# ComS 229 Project 2 — prototype

## The Game of Life

October 21, 2013

# 0 Introduction

The "Game of Life" (GoL in this document) is a very simple but interesting model of life, devised by John Conway in the 1970's. It is not really a "game", because there are no players; it is probably more accurate to call it a "simulation". For this project, you will write some C++ programs to simulate GoL.

In GoL, the terrain is a two–dimensional grid of square cells, as shown in Figure 1. In this document, cells will be indexed using $x$ and $y$ integer coordinates. Theoretically, the terrain is infinitely large in all directions. At any point in time, each cell in the grid may be in one of two possible states: *alive* or *dead*. Time is also discretized, and everything evolves together in "steps" or "generations". The state of a cell evolves over time according to the states of its eight *neighbors* (see Figure 2), according to the following rules.

1. If a cell is *alive* in generation $i$, then:
   (a) if there are fewer than two *alive* neighbors in generation $i$, then the cell will be *dead* in generation $i + 1$ (from loneliness).
   (b) if there are two or three *alive* neighbors in generation $i$, then the cell will be *alive* in generation $i+1$.
   (c) if there are more than three *alive* neighbors in generation $i$, then the cell will be *dead* in generation $i + 1$ (from over-population).
2. If a cell is *dead* in generation $i$, then:
   (a) if there are exactly three *alive* neighbors in generation $i$, then the cell will be *alive* in generation $i+1$ (from reproduction); otherwise, it will be *dead*.

A simple example of evolution from one generation to another is shwon in Figure 3.

## 0.0 Warning 0: These are 'loose' specs

This document does not attempt to *rigorously* define what your code must do. There *are* cases that are left unspecified: some on purpose, and some because I did not think of everything. Similarly, the project is quite open–ended: there are several different, reasonable designs that can work effectively; it is up to you to choose one.

## 0.1 Warning 1: There will be more

Design your classes carefully, with future expansion in mind. You *will* be graded on your class design, as part of the "style" portion of the project.

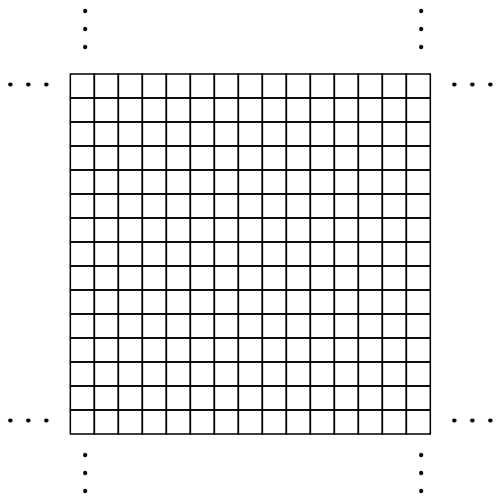

Figure 1: The terrain in The Game of Life. The grid continues forever in all directions.

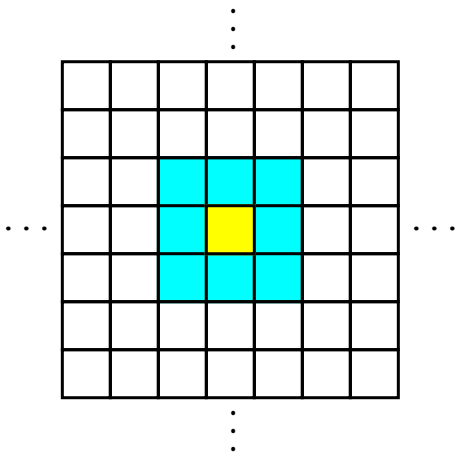

Figure 2: The eight blue cells are the neighbors of the yellow cell.

# 1 The programs

You must write the programs described below, in C++, with the specified executable names. As a general rule, your programs should be "user friendly". That means they should print useful error messages whenever possible. Additionally, your programs should print any messages or diagnostic information to standard error, with standard output reserved for output. I do not require that your programs be absolutely "quiet", but take care that the default behavior prints at most a few messages. Finally, you may implement additional switches or features if you like.

