Assignment 1: The Game of Life

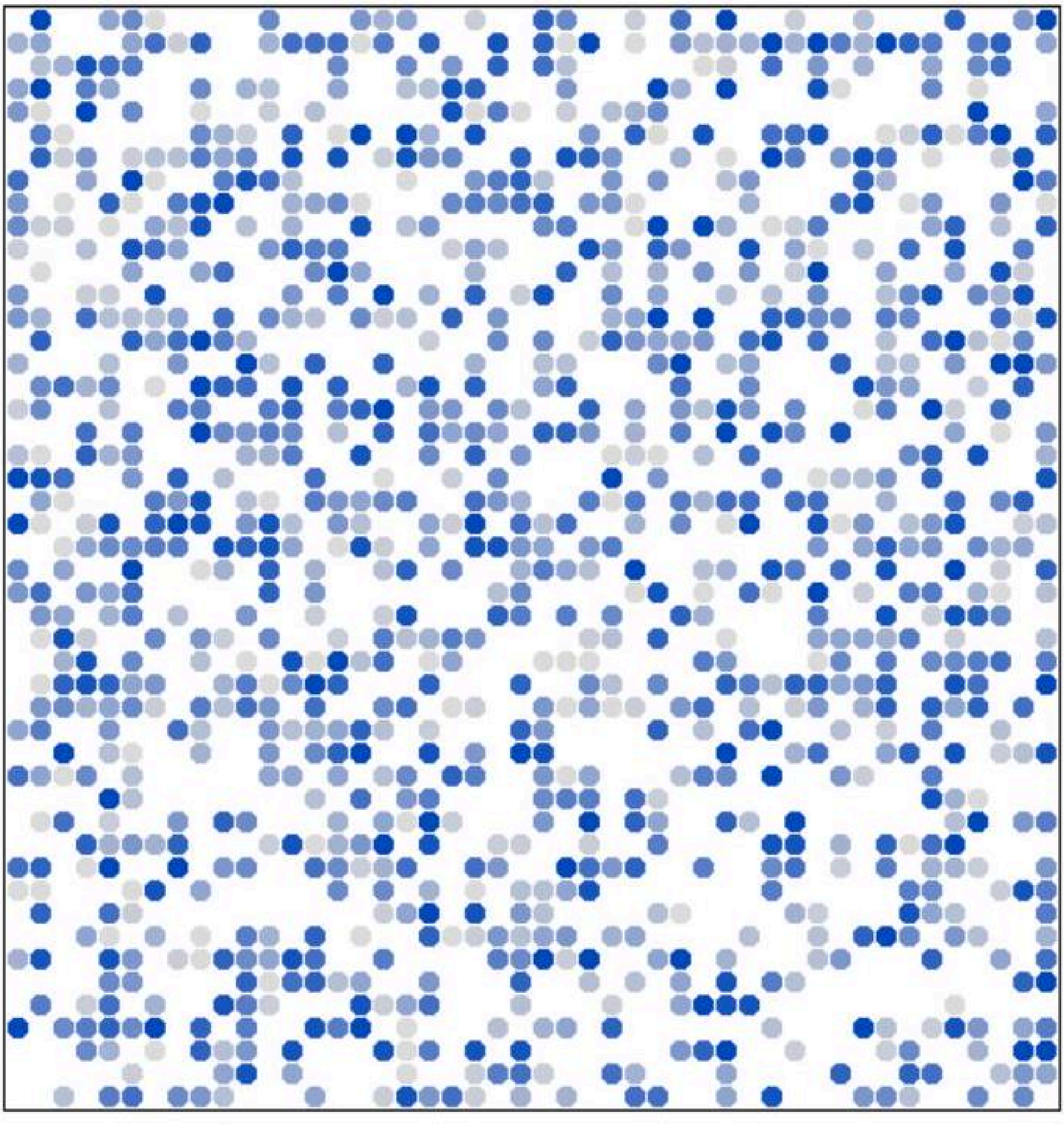
作业 1：生命的游戏

A brilliant assignment from Julie Zelenski.

