10. The End

Ken Rice
Tim Thornton

University of Washington

Seattle, July 2013

In this session

- Notes on the Special Exercise
- What next?

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

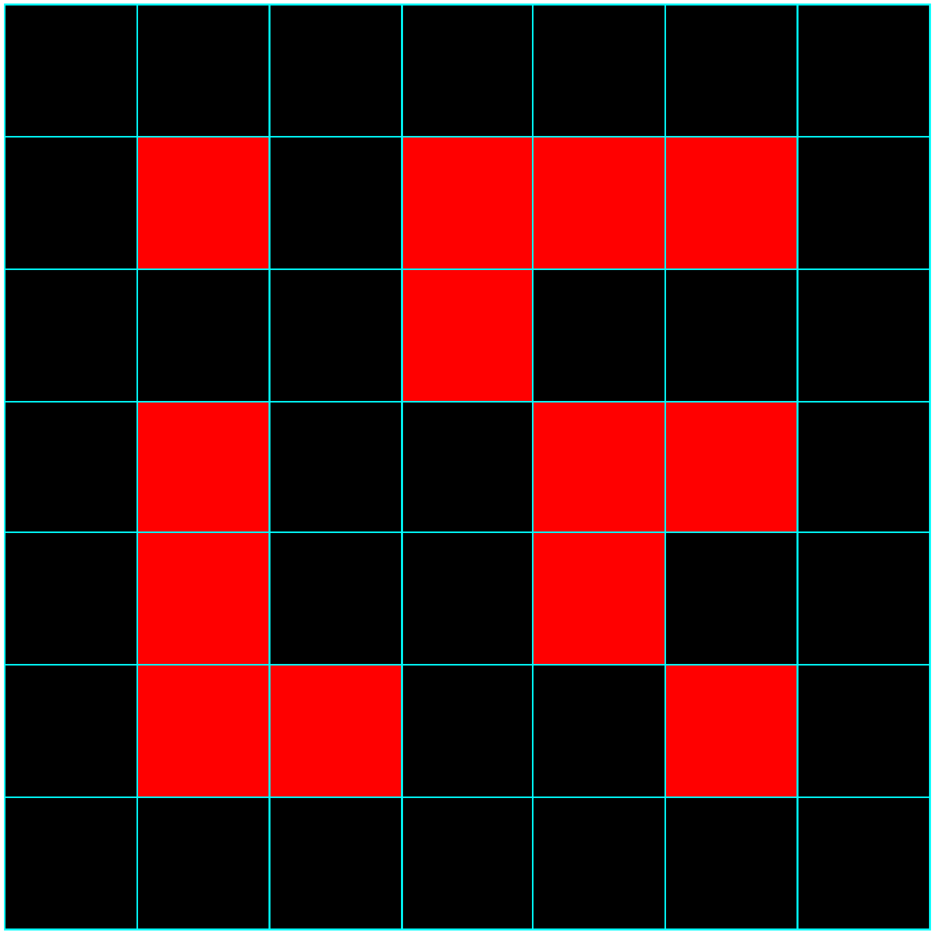
If alive==1 and #neighbors <2,	alive <− 0
If alive==1 and #neighbors ==2 or 3,	alive <− 1
If alive==1 and #neighbors >3,	alive <− 0
If alive==0 and #neighbors ==3,	alive <− 1

– other dead cells stay dead.

