311         if(new_char != '\t' && new_char != '\n')
312             live_cell = new_char;
313     }
314 
315     void set_random_percentage(int new_percentage)
316     {
317         if(0 <= new_percentage && new_percentage <= 100)
318             random_percentage = new_percentage;
319     }
320 
321 };
322 
323 class Menu {
324 
325     private:
326 
327     // Gives the first non-newline character on stdin
328     static char read_character()
329     {
330         char kar;
331 
332         while((kar = cin.get()) == '\n'){};
333 
334         return kar;
335     }
336 
337     // Reads a number from stdin, stops when number would exceed maximum
338     static int read_number(int maximum)
339     {
340         int num = 0;
341 
342         char kar = read_character();
343         do {
344             if ('0' <= kar && kar <= '9')
345             {
346                 int new_num = num * 10;
347                 new_num += kar - '0';
348                 if(new_num <= maximum)
349                 {
350                     num = new_num;
351                 } else {
352                     // Prevent extra numbers from being seen as menu-input
353                     while(cin.get() != '\n'){}
354                     return num;
355                 }
356             }
357             kar = cin.get();
358         } while(kar != '\n');
```

```
359         return num;
360     }
361
362     static void print_cursor_menu()
363     {
364         cout << "Use w,a,s,d to move cursor around" << endl;
365         cout << "Press space to toggle cell" << endl;
366         cout << "Press b to return to menu" << endl;
367     }
368
369     // handles input, returns whether screen should be redrawn
370     static bool input_cursor_menu(Life *game, char input)
371     {
372         switch(input)\
373         {
374             case 'w':
375                 game->moveCursor(0, -1);
376                break;
377             case 'a':
378                 game->moveCursor(-1, 0);
379                 break;
380             case 's':
381                 game->moveCursor(0, 1);
382                 break;
383             case 'd':
384                 game->moveCursor(1, 0);
385                 break;
386
387             case ' ':
388                 game->toggleCursor();
389                 break;
390             case 'b':
391                 Menu::current_menu = Menu::MAIN;
392                 break;
393         }
394         return false;
395     }
396
397     static void print_param_menu(Life *game) {
398         cout << endl;
399         cout << "Change parameters or press b to go back" << endl;
400         cout << "1.View Step Size (" << game->get_view_step_size() << ") ";
401         cout << "2.Cursor Step Size (" << game->get_cursor_step_size() << ") ";
402         cout << "3.Random Filling Percentage (" << game->get_random_percentage()
403                 << ") ";
404         cout << "4.Alive Char ('" << game->get_live_cell() << "') ";
405         cout << "5.Dead Char ('" << game->get_dead_cell() << "') " << endl;
406     }
407
408     // handles input, returns whether screen should be redrawn
409     static bool input_param_menu(Life *game, char input){
410         switch(input){
411             case '1':
412                 game->printView(false);
413                 cout << endl;
414                 cout << "Numbers will be read as long as they don't exceed the "
415                     << "given maximum, further digits are ignored" << endl;
416                 cout << "Give a new step size (0-500) for view changes:"
417                     << endl;
418                 game->set_view_step_size( read_number(500) );
419                 break;
```