这是朱莉-泽伦斯基的一项杰出任务。

Let the fun begin! Your first real assignment centers on the use of a two-dimensional grid as a data structure in a cellular simulation. It will give you practice with control structures, functions, templates, and even a bit of string and file work. More importantly, the program will exercise your ability to decompose a large problem into manageable components. What's especially stellar about this problem is that you can construct a beautiful and elegant solution if you take the time to think through a good design.

开始吧你的第一个真正任务是在细胞模拟中使用二维网格作为数据结构。它将让你练习控制结构、函数、模板，甚至是一些字符串和文件工作。更重要的是，该程序将锻炼你把一个大问题分解成可管理组件的能力。这个问题的特别之处在于，如果你花时间思考一个好的设计，你就能构建出一个漂亮而优雅的解决方案。



Project Due: Wednesday, October 2nd at 5:00 p.m.

项目提交时间：10 月 2nd 日星期三下午 5:00

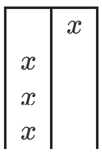
The Problem 问题所在

Your mission is to implement a version of the game of Life, originally conceived by the British mathematician J.H. Conway in 1970 and popularized by Martin Gardner in his Scientific American column. The game is a simulation that models the life cycle of bacteria. Given an initial pattern, the game simulates the birth and death of future generations using simple rules. It’s largely a lava lamp for mathematicians.

该游戏最初由英国数学家康威（J.H. Conway）于 1970 年提出，并由马丁-加德纳（Martin Gardner）在他的《科学美国人》专栏中加以推广。该游戏是一个模拟细菌生命周期的模型。给定一个初始模式，游戏利用简单的规则模拟后代的出生和死亡。它在很大程度上是数学家的熔岩灯。

The simulation is run within a two-dimensional grid. Each grid location is either empty or occupied by a single cell (X). A location’s neighbors are any cells in the eight adjacent locations. In the following example, the shaded location has three “live” neighbors:

模拟在一个二维网格内运行。每个网格位置要么是空的，要么被一个单元格（X）占据。一个位置的邻居是相邻八个位置中的任何单元格。在下面的示例中，阴影位置有三个 "活 "邻居：



The Rules 规则

The simulation starts with an initial pattern of cells on the grid and computes successive generations of cells according to the following rules:

模拟从网格上单元格的初始模式开始，按照以下规则计算单元格的连续世代：

A location that has zero or one neighbors will be empty in the next generation. If a cell was in that location, it dies of loneliness.

在下一代中，没有邻居或只有一个邻居的位置将是空的。如果一个细胞曾在该位置，那么它就会死于孤独。

A location with two neighbors is stable-that is, if it contained a cell, it still contains a cell. If it was empty, it’s still empty.

有两个邻居的位置是稳定的，也就是说，如果它包含一个单元格，它仍然包含一个单元格。如果原来是空的，现在也还是空的。

A location with three neighbors will contain a cell in the next generation. If it was unoccupied before, a new cell is born. If it currently contains a cell, the cell remains. Good times.

有三个邻居的位置在下一代中将包含一个单元。如果该位置之前无人居住，则会诞生一个新单元格。如果该位置目前有一个单元格，则该单元格保留。美好时光

A location with four or more neighbors will be empty in the next generation. If there was a cell in that location, it dies of overcrowding.

有四个或更多邻居的地方在下一代会空无一人。如果该地点曾有一个细胞，它也会因过度拥挤而死亡。

The births and deaths that transform one generation to the next must all take effect simultaneously. Thus, when computing a new generation, new births and deaths in that generation don’t impact other births and deaths in that generation. You’ll need to work on two versions of the grid-one for the current generation, and a second that allows you to compute and store the next generation without changing the current one.