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

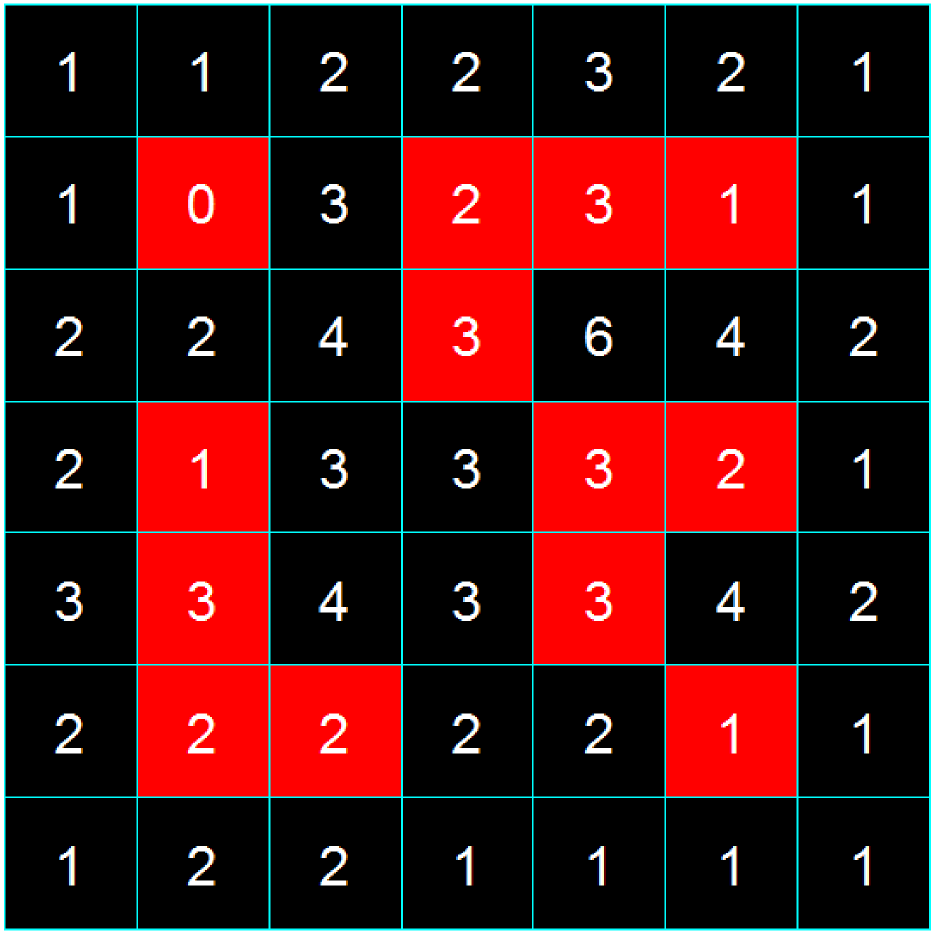
Game of Life: The Rules

An example update;



Game of Life: The Rules

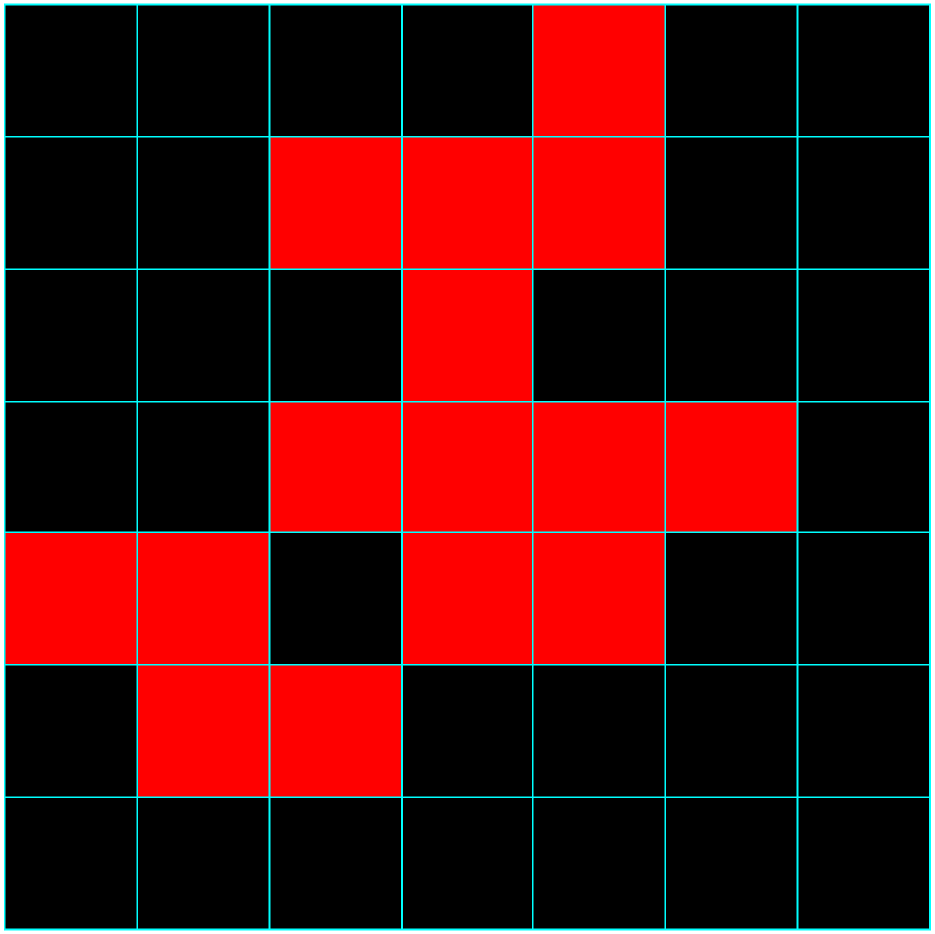
An example update;