## 1.0 `life`

This program reads a single input file, whose format is specified in Appendix A, either from a pathname passed as an argument, or from standard input if no file names are given as arguments. The input file

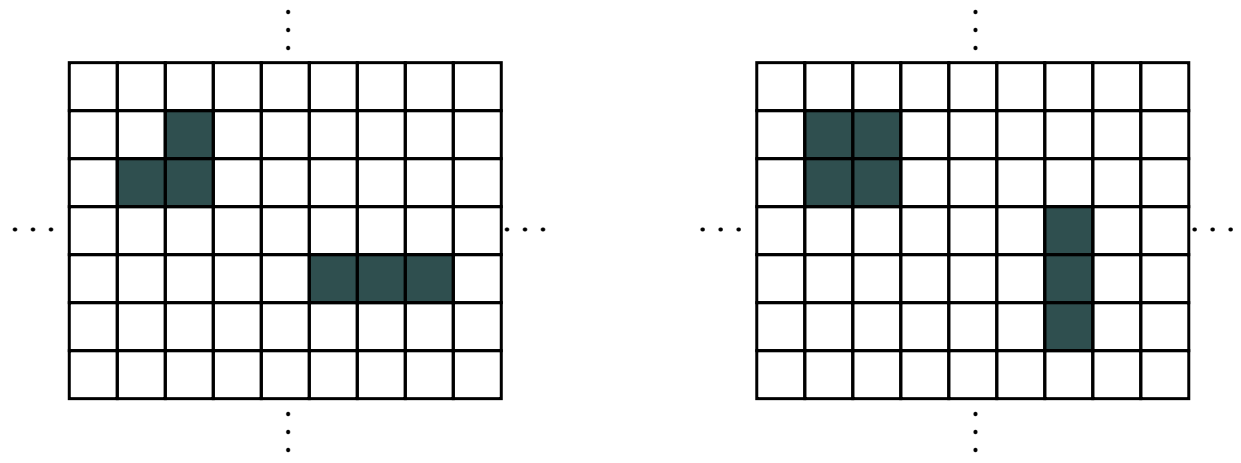


Figure 3: Example of evolution according to GoL rules. Grey squares represent *alive* cells and white ones are *dead* cells.

```
~~~~~~~~~~~~~
~~~~@@~~~~~~~
~~~~@@~~~~~~~
~~~~~~~~~@~~~
~~~~~~~~~@~~~
~~~~~~~~~@~~~
~~~~~~~~~~~~~
```

Figure 4: Visual output of `life`, for the file in Figure 6, generation 1.

specifies the size of the terrain and the initial configuration for generation 0. The program determines and displays the state of each cell at generation $n$. The program should accept the following command–line switches.

| | |
|---|---|
| `-f` | Output should be in the same file format as Appendix A. |
| `-g` $n$ | Specify the desired generation number. If omitted, the default should be $n = 0$. |
| `-h` | Display a help screen that describes the program. |
| `-tx` $l..h$ | Set the $x$ range of the terrain; overrides values specified in the input file. |
| `-ty` $l..h$ | Set the $y$ range of the terrain; overrides values specified in the input file. |
| `-v` | Output should be *visual*: an ASCII display of the terrain with appropriate characters for the *dead* and *alive* cells. In Figure 4, the dead character is "`~`" and the alive character is "`@`". |
| `-wx` $l..h$ | Set the $x$ range of the output window; otherwise this defaults to the $x$ range of the terrain. |
| `-wy` $l..h$ | Set the $y$ range of the output window; otherwise this defaults to the $y$ range of the terrain. |

Note that ranges $l..h$ specify that the value $v$ is between $l$ and $h$, inclusive (i.e., $l \leq v \leq h$).

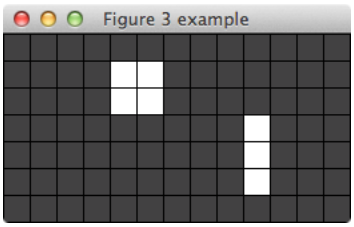


Figure 5: Display of `life_gui`, for the file in Figure 6, generation 1, size 20.

### 1.1 `life_gui`

This is a graphical version of the `life` program. Standard C++ cannot do this by itself, so you will use the Qt 4 library (an open–source, C++ library for GUIs; this is already installed on `pyrite`). You are strongly encouraged to download "C++ GUI Programming with Qt 4" from Blackboard, and to work through all examples in Chapter 1, the first example in Chapter 2 (up to "Rapid Dialog Design"), the icon editor example in Chapter 5, and the scrolling example in Chapter 6.

The program reads a single input file, whose format is specified in Appendix A, either from a pathname passed as an argument, or from standard input if no file names are given as arguments. The program determines and displays the state of each cell at generation $n$, as a grid and using the colors specified for each state (see Figure 5 for an example). The window title should be the `Name` specified in the input file. The program should accept the following command–line switches.

| | |
|---|---|
| `-g` $n$ | Specify the desired generation number. If omitted, the default should be $n = 0$. |
| `-h` | Display a help screen that describes the program, and terminate before starting the GUI. |
| `-s` $n$ | Specify the size, in pixels, of each square in the grid (default should be 10). Grid lines should be omitted when the size is less than 4. |
| `-tx` $l..h$ | Set the $x$ range of the terrain; overrides values specified in the input file. |
| `-ty` $l..h$ | Set the $y$ range of the terrain; overrides values specified in the input file. |
| `-wx` $l..h$ | Set the (initial) $x$ range of the output window (default is terrain $x$ range). |