一代人的出生和死亡必须同时发生。因此，在计算新世代时，该世代中新的出生和死亡不会影响该世代中其他的出生和死亡。您需要在两个版本的网格上工作--一个是当前世代的网格，另一个是允许您在不改变当前世代的情况下计算和存储下一代的网格。

Check your understanding of these rules with the following example. The two patterns below should alternate forever:

通过下面的示例检查您对这些规则的理解。下面的两个图案应该永远交替出现：

	X	X	X	

		X		
		X		
		X		

The Starting Configuration

启动配置

To get the colony underway, your program should offer the user the option to start everything off with a randomly generated grid, or to read in a starting grid from data file. If the grid is to be randomly generated, then set the grid width to be some random value between 40 and 60, inclusive, and the grid height to be some random value between 40 and 60, inclusive, and flip a fair coin to decide whether a particular cell should be occupied or unoccupied. If a cell is occupied, then set its age to be some random value between 1 and kMaxAge, inclusive.

为了让殖民地开始运行，你的程序应该让用户选择以随机生成的网格开始一切，或者从数据文件中读入一个起始网格。如果网格是随机生成的，那么网格宽度应设置为 40 至 60 之间的某个随机值，网格高度应设置为 40 至 60 之间的某个随机值，然后掷一枚公平的硬币来决定某个单元格是被占用还是未被占用。如果单元格被占用，则将其年龄设置为 1 至 kMaxAge（包括 kMaxAge）之间的某个随机值。

If the user prefers to load a starting configuration from a file, then you can bank on the following file format:

如果用户希望从文件中加载起始配置，则可以使用以下文件格式：

```
# Any line that begins with a #
# is a comment and is ignored
30                                  <- Number of rows
25                                  <- Number of columns per row
-----XXXXX-----XXXXX-----           <- Each line is one row of the grid
----------XXXXX----------           <- X means cell is live, dash is not
------XXXX-----XXXX------
    . . . and so on . . .
```

You can read the file line-by-line using the standard getline method (note that this is distinct from the CS106-specific getLine that just reads from the console). All colony files obey the above format, and you may assume that the file contents are properly formatted (i.e. it is not required that you do error checking). When read from a file, all cells start at an age of one.

你可以使用标准的 getline 方法逐行读取文件（注意，这有别于 CS106 专用的 getLine，后者只是从控制台读取）。所有殖民地文件都遵循上述格式，您可以假定文件内容格式正确（即不需要进行错误检查）。从文件中读取时，所有细胞的年龄都从 1 开始。

Managing the Grid 网格管理

Basically, what's needed for storing the grid is a two-dimensional array. However, the built-in C++ array is fairly primitive. C arrays (and thus C++ arrays) are big on efficiency but low on safety and convenience. They don't track their own length, they don't check bounds, and they require you manage computer memory in a way that isn't all that fun in an introductory assignment. We're taking a more modern approach—that of using a Grid class that provides the abstraction of a two-dimensional array in a safe, encapsulated form with various conveniences for the client. Just as the Vector (our equivalent of the Java ArrayList) provides a cleaner and less error-prone abstraction for a one-dimension array, the Grid offers the same upgrade for two-dimensional needs. These classes, among others, are from the CS106 tool set that we will be using throughout the quarter. Refer to the Queen Safety lecture examples if you want to see the Grid in action.

基本上，存储网格所需的是一个二维数组。然而，内置的 C++ 数组相当原始。C 数组（因此也是 C++ 数组）的效率很高，但安全性和便利性却很低。它们不跟踪自身长度，不检查边界，而且要求你以一种在入门作业中并不有趣的方式管理计算机内存。我们采用了一种更现代的方法--使用网格类，它以安全、封装的形式提供了二维数组的抽象，并为客户端提供了各种便利。正如 Vector（相当于 Java ArrayList）为一维数组提供了更简洁、更不易出错的抽象一样，Grid 也为二维数组提供了同样的升级。这些类和其他类都来自 CS106 工具集，我们将在整个季度中使用。如果您想了解网格的实际应用，请参阅 Queen Safety 讲座示例。

Your grid should store an integer for each location rather than a simple alive-or-dead Boolean. This allows you to track the age of each cell. Each generation that a cell lives through adds another year to its age. A cell that has just been born has age 1. If it makes it through the next generation, that cell has age 2, on the following iteration it would be 3 and so on. During the animation, cells will fade to light gray as they age to depict the cell life cycle.

