

# The Life of the Game

## Self-Review Questions

---

**Self-review 12.1** Why did we create a class for this application rather than just leave it as a set of functions in a module? Is this actually needed?

**Self-review 12.2** What do the terms *event*, *event handler* and *event loop* mean.

**Self-review 12.3** Which are the event handler/callback methods in the Life class?

**Self-review 12.4** Why have we used multiple threads in this program?

**Self-review 12.5** What is the reason for changing from using a list of list of Booleans as the representation of the state of the universe?

**Self-review 12.6** Why have buttons been used to present the cells of the game universe?

**Self-review 12.7** What have the test programs tested?

**Self-review 12.8** What have the test programs not tested?

**Self-review 12.9** What was the run-time error that had to be avoided in Section ??? Why did it need to be avoided?

**Self-review 12.10** Why is 'separation of concerns' a good approach to program design?

**Self-review 12.11** The Game of Life has many stable patterns. In the text, we mentioned Block, Boat and Blinker. Other stable shapes are *Toad*, *Glider* and *Lightweight Spaceship (LWSS)*. What are these shapes and why are they stable?

*Hint: The answer to the question isn't in this book. You will need to research another sources.*

**Self-review 12.12** What are *observer variables* and how are they used in GUIs?

**Self-review 12.13** What is the Tkinter ‘scale’ widget used for?

**Self-review 12.14** What does the join method (in the Thread class) do?

**Self-review 12.15** We’ve said that the universe which starts off with no life is stable. What would happen if we started the game with every cell being live?

## Programming Exercises

---

**Exercise 12.1** Extend the test program for the Game of Life code to include *Toad*, *Glider* and *LWSS* as part of the test.

**Exercise 12.2** Extend the test program for the Game of Life code to include *Gosper’s Gliding Gun* as part of the test.

**Exercise 12.3** Amend the Game of Life program to use an icon rather than # as the displayed symbol.

**Exercise 12.4** HighLife is a variation on Conway’s Game of Life. Instead of the 23/3 rule (a live cell stays alive only if it has two or three neighbors otherwise it dies, and dead cell is made live if it has exactly three live neighbors), HighLife uses the 23/36 rule (a live cell stays alive only if it has two or three neighbors otherwise it dies, and dead cells become alive if they have either three or six live neighbors). Create a version of the Game of Life program that uses the HighLife rules.

**Exercise 12.5** As presented in this chapter, the user can use the Game of Life GUI to populate the universe randomly. Add a list box which gives the user options to try out the Block, Boat and Blinker societies.

*Hint: Storing these societies as lists in the program should make things straightforward.*

**Exercise 12.6** In the exercise above, you allowed the user to try out some pre-selected societies. Storing the societies as data structures in the code makes it difficult to extend the range of possibilities. For this exercise extend the program so that the societies are stored in files that are read in when the program starts. The list box of possible societies for the user to try should be calculated from the files that were read in and not predefined.

**Exercise 12.7** Add the functionality to the Game of Life program of allowing users to store the *current* game state in a file, and to reload it when they next start the program.

**Exercise 12.8** The Game of Life program in this chapter is an example of a program which can have multiple user interfaces – in this case, one on the console and a GUI. This exercise is to write a program that implement the Caesar Cipher from Section ?? (page ??) but which has multiple user interfaces. The program should have two interfaces to the cipher – one using the console and one which is a GUI.

*Hint: Make sure that the GUI classes, the classes to control console interaction and the actual cipher are all separate. This will mean that you should be able to swap between the two interfaces and add new ones without changing the Caesar Cipher.*

## Challenges

---

**Challenge 12.1** Investigate the technique of using mock objects to help test the user interface.

**Challenge 12.2** Create a class called LifeCell that removes the need for dealing directly with StringVars and button construction (for the grid, anyway). That is, something that can be used to set state (`cella.makeAlive`, or `cella.setState(True)`, perhaps) and expose an associated button for adding to the GUI (`cella.getButton` might be appropriate).

**Challenge 12.3** Exercise 12.4 introduces the HighLife rules and a simple notation for rules for this kind of cellular automata. The standard Game of Life is described in this notation as 23/3, whilst HighLife is described as 23/36. Modify the program so that the rule is set by the user by entering two sequences of numbers into two text boxes.

When your new version of the game is working, enter the rule in the image above. Try it on a single live cell.

**Challenge 12.4** Sudoku is a popular single-player puzzle game played on a 9×9 grid (see the example below). The object of the game is to fill every square with a digit from 1–9 such that each row and column contains exactly one instance of each digit, as does every 3×3 square.

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

For this challenge you should implement a stand-alone GUI-based Sudoku. Sudoku puzzles are quite hard to generate so it might

be sensible to download a set of puzzles and solutions from a free online source rather than making your own in the first instance. Doing random creation of puzzles is a natural evolution of the program. From a usability point of view being able to pause a game and restart it later part solved is a good idea so storing puzzles and solutions in a sensible format that you can easily parse is an important part of the solution.

Wikipedia has a good article on Sudoku at <http://en.wikipedia.org/Sudoku>.