| `-wy` $l..h$ | Set the (initial) $y$ range of the output window (default is terrain $y$ range). |

Note that for the final version, you will add "controls" for the simulator, and many of the values specified above with switches will be changed interactively.

## 2 Limits

Your terrain will be bounded in both the $x$ and $y$ directions, and will be a finite rectangle. You may assume that the cells outside of this rectangle remain in the *dead* state, regardless of what happens inside the rectangle.

## 3 A note on working together

This assignment is intended as an *individual* project. However, some amount of discussion with other students is expected and encouraged. The discussion below should help resolve the grey area of "how much collaboration is too much".

### 3.0 Not allowed

Basically, any activity where a student is able to bypass intended work for the project, is not allowed. For example, the following are definitely not allowed.

- Working in a group (i.e., treating this as a "group project").
- Posting or sharing code.
- Discussing solutions at a level of detail where someone is likely to duplicate your code.
- Using a snippet of code found on the Internet, that implements part of the assignment. If you have any doubt, check with the instructor first.

As a general rule, if you cannot honestly say that the code is yours (including the ideas behind it), then you should not turn it in.

### 3.1 Allowed

- Sharing test files (please post them on Piazza).
- Discussions to clarify assignment requirements, file formats, etc.; again, please post these on Piazza.
- High–level problem solving (but be careful — this is a slippery slope).
- Generic C++ discussions. If you have trouble with your code and can distill it down to a short, generic example, then this may be posted on Piazza for discussion.

## 4 Submitting your work

You should turn in a gzipped tarball containing all source code, a makefile, and a `README` file that documents your work. The tarball should be uploaded in Blackboard.

Your executables will be tested on `pyrite.cs.iastate.edu`. You should **test early, and test often** on `pyrite`. We will use some scripts to check your code; this means you should not change the name of the executables.

## 5 Grading

The following distribution of points will be used for grading the "prototype".

**life** : 200 points

**life_gui** : 200 points

**makefile** : 50 points

Simply typing "`make`" should build all of your executables. Also, "`make clean`" should remove the executables and any other automatically–generated files. You may include other targets for your convenience as you choose (e.g., "`make tarball`"). Note that this `Makefile` will need to be more clever than the last project, because you will need to let Qt automatically generate a makefile for `life_gui`.

Also, note that if your code does not *build* because of compiler or linker errors (either using `make` or by hand), you will likely lose much more than 50 points. Effectively: the TAs reserve the right to not grade your project *at all* if it fails to compile or link. **Test early, and test often on `pyrite`.**

**Documentation & style** : 50 points

Based on the `README` file only, which should contain a brief description of what the executables do (targeted at an end user), a brief description of your class and code design, and how the source code is broken into files (targeted at a programmer who needs to modify your code).

**Total prototype** : 500 points

# A Input file format

The files for this project are free–form text files that contain statements of the form