您的网格应该为每个位置存储一个整数，而不是一个简单的 "活 "或 "死 "布尔值。这样就可以跟踪每个单元格的年龄。单元格每活过一代，其年龄就会增加一岁。刚出生的细胞年龄为 1。如果它活到了下一代，那么它的年龄就是 2 岁，在接下来的迭代中，它的年龄就是 3 岁，以此类推。在动画中，细胞会随着年龄的增长逐渐变为浅灰色，以描述细胞的生命周期。

As mentioned earlier, you will need two grids: one for the current generation and a separate scratch grid to set up the next one. After you have computed the entire next generation, you can copy the scratch grid contents into the current grid to get ready for the next iteration. Copying a grid is trivial—just assign one grid to another using =. The Grid class implements the regular assignment operator to do a full, deep copy.

如前所述，你需要两个网格：一个用于当前一代，另一个用于建立下一代。计算完整个下一代后，可以将从头网格的内容复制到当前网格，为下一次迭代做好准备。复制网格非常简单，只需使用 = 将一个网格赋值给另一个网格即可。网格类实现了常规赋值运算符，可以进行全面、深入的复制。

Animating the World 动画世界

We provide the life-graphics.h module that exports a LifeDisplay class to help you draw the cells in its own window. Our cell-drawing methods will slowly fade cells as they age. A newly born cell is drawn as a dark dot and with each generation the cell will slightly fade until it stabilizes as a faint gray cell. See the life-graphics.h interface in the starter files for details on the specifics of the methods we provide and how to use them.

我们提供的 life-graphics.h 模块导出了一个 LifeDisplay 类，可帮助您在自己的窗口中绘制单元格。我们的单元格绘制方法会随着单元格的老化而慢慢褪色。刚出生的细胞会被绘制成一个黑点，每一代细胞都会轻微褪色，直至稳定为一个淡灰色的细胞。有关我们提供的方法的具体细节和使用方法，请参阅启动文件中的 life-graphics.h 接口。

You should offer the user 3 options for simulation speed (slow, medium and fast); this will translate to shorter or longer pauses between generations. You decide how long the pause times are. Use what you think is reasonable. Make sure you have 3 distinct speeds. We also want you to support a manual mode where the user has to explicitly hit return to advance to the next generation. This mode is particularly handy when you are first learning the rules or need to do some careful debugging.

您应该为用户提供 3 种模拟速度选择（慢速、中速和快速）；这将转化为两代之间较短或较长的停顿时间。暂停时间的长短由您决定。使用你认为合理的时间。确保有 3 种不同的速度。我们还希望你们支持手动模式，用户必须明确地点击回车键才能进入下一代。这种模式在您初次学习规则或需要仔细调试时特别方便。

You should continue computing and drawing successive generations until the colony becomes completely stable or the user clicks the mouse button. Stability is defined as no changes between successive generations (other than live cells continuing to age) and all cells hit or exceed the constant kMaxAge (defined in “life-constants.h”). Colonies that infinitely repeat or alternate are not considered stable. When advancing to the next generation, you should check if the colony has become stable and if so, stop the simulation. If a non-manual mode has been selected, then if the user clicks the mouse button, the simulation should stop. Note, if the user has selected manual mode for the simulation speed, there is no need to check for user clicks. Instead, in manual mode, if the user enters “quit” instead of just hitting enter at the prompt, the simulation should stop. When a simulation ends, you should then offer the user a chance to start a new simulation or quit the program entirely.

您应继续计算并绘制连续世代，直到菌落完全稳定或用户点击鼠标按钮。稳定的定义是连续几代之间没有变化（活细胞继续衰老除外），并且所有细胞都达到或超过常数 kMaxAge（在 "life-constants.h "中定义）。无限重复或交替的菌落不被认为是稳定的。在进入下一代时，应检查菌落是否已经稳定，如果是，则停止模拟。如果选择的是非手动模式，那么如果用户点击鼠标按钮，模拟就会停止。请注意，如果用户选择了手动模式来提高模拟速度，则无需检查用户点击情况。相反，在手动模式下，如果用户输入 "quit（退出）"，而不是直接在提示符下按回车键，模拟就会停止。模拟结束后，用户可以重新开始模拟或完全退出程序。

Start Early 尽早开始

This assignment is likely bigger and more complicated than anything you built during your time in CS106A or AP Java. To ensure that you’re not blindsided by the sheer amount of work, we highly suggest that you start early on this assignment.