```
420
421             case '2':
422                 game->printView(false);
423                 cout << endl;
424                 cout << "Numbers will be read as long as they don't exceed the "
425                     << "given maximum, further digits are ignored" << endl;
426                 cout << "Give a new step size (0-" << game->
                        get_maximum_cursor_step_size()
427                     <<") for cursor changes:" << endl;
428                 game->set_cursor_step_size(
429                         read_number( game->get_maximum_cursor_step_size() )
430                     );
431                 break;
432
433             case '3':
434                 game->printView(false);
435                 cout << endl;
436                 cout << "Numbers will be read as long as they don't exceed the "
437                     << "given maximum, further digits are ignored" << endl;
438                 cout << "Give a percentage (0-100) for random filling:" << endl;
439                 game->set_random_percentage( read_number(100) );
440                 break;
441
442             case '4':
443                 game->printView(false);
444                 cout << endl << endl;
445                 cout << "Give a new character for live cells: "
446                     << "(Tab is not allowed)" << endl;
447                 game->set_live_cell( read_character() );
448                 break;
449
450             case '5':
451                 game->printView(false);
452                 cout << endl << endl;
453                 cout << "Give a new character for dead cells: "
454                     << "(Tab is not allowed)" << endl;
455                 game->set_dead_cell( read_character() );
456                 break;
457
458             case 'b':
459                 current_menu = MAIN;
460                 return false;
461
462             default:
463                 return false;
464         }
465         return true;
466     };
467
468     static void print_main_menu()
469     {
470         cout << "1. Exit ";
471         cout << "2. Clean world ";
472         cout << "3. Clean view ";
473         cout << "4. Change parameters ";
474         cout << "5. Random ";
475         cout << "6. Toggle using cursor ";
476         cout << "7. Load glidergun.txt";
477         cout << " 8. Compute one generation ";
478         cout << "9. Run Game of Life " << endl;
479         cout << "Use w,a,s,d to move view around" << endl;
```

```
480     }
481
482     // handles input, returns whether screen should be redrawn
483     static bool input_main_menu(Life *game, char input)
484     {
485         switch(input)
486         {
487             // Movements
488             case 'w':
489                 game->moveView(0, -1);
490                 break;
491             case 'a':
492                 game->moveView(-1, 0);
493                 break;
494             case 's':
495                 game->moveView(0, 1);
496                 break;
497             case 'd':
498                 game->moveView(1, 0);
499                 break;
500
501             // Menu
502             case '1': exit(0);
503
504             case '2':
505                 game->killAll();
506                 break;
507             case '3':
508                 game->killView();
509                 break;
510             case '4':
511                 current_menu = PARAM;
512                 break;
513             case '5':
514                 game->makeRandomAlive();
515                 break;
516             case '6':
517                 game->resetCursor();
518                 current_menu = CURSOR;
519                 break;
520             case '7':
521                 game->fillViewFromFile("gliderGun.txt");
522                 break;
523             case '8':
524                 game->nextGeneration();
525                 break;
526             case '9':
527                 for(;;)
528                 {
529                     game->nextGeneration();
530                     game->printView(false);
531                     cout << endl << endl << endl;
532                     usleep(200000); // Sleep for 50ms
533                 }
534                 break;
535         }
536         return false;
537     }
538
539     public:
540
```

```
541     static int current_menu;
542 
543     static const int MAIN = 0;
544     static const int CURSOR = 1;
545     static const int PARAM = 2;
546 
547     static void print_menu(Life *game)
548     {
549         switch(current_menu){
550             case CURSOR:
551                 print_cursor_menu();
552                 break;
553 
554             case PARAM:
555                 print_param_menu(game);
556                 break;
557 
558             case MAIN:
559             default:
560                 print_main_menu();
561                 break;
562         }
563     }
564 
565     // Handles input for the active menu
566     // Returns whether the screen should be redrawn
567     static bool handle_input(Life *game, char input){
568         if(input == '\n')
569         {
570             return true;
571         } else {
572             switch(current_menu){
573                 case CURSOR:
574                     return input_cursor_menu(game, input);
575                     break;
576 
577                 case PARAM:
578                     return input_param_menu(game, input);
579                     break;
580 
581                 case MAIN:
582                 default:
583                     return input_main_menu(game, input);
584                     break;
585             }
586         }
587     }
588 };
589 
590 int Menu::current_menu = Menu::MAIN;
591 
592 int main()
593 {
594     Life *game = new Life();
595     Menu::current_menu = Menu::MAIN;
596 
597     game->printView(false);
598     Menu::print_menu(game);
599 
600     for(;;) {
601         char input = cin.get();
```

```
602             if(Menu::handle_input( game, input) ) {
603                 game->printView( Menu::current_menu == Menu::CURSOR );
604                 Menu::print_menu(game);
605             };
606         }
607 
608         return 0;
609     }
```