Outline

1. The Game of Life
   - simulating cellular growth
   - cellular automata

2. Representing the Grid
   - lists of lists
   - counting life neighbors
   - a GUI for the Game of Life

MCS 260 Lecture 36
Introduction to Computer Science
Jan Verschelde, 24 July 2023
The Game of Life

- simulating cellular growth
- cellular automata

Representing the Grid

- lists of lists
- counting life neighbors
- a GUI for the Game of Life
The Game of Life
simulating cellular growth

Our application is the “game of life” defined by the mathematician John Horton Conway.

Consider a rectangular grid of squares.
At any square, there can be a living cell or not.

The rules of the game determine whether life or not:

1. An empty cell becomes alive if it has 3 neighbors.

2. A living cell can either die or survive, as follows:
   1. die by loneliness, if the cell has one or no neighbors;
   2. die by overpopulation, if the cell has $\geq 4$ neighbors;
   3. survive, if the cell has two or three neighbors.

Complex patterns may already occur from simple rules.
early in the game
formation of patterns

the game of life

scale controls the delay time

start

stop
patterns at the end
1. The Game of Life
   - simulating cellular growth
   - cellular automata

2. Representing the Grid
   - lists of lists
   - counting life neighbors
   - a GUI for the Game of Life
Cellular Automata

A cellular automaton is a discrete model of cells. Each cell has a state (alive or dead) on a regular grid. Rules define the transition of the states.

Given rules and configurations of cells, one discovers patterns, such as gliders.

A glider in the game of life (* in live cell):

```
  ****
  ****
  *
  ****
  ****
  ***
```

While cellular automata simulate cell growth, they give ways to explore the nature of computation. Stephen Wolfram: *a new kind of science*, 2002.
Python Scripts for the Game

Ingredients in the program:

1. representing the rectangular grid:
   - we use a grid of $r$ rows and $c$ columns,
   - define the rule of an element of the grid.

2. the Graphical User Interface (GUI):
   - determine the layout of the GUI,
   - functionality: draw cells in an animation.
The Game of Life
- simulating cellular growth
- cellular automata

Representing the Grid
- lists of lists
- counting life neighbors
- a GUI for the Game of Life
lists of lists

We can define the grid as a lists of rows:

- every row is a list of boolean values,
- every row has the same number of elements.

Making a list of 5 \texttt{False} values with a list comprehension:

```python
>>> L = [False for _ in range(5)]
```

A double list comprehension to generate a list of 3 rows, every row has 2 elements:

```python
>>> g = [[False for _ in range(2)] for _ in range(3)]
```

```python
>>> for row in g: print(row)
...
[False, False]
[False, False]
[False, False]
```
printing a list of lists

More list comprehensions:

```python
>>> g = [[False for _ in range(2)] for _ in range(3)]
>>> for row in g: print(row)
...
[False, False]
[False, False]
[False, False]
>>> for row in g: print([' %d' % item for item in row])
...
[' 0', ' 0']
[' 0', ' 0']
[' 0', ' 0']
```  

The decimal format of a boolean gives a 0 or a 1.
at the command prompt

$ python game.py
give number of rows : 10
give number of columns : 20

Initial State :

```
1 1 0 0 1 1 0 1 0 0 1 0 1 0 1 1 0 0 1 0
1 0 0 1 1 0 1 0 1 0 1 1 1 1 1 0 1 0 1 1 0
0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 1 0 1 1 1 0 0
0 1 1 1 0 1 0 1 0 0 0 1 0 0 0 1 1 0 1 1 1
0 1 0 0 0 0 0 0 1 0 1 1 0 1 1 0 0 1 0 1
0 1 0 1 0 0 0 0 1 1 1 1 0 1 1 1 1 1 1 1
0 1 0 1 1 1 1 1 1 0 0 0 0 1 0 1 1 1 1 1
1 1 1 0 0 1 0 0 0 0 1 0 1 1 0 1 0 0 1 0
1 1 0 1 0 0 1 1 1 1 0 0 0 0 1 0 0 0 0 1 1
0 1 0 0 0 0 1 0 1 0 1 1 1 0 0 0 0 0 0 1
```

continue ? (y/n)
The function `main()` of `game.py` simulates the game of life as long as the user wants to continue.

