Assignment 6 - Conways Game of Life

Submit Assignment

Due Nov 1 by 11:59pm **Points** 0 **Submitting** a file upload **File Types** hpp, cpp, txt, and in
Available Oct 22 at 12am - Nov 3 at 11:59pm 13 days

The purpose of this assignment is to give you more experience in developing C++ classes, taking advantage of inheritance and polymorphism through inheritance. The subject you'll be using for this assignment is Conway's Game of Life (**wiki** (https://en.wikipedia.org/wiki/Conway%27s_Game_of_Life)). You'll write a simulator that updates the state of a word according to a set of rules and then render that state to the console, animating it over time. I think you'll enjoy this, it is fun to play around with once it is up and going.

Assignment

Write a program that provides an interesting animation using Conway's Game of Life rules and using at least the required patterns indicated below.

Required Patterns

- Acorn
- Block
- Blinker
- Glider
- Gosper Glider Gun

All of these patterns are shown in the wiki: **link** (https://en.wikipedia.org/wiki/Conway%27s_Game_of_Life)

All patterns must be derived from the following class (Pattern.hpp):

```
#pragma once

#include <cstdint>

class Pattern
{
  public:
    virtual std::uint8_t getSizeX() const = 0;
    virtual std::uint8_t getSizeY() const = 0;
    virtual bool getCell(std::uint8_t x, std::uint8_t y) const = 0;
};
```

- `getSizeX` - Returns the horizontal width (in cells) of the pattern.
- `getSizeY` - Returns the vertical height (in cells) of the pattern.
- `getCell` - Returns `true` if the cell in the pattern is filled, `false` otherwise.

Simulator

Create a simulator class, named LifeSimulator, that can be initialized with some state (of patterns) and then update the game of life simulation. Use the following as the (required) public interface for the class.

```
class LifeSimulator
{
  public:
    LifeSimulator(std::uint8_t sizeX, std::uint8_t sizeY);

    void insertPattern(const Pattern& pattern, std::uint8_t startX, std::uint8_t startY);
    void update();

    std::uint8_t getSizeX() const { return m_sizeX; }
    std::uint8_t getSizeY() const { return m_sizeY; }
    bool getCell(std::uint8_t x, std::uint8_t y) const { return m_grid[x][y]; }
};
```

- The constructor accepts a `sizeX` and `sizeY` indicating the size of the world. You'll want to set these equal to the width/height of the console the program runs within.
- `insertPattern` - Adds the pattern to the world, with the upper left corner beginning at `startX` and `startY`.
- `update` - Performs a single step update of the world, following the four rules specified in the wiki article.
- `getSizeX` & `getSizeY` - Return the size of the world.
- `getCell` - Returns `true` if the world cell is alive, `false` otherwise.

You may add any private fields and members you like, but can not modify the public interface in any way.

Renderer

Use the following class (Renderer.hpp) as the base class from which you derive and implement a `RendererConsole` class.

```
class Renderer
{
  public:
    virtual void render(const LifeSimulator& simulation) = 0;
};
```

Again, derive a `RendererConsole` class from `Renderer` and provide an implementation that renders the `LifeSimulator` world to the console.

Use the same rlutil.h header file from the previous assignment for console rendering.A few things you'll want to do when rendering each frame of animation...

- use `rlutil::cls()` to clear the screen.
- use `rlutil::hidecursor()` before rendering.
- ...render...
- use `rlutil::showcursor()` after rendering.
- (maybe) use `rlutil::resetColor()` after rendering.

It is best to only render cells that are alive. In other words, loop through the world and when you find a cell that is alive, use `rlutil::locate` to set the location of the cursor and the use `rlutil::setChar` to draw the cell. If you draw all cells, dead or alive, the rendering is too slow.

Animation

Your program will run for some number of steps for the animation. You can use a simple counted for loop for this, but you'll want to pause for a little bit between each animation step in order for the rendering to look okay. You can use a line of code like the following to pause for some number of milliseconds...

```
std::this_thread::sleep_for(std::chrono::milliseconds(10));
```

The `<thread>` header file needs to be included for this code to compile.

Submission Notes

Use the following filenames for your code. If you have additional code files due to other patterns, that is fine, but please following the same naming convention for the files and class names (consistency is king!).

- Turn in the following files:
  - **main.cpp**
  - **LifeSimulator.hpp, LifeSimulator.cpp**
  - **Renderer.hpp**
  - **RenderConsole.hpp, RenderConsole.cpp**
  - Pattern Files
    - **Pattern.hpp**
    - **PatternAcorn.hpp, PatternAcorn.cpp**
    - **PatternBlinker.hpp, PatternBlinker.cpp**
    - **PatternBlock.hpp, PatternBlock.cpp**
    - **PatternGlider.hpp, PatternGlider.cpp**
    - **PatternGosperGliderGun.hpp, PatternGosperGliderGun.cpp**
  - **CMakeLists.txt**
- Please have your code read ***EXACTLY*** "dictionary.txt". This way the grader can use the same word list for everyone while grading.
  - You need to have this file in the folder where the program runs, which is most likely the /build folder.
- CMakeLists.txt target
  - Use ***EXACTLY*** `ConwaysLife` as the name for the `add_executable` target for the application.
- There are no unit tests associated with this assignment.
- Your code must compile without any warnings or compiler errors. See syllabus regarding code that has compiler errors.
- Your code must adhere to the CS 3460 coding standard: **link**
- Your code must be formatted through the use of the following clang-format configuration file: **link**

Sample View

The following shows an example of my program running with the Gosper Glider Gun.

ConwaysLife.mp4