这项作业可能比您在 CS106A 或 AP Java 课程中构建的任何程序都要庞大和复杂。为了确保您不会被巨大的工作量吓到，我们强烈建议您尽早开始这项作业。

By the end of this weekend, you might have a partially working program that:

到本周末结束时，你可能会有一个可以部分运行的程序：

prints out all of the text that gets printed to the console, as illustrated by the sample application,

会打印出所有打印到控制台的文本，如示例应用程序所示、

prompts the user to initialize a grid based on the contents of a data file,

提示用户根据数据文件的内容初始化网格、

populates a grid by reading in the contents of the named file, and

通过读取指定文件的内容来填充网格，而

draws the initial state of the grid to the graphics window, waiting for a mouse click to clear the screen and start over.

将网格的初始状态绘制到图形窗口中，等待鼠标点击以清空屏幕并重新开始。

In a nutshell, the checkpoint implements the setup and all of the plumbing for the game of Life, without playing the actual simulation. Should you find yourself struggling with this suggestion, please feel free to come to office hours or get help in the LAIR.

简而言之，检查点实现了 "生命 "游戏的设置和所有管道，而无需进行实际模拟。如果您发现自己在这项建议中遇到困难，请随时到办公时间或到实验室寻求帮助。

Program Strategies 计划策略

A high-quality solution to this program will take time. It is not recommended that you wait until the night before to hack something together. It is worthwhile to map out a strategy before starting to code. Amazingly concise and elegant solutions can be constructed aim to make yours one of them!

该程序的高质量解决方案需要时间。我们不建议您等到前一天晚上才开始编写代码。在开始编码之前，制定一个策略是值得的。我们可以构建出令人惊叹的简洁而优雅的解决方案，目标是让你的解决方案成为其中之一！

Decomposition: This program is an important exercise in learning how to put decomposition to work. Decomposition is not just some frosting to spread on the cake when it’s all over-it’s the tool that helps you get the job done. With the right decomposition, this program is much easier to write, test, and debug! With the wrong decomposition, all of the above will be a chore. You want to design your functions to be of small, manageable size and of sufficient generality to do the different flavors of tasks needed. Strive to unify all the common code paths. A good decomposition will allow you to isolate the details for all tricky parts into just a few functions.

分解：这个程序是学习如何运用分解的重要练习。分解不仅仅是蛋糕上的糖霜，而是帮助你完成工作的工具。有了正确的分解，程序的编写、测试和调试就容易多了！如果分解错误，上述所有工作都会变得繁琐。您希望设计的函数小巧、易于管理，并具有足够的通用性，以完成所需的各种任务。努力统一所有常见的代码路径。良好的分解可以让您将所有棘手部分的细节分离到少数几个函数中。

Avoid special cases, don’t handle inner cells, edge cells, and corner cells differently-write general code that works for all cases. No special cases equals no special-case code to debug! Think carefully about how to design these functions to be sufficiently general for all use cases.

避免特殊情况，不要区别对待内部单元格、边缘单元格和角落单元格--编写适用于所有情况的通用代码。没有特殊情况，就没有需要调试的特殊情况代码！仔细考虑如何设计这些函数，使其足够通用于所有用例。

Incremental Development and Testing: Don’t try to solve it the entire task at once. Even though you should have a full design from the beginning, when you’re ready to implement, focus on implementing features one at a time. Start off by writing code to meet the list of goals in the Start Early section. Continue by seeding your grid with a known configuration to make it easier to verify the correctness of your simulation. Initialize your grid to be mostly empty with a horizontal bar of three cells in the middle. When you run your simulation on this, you should get the alternating bar figure on page 3 that repeatedly inverts between the two patterns. It’s easiest to get your program working on such a simple case first. Then you move on to more complicated colonies, until it all works perfectly.