In addition to writing the grid, `game.py` defines a module exporting:

1. initialize a random grid `grid`,
2. computing the neighbors of `grid[i][j]`,
3. applying the rule to update the `grid`.

In `life.py`, the GUI for the game of life:

```python
import game
```
The Game of Life
- simulating cellular growth
- cellular automata

Representing the Grid
- lists of lists
- counting life neighbors
- a GUI for the Game of Life
counting life neighbors

\[
\begin{array}{ccc}
[i-1,j-1] & [i-1,j] & [i-1,j+1] \\
[i,j-1] & [i,j] & [i,j+1] \\
[i+1,j-1] & [i+1,j] & [i+1,j+1] \\
\end{array}
\]

\[k = 0\]
\[\text{if } A[i-1][j]: \ k += 1; \ \text{if } A[i-1][j-1]: \ k += 1\]
\[\text{if } A[i+1][j]: \ k += 1; \ \text{if } A[i+1][j-1]: \ k += 1\]
\[\text{if } A[i][j-1]: \ k += 1; \ \text{if } A[i-1][j+1]: \ k += 1\]
\[\text{if } A[i][j+1]: \ k += 1; \ \text{if } A[i+1][j+1]: \ k += 1\]

\[\text{if } 0 < i < \text{len}(A)-1 \ \text{and} \ 0 < j < \text{len}(A[i])-1\]
The Game of Life
- simulating cellular growth
- cellular automata

Representing the Grid
- lists of lists
- counting life neighbors
- a GUI for the Game of Life
Components of the GUI:

1. A scale to change the drawing speed
   The variable defined by the scale is the time the GUI waits to update the canvas.

2. A label to document the scale

3. A canvas to draw the cells
   The canvas has three parameters:
   1. The number of rows in the grid,
   2. The number of columns in the grid,
   3. The size of each square.

   We set the parameters at the start of the GUI.
functionality of the GUI — mapping grid to canvas

Mapping the grid to canvas coordinates:

\[
\begin{array}{ccc|ccc|ccc}
\end{array}
\]

The row index is \( i \) while the column index is \( j \). On canvas: \( y \) is \( i \) and \( x \) is \( j \).

In drawing cells, first all is wiped out from canvas, then for every living cell, a green rectangle is drawn.

As long as \texttt{self.grow} is \texttt{True}:

1. we update the grid of cells
2. wait to refresh the canvas
3. draw the cells and update the canvas
summary of the design

The scripts `game.py` and `life.py` follow an incremental modular design:

- **the module `game.py` defines the rules** and allows for early testing
- **the GUI is in `life.py`, rendering separate from rules of the game**

Limitation: cells are squares, dictated by the grid. It is hard to give cells some individuality, allowing for slight variations of the rules, e.g.: introducing randomness.
Exercises

1. In an object oriented version of the module `game.py`, use the code for the function `random_grid` in the constructor of the grid.

2. In an object oriented version of the module `game.py`, use the code for the function `write` to define the string representation of the grid.

3. In an object oriented version of the module `game.py`, use the code for the functions `neighbors` and `update` to define the transition to the next state of the grid. You can name the method `update()`.

4. Modify the function `neighbors` so that the grid has the topology of a doughnut: the neighbors at the far left edge are at the far right edge and the neighbors at the lower edge are at the upper edge.
Modify the initialization of the game so there is at least one glider.

Add an entry widget to the GUI to display the number of live cells in each stage.

Instead of a boolean list of lists to represent the grid, use a list of lists of integers. A square with no life holds a zero. A living cell is encoded by a positive number equal to the stage in which it was created. Describe the changes that need to be made to the scripts.

Change the drawing of the cells in the GUI so that newborn cells are colored in red. Surviving cells remain displayed in green.