```
Identifier = value;
```

where the `value` may take the following forms, based on the Identifier:

- a quoted string;
- an integer;
- a range of integers, which has the form `low..high`;
- a triple of integers, which has the form `(a, b, c)`;
- a list of pairs, which has the form `(x1, y1), (x2, y2), ...`;
- a struct, which has the form `{ <stmts> }`, where `<stmts>` are zero or more statements.

Each statement may include an arbitrary amount of whitespace (including newlines) anywhere, except in the middle of an identifier or in the middle of an integer (effectively, the same rules as the C++ programming language). Unless it appears in a quoted string, the character '`#`' means to ignore the rest of the line, and counts as a section of whitespace. In the input file, a newline character has significance *only* for ending comments that start with '`#`'. When '`#`' appears inside a quoted string, it acts as an ordinary character.

A "Game of Life" file consists of a single statement assigning a struct to the "`Life`" identifier. The following identifier assignments are allowed inside a "life struct", and may appear in any order:

| | |
|---|---|
| `Name` | as a quoted string (optional; default is empty string). Sets the name of the simulation. |
| `Terrain` | as a "grid dimension" struct (defined below). This specifies the terrain dimensions. |
| `Window` | as a "grid dimension" struct. This specifies the dimensions of the "viewing window", and is optional; if omitted, this should be the same as the terrain. |
| `Chars` | as a struct with statements to set the ASCII character code to be displayed for "`Alive`" and "`Dead`" cells. |
| `Colors` | as a struct with statements to set the RGB color values for "`Alive`" and "`Dead`" cells. |
| `Initial` | as a struct with a statement to set the "`Alive`" cells for generation 0, as a list of pairs. Any coordinates not listed here will have initial state *dead*. |

A "grid dimension" struct contains statements, each for setting identifier "`Xrange`" or "`Yrange`" to a range of integers. The file shown in Figure 6 is an example of a legal file. Note that the equivalent file shown in Figure 7 is *also legal*.

```
#
# Comments, to be ignored
#

Life =
{
  Name = "Figure 3 example";

  Terrain = {
    Xrange = -6..6;
    Yrange
    = -3 .. 3 ;
  };

  # Use the same window as the terrain
  # i.e., view everything

  Chars = {
    Alive = 64;  # ASCII code for @
    Dead = 126;  # ASCII code for ~
  };

  Colors = {
    Alive = (255, 255, 255);  # White
    Dead = (64, 64, 64);      # Dark grey
  };

  Initial = {
    # Set coordinates of the alive cells
    # everything else is dead
    Alive = (-2, 1), (-1, 2), (-1, 1),
            (2, -1), (3, -1), (4, -1);
  };
};
```

Figure 6: An example "life" file, for generation 0 as shown in Figure 3.

```
Life={Name="Figure 3 example";Terrain={Xrange=-6..6;Yrange=-3..3;};Chars={
Alive=64;Dead=126;};Colors={Alive=(255,255,255);Dead=(64,64,64);};Initial
={Alive=(-2,1),(-1,2),(-1,1),(2,-1),(3,-1),(4,-1);};};
```

Figure 7: A less readable, but still valid, "life" file (same as Figure 6).