增量开发和测试：不要试图一次性解决整个任务。即使你从一开始就应该有一个完整的设计，当你准备好实施时，也要集中精力一次实施一个功能。首先，编写代码以实现 "尽早开始 "部分中的目标列表。继续在网格中添加已知配置，以便更容易验证仿真的正确性。将网格初始化为大部分为空，中间有一个由三个单元格组成的水平条。在此基础上运行仿真时，您应该会看到第 3 页上的交替条形图，并在两种模式之间反复倒换。首先让程序在这种简单的情况下运行是最简单的。然后再处理更复杂的情况，直到程序完美运行为止。

Avoid the “extra-layer-around-the-grid” approach: At first glance, some students find it desirable to add a layer around the grid-an extra row and column of imaginary cells surrounding the grid. There is some appeal in forcing all cells to have exactly eight neighbors by adding “dummy” neighbors and setting up these neighbors to always be empty. However, in the long run this approach introduces more problems than it solves and leads to confusing and inelegant code. Before you waste time on it, let us tell you up front that this is a poor strategy and should be avoided. Just assume that locations off the grid are empty and can never count as neighbors.

避免 "在网格周围多加一层 "的方法：乍一看，有些学生认为在网格周围增加一层--在网格周围多加一行和一列假想的单元格--是可取的。通过添加 "假 "相邻单元格并设置这些相邻单元格始终为空，迫使所有单元格都有八个相邻单元格，这种方法有一定的吸引力。然而，从长远来看，这种方法带来的问题比解决的问题更多，而且会导致代码混乱和不流畅。在你浪费时间之前，让我们先告诉你这是一个糟糕的策略，应该避免。只需假设网格外的位置是空的，永远不能算作邻居。

Other Random Notes and Hints

其他随机说明和提示

Error checking is an important part of handling user input. Where you prompt the user for input, such as asking for file to open, you should validate that the response is reasonable and if not, ask for another. (You can assume that the data files themselves are well formed, though.)

错误检查是处理用户输入的一个重要部分。在提示用户输入（如要求打开文件）时，应验证响应是否合理，如果不合理，则要求用户重新输入。(不过你可以假定数据文件本身的格式是正确的）。

While developing, it’s helpful to circumvent the code that prompts for the configuration and just force your simulation to always open “Simple Bar” or whatever you are currently working on. This will save you from having to answer all the prompts each time when you do a test run.

在开发过程中，避开提示配置的代码，只强制模拟始终打开 "Simple Bar "或当前正在运行的程序，会很有帮助。这样就不必在每次测试运行时都回答所有提示。

Although you might be tempted, you should not use any global variables for this assignment. Global variables seem convenient, but they have a lot of drawbacks. In this class we will never use them. Always assume that global variables are forbidden unless we explicitly tell you otherwise. (Global constants are fine, though.)

虽然你可能会受到诱惑，但你不应该在这项作业中使用任何全局变量。全局变量看起来很方便，但有很多缺点。在本课中，我们将永远不会使用它们。除非我们明确告诉你禁止使用全局变量，否则请务必这样做。(全局常量倒是可以使用）。

Note that all of the data files reside in a directory called “files”. When opening a data file name “Fish”, you need to type in the filename as “files/Fish” else the ifstream constructor won’t be able to find it.

请注意，所有数据文件都位于名为 "files "的目录中。打开名为 "Fish "的数据文件时，需要输入文件名 "files/Fish"，否则 ifstream 构造函数将无法找到它。

Allowing the game to animate while constantly polling for mouse clicks is a tricky thing. Look at the "gevents. **h** " documentation to see how one can poll for mouse events without blocking. I don’t want you to invest too much energy into mouse and timer events, so I’m presenting my own approach to non-blocking mouse event detection right here.