1	1	2	2	3	2	1
1	🟥 0	3	🟥 2	🟥 3	🟥 1	1
2	2	4	🟥 3	6	4	2
2	🟥 1	3	3	🟥 3	🟥 2	1
3	🟥 3	4	3	🟥 3	4	2
2	🟥 2	🟥 2	2	2	🟥 1	1
1	2	2	1	1	1	1

Game of Life: The Rules

An example update;



Game of Life: What do we need?

Objects;

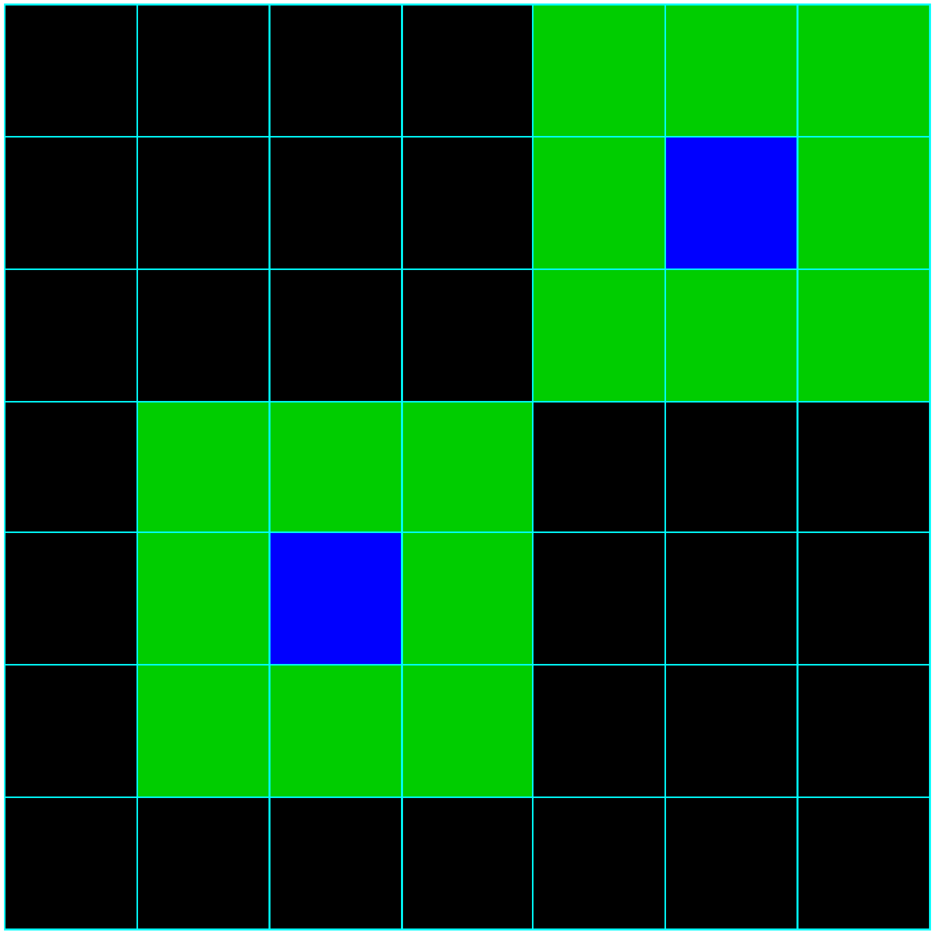
- A `matrix` of cells, each 1 or 0
- A `matrix` containing # neighbours each cell has
- Another `matrix` of cells, each 1 or 0 – containing the updated values

