Optional Evening Session

Ken Rice
Tim Thornton

University of Washington

Seattle, July 2015
In this session

- Notes on the Special Exercise
- Some code to get you started

Before we start, please take a few minutes to read the exercise.
In this session

Why are we doing this?

- Practice writing loops – over rows & columns
- Practice breaking a multi-step job into component parts, and doing each of them in turn

This is a simple evolutionary model – the simplest Conway could devise that does anything useful, or interesting. Much of what he learned/proved about it was based on computer simulations, like ours.

It was devised in 1970, and early, error-prone experimentation was done on a Go board.
Conway’s Game of Life: The Rules

Cells live on a grid, they can be alive (1) or dead (0). At each generation they have a number of live neighbors – defined at the 8 surrounding cells.

Cells live, die, and become alive according to these rules;

- If alive==1 and #neighbors < 2, alive ← 0
- If alive==1 and #neighbors == 2 or 3, alive ← 1
- If alive==1 and #neighbors > 3, alive ← 0
- If alive==0 and #neighbors == 3, alive ← 1

– other dead cells stay dead.

(NB nothing is random here – deliberately! – but it’s also straightforward to allow life/death to be somewhat stochastic)
Conway’s Game of Life: The Rules

An example update;
Conway’s Game of Life: The Rules

An example update;
Conway’s Game of Life: The Rules

An example update;
Game of Life: What do we need?

Objects:

- A matrix of cells, each 1 or 0
- A matrix containing ≠ neighbours each cell has
- Another matrix of cells, each 1 or 0 – containing the updated values

Code to do the following jobs:

- Count number of neighbors for cells
- Updating the alive/dead status
- Plot the current status, for all cells
Game of Life: Counting neighbors

Most cells have 8 neighbors...
Game of Life: Counting neighbors

...but some ‘edge cases’ don’t (yuk!)
Game of Life: Counting neighbors

...but some ‘edge cases’ don’t (yuk!)
Game of Life: Counting neighbors

Easier: count on a grid with zeroed-out edges, don’t plot them;
Game of Life: Counting neighbors

Some code to do the counting:

nrows <- 7
cols <- 7
alive <- matrix(0, nrows+2, ncols+2) # "+2" is adding the gray border

# add some "alive" cells
alive[4,4:6] <- 1
alive[7:8,7] <- 1

# do the neighbour counting - only for the non-gray cells
neebs <- matrix(0, nrows+2, ncols+2)
for(i in 2:(nrows+1)){
    for(j in 2:(ncols+1)){
        neebs[i,j] <- alive[i-1,j-1] +
        alive[i-1,j ] +
        alive[i-1,j+1] +
        alive[i ,j-1] +
        alive[i ,j+1] +
        alive[i+1,j-1] +
        alive[i+1,j ] +
        alive[i+1,j+1] # adding over the 8 neighbors
    } # close j loop
} # close i loop
Game of Life: Plotting status

There are many ways to plot the cells – rect() offers one simple way; if $i$ indexes rows and $j$ columns, we need e.g.

\[
\begin{align*}
\text{xleft} & \quad j - 1/2 \\
\text{ybottom} & \quad i - 1/2 \\
\text{xright} & \quad j + 1/2 \\
\text{ytop} & \quad i - 1/2
\end{align*}
\]

... and also specify color – e.g. 1 for black/dead, 2 for red/alive.

Recall Sessions 3/4; first set up an empty plot (type="n") ...

```r
plot(0,0, type="n", xlab="", ylab="", axes=F,
     xlim=c(0.5,nrows+0.5), ylim=c(0.5,ncols+0.5), asp=1)
```

... then add the cell entries – with another double loop.

```r
for(i in 1:nrows){
    for(j in 1:ncols){
        rect(j-0.5,i-0.5,j+0.5,i+0.5,
             col=alive[i+1,j+1] + 1, border="cyan")
    }
}
```
Game of Life: Updating status

How to update? (recall the grey border trick, again)

alive.new <- matrix(0, nrows+2, ncols+2)  # note full of zeros
for(i in 2:(nrows+1)){
  for(j in 2:(ncols+1)){
    if(alive[i,j]==1 & neebs[i,j]<2){ alive.new[i,j] <- 0 }
    if(alive[i,j]==1 & neebs[i,j]%in%2:3){ alive.new[i,j] <- 1 }
    if(alive[i,j]==1 & neebs[i,j]>3){ alive.new[i,j] <- 0 }
    if(alive[i,j]==0 & neebs[i,j]==3){ alive.new[i,j] <- 1 }
  }
}
alive <- alive.new

Note: the other alive==0 cells stay dead, so there’s no need for another if() statement here
Some code to check your counting;

```r
for(i in 1:nrows){
    for(j in 1:ncols){
        text(j,i, neebs[i+1,j+1], col="white") }}
```

Why `text(j,i, ...)`? Note that `text()` takes x and y co-ordinates, which correspond to index j and i respectively – as with plotting status.
Some pseudo-code; fill in the rest yourself – cut-and-pasting the parts from earlier slides.

\[\text{nrows} \leftarrow 7\]
\[\text{ncols} \leftarrow 7\]
\[\text{alive} \leftarrow \# \ldots\text{some initial state}\]
\[\text{plot}(0,0 \# \ldots\text{set up the plot}\]
\[\quad \# \ldots\text{plot the initial state}\]

\[\text{for } k \text{ in (1:100)}{\{\]
\[\quad \# \text{ count neighbors (a double loop)}\]
\[\quad \# \text{ update status – who lives/dies? (a double loop)}\]
\[\quad \text{alive} \leftarrow \text{alive.new}\]
\[\quad \# \text{ plot again (a double loop)}\]
\[\}\]

- Then... sit back and be mesmerized!
- Start with random entries, and try a (much) bigger grid
The End (for now)

Notes;

• The coding here is designed to be easy to read, not to be optimally fast – slow code that works is better than fast code that doesn’t!

• In Session 10 we’ll review some ways to speed up the code (and still have it work)

• ... and ways to have the grid ‘wrap around’

• ... also ways to make animations