让游戏在不断轮询鼠标点击的同时产生动画是一件棘手的事情。请查看 "gevents. **h** "文档，了解如何在不阻塞的情况下轮询鼠标事件。我不希望你在鼠标和计时器事件上投入过多精力，所以我在这里介绍我自己的无阻塞鼠标事件检测方法。

```
static void runAnimation(LifeDisplay& display, Grid<int>& board, int ms) {
    GTimer timer(ms);
    timer.start();
    while (true) {
        GEvent event = waitForEvent(TIMER_EVENT + MOUSE_EVENT);
        if (event.getEventClass() == TIMER_EVENT) {
                advanceBoard(display, board);
        } else if (event.getEventType() == MOUSE_PRESSED) {
                break;
        }
    }
    timer.stop();
}
```

You should cannibalize the code snippet above and extend it as necessary as you fold it into your own solution.

在将上面的代码片段整合到自己的解决方案中时，你应该利用它，并根据需要对其进行扩展。

Coding Standards, Commenting, Style

编码标准、注释、风格

This program has quite a bit of substance, however, the coding complexity doesn’t release you from also living up to our style standards. Carefully read over the style information we have given you and keep those goals in mind. Aim for consistency. Be conscious of the style you are developing. Proofread. Comment thoughtfully.

本程序的内容相当丰富，但是，编码的复杂性并不意味着您可以不遵守我们的风格标准。请仔细阅读我们为您提供的风格信息，并牢记这些目标。力求一致。有意识地形成自己的风格。校对。深思熟虑地发表评论。

Test Colonies 试验菌落

We have provided several colony configuration files for you to use as test cases. There is also a compiled version of our solution, so that you can see what the correct evolutionary

我们提供了几个殖民地配置文件，供您用作测试用例。我们还提供了解决方案的编译版本，这样您就可以看到正确的演化过程。

patterns should be. Some of our provided colonies evolve with interesting kaleidoscopic regularity; a few are completely stable from the get-go, and others eventually settle into a completely stable or infinite repetition or alternation. Each file has a comment at the top of the file that identifies the contributor and tells a bit of what to expect from the colony.

模式应该是这样的。我们提供的一些菌落以有趣的万花筒式规律演化；一些菌落从一开始就完全稳定，而另一些菌落则最终进入完全稳定或无限重复或交替的状态。每个文件的顶部都有一个注释，标明了贡献者的身份，并说明了对菌落的一些预期。

We’d love additional colony contributions, so if you create a beautiful bacteria ballet of your own, please send it to us and we can share it with the class.

我们希望得到更多菌落的贡献，如果您自己创作了美丽的细菌芭蕾，请发送给我们，我们将与全班分享。

Getting Started 入门

Follow the Assignments link on the class web site to find the starter bundle for Assignment 1. You should download this file to your own computer and open the bundle. For this assignment, the bundle contains some custom header files, our lifegraphics.cpp implementation, and an incomplete version of life.cpp.

点击课堂网站上的 "作业 "链接，找到作业 1 的启动包。您应将该文件下载到自己的计算机上并打开捆绑包。对于本作业，捆绑包中包含一些自定义头文件、我们的 lifegraphics.cpp 实现和一个不完整版本的 life.cpp。

Deliverables 交付成果

You only need to electronically submit those files that you modified, which in this case, will probably just be life.cpp. There are instructions on the course web site outlining how to electronically submit on the assignments page. You are responsible for keeping a safe copy to ensure that there is a backup in case your submission is lost or the electronic submission is corrupted. You are also responsible for checking to make sure your submission was correctly received by checking your submitted assignments on the submission site (it should show you past submissions and the files they contain). Make sure your submission contains the correct files.

您只需要以电子方式提交您修改过的文件，在这种情况下，可能只提交 life.cpp。课程网站的 "作业 "页面上有如何提交电子文档的说明。您有责任保存一份安全的副本，以确保万一丢失或电子提交文件损坏时有备份。您也有责任在作业提交网站上检查您已提交的作业（该网站应显示您过去提交的作业及其包含的文件），以确保您提交的作业被正确接收。确保您提交的作业包含正确的文件。