# Course $C^{++}$, Exercise 11

28.05.2013

The *game of life* was invented by John Conway in 1970. It is a game in which figures develop by themselves in a two dimensional grid. The grid is built-up from booleans. The grid can be infinite in principle, but of course we can implement only a finite grid. At each point time $t$, it is computed for each boolean in the grid, what will be the state of the boolean on time $t + 1$. The computation rules are as follows:

1. If a cell is on, and has less than two neighbours that are on, it will be off on the next time moment.
2. If a cell is on, and has two or three neighbours that are on, it will stay on on the next time moment.
3. If a cell is on, and has more than three neighbours that are on, it will be off on the next time moment.
4. If a cell is off, and has more than three neighbours that are on, it will be on on the next moment.

See also `http://en.wikipedia.org/wiki/Conway%27s_Game_of_Life`.

1. Use as a starting point:

```
#include <initializer_list>
#include <iostream>
#include <vector>


class grid
{


   struct cell
   {
      bool s0;   // Current state.
      bool s1;   // Next state.
```

```
      cell( )
         : s0( false ),
           s1( false )
      { }
   };


   unsigned int xsize;
   unsigned int ysize;
   cell* c;


public:
   // Points in the grid are indexed in the computer graphics way,
   // not in the matrix way.
   // (0,0) is the left bottom corner.
   // (x-1,0) is the right bottom corner.
   // (0,y-1) is the left top corner.
   // (x-1,y-1) is the right top corner.

   grid( unsigned xsize, unsigned int ysize )
      : xsize( xsize ),
        ysize( ysize ),
        c( new cell [ xsize * ysize ] )
   { }

   grid( const grid& g );
   void operator = ( const grid& g );
   ~grid( );

   cell* operator [] ( unsigned int x )
   {
      return c + x * ysize;
   }

   const cell* operator [] ( unsigned int x ) const
   {
      return c + x * ysize;
   }

   void clear( );

   void add( unsigned int x, unsigned int y,
             const std::vector< std::vector< bool >> & );
```

Implement the methods.

2. Write method `nextgeneration( )`.

3. Write a method `ostream& operator << ( std::ostream& , const grid& )`
   It should print the grid in such a way that the orientation is correct, so that you can see what is going on. You can use dots and X's to dintinguish cells that are on from those that are of.

4. We are going to try to plot the grid. For this we use OpenGL (`http://www.opengl.org/`) and SFML, (`http://www.sfml-dev.org/`) which are installed in room 7, and hopefully also in 137.

   Insert

   ```
   sf::Window app( sf::VideoMode( 1024, 812, 32 ), "Game of Life" );
   app. Display( );
   ```

   at the beginning of `main( )`. This should open a window, in which you can plot the grid. Insert

   ```
   #include <SFML/System.hpp>
   #include <SFML/Window.hpp>
   ```

   at the beginning of the program. Since SFML uses some additional library, the option `-lGLU -lsfml-graphics -lsfml-window -lsfml-system` must be passed to the linker. You can use the following Makefile:

   ```
   Flags = -Wreturn-type -pedantic -pedantic-errors -Wundef -std=c++11
   Link = -lGLU -lsfml-graphics -lsfml-window -lsfml-system
   CPP = g++

   life : life.o grid.o
      $(CPP) -o life $(Flags) $(Link) grid.o life.o

   life.o : life.cpp life.h
      $(CPP) -c $(Flags) life.cpp -o life.o

   grid.o : grid.cpp life.h
      $(CPP) -c $(Flags) grid.cpp -o grid.o
   ```

   Before plotting you must clear the screen, and set the projection:

   ```
   glClearColor( 0.1, 0.1, 0.1, 1.0 );
   glClearDepth( 1.0 );
   ```

```
glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );

glMatrixMode( GL_PROJECTION );
glLoadIdentity( );
glScalef( 1.0 / xsize, 1.0 / ysize, 1.0 );
```

Before you can plot a point, you have to set a color: `gColor3f( R, G, B );` The RGB values are floats between 0 and 1.

In order to plot a square, use:

```
glBegin( GL_POLYGON );
   glVertex3f( x, y, 0 );
   glVertex3f( x + 1, y, 0 );
   glVertex3f( x + 1, y + 1, 0 );
   glVertex3f( x, y + 1, 0 );
glEnd( );
```

Polygons must always be plotted left-rotating.

After all plotting is complete, use `glFlush( )` to empty any command buffers. Using this, implement a method `plot( ) const` that plots the grid.

5. In order to let the program run at reasonable speed, use

```
sf::Clock time;
time. Reset( );
```

Use `time. GetElapsedTime( )` to decide when a next field can be drawn. (This is very inefficient, because it wastes CPU time when nothing is drawn, but I don't know how to let a job sleep in SFML.)