Code to do the following jobs;

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

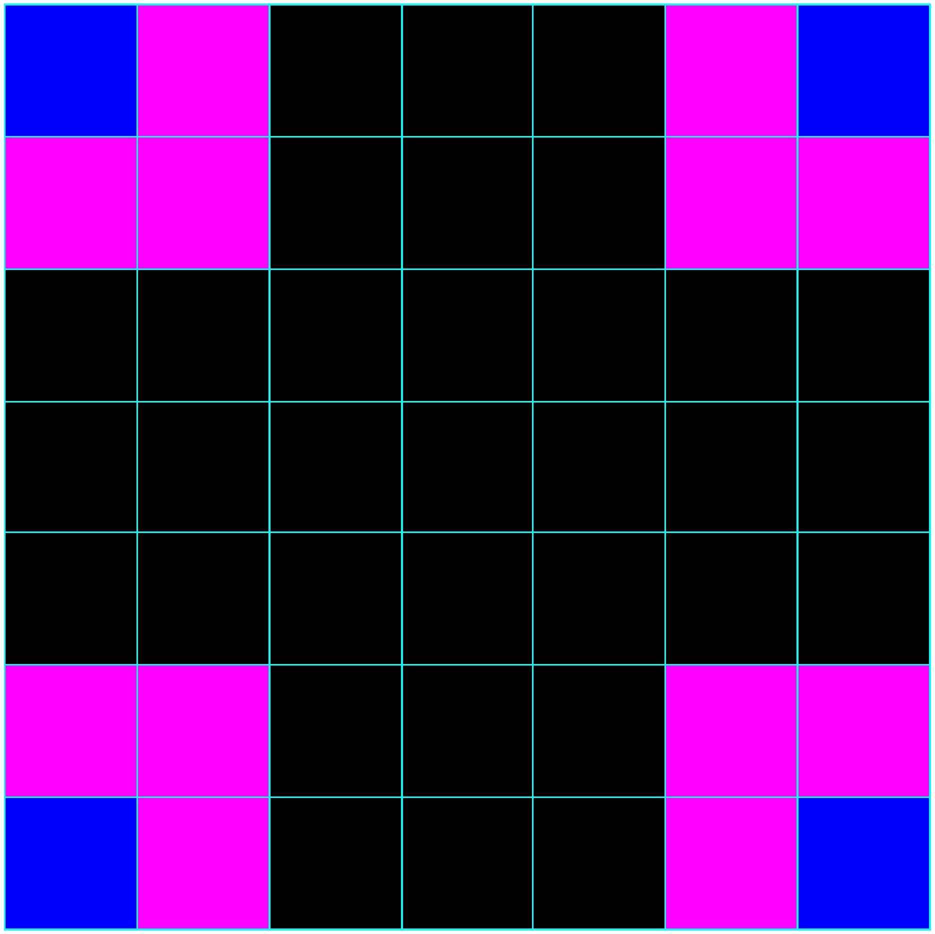
Game of Life: Counting neighbors

First option for neighbors; count based on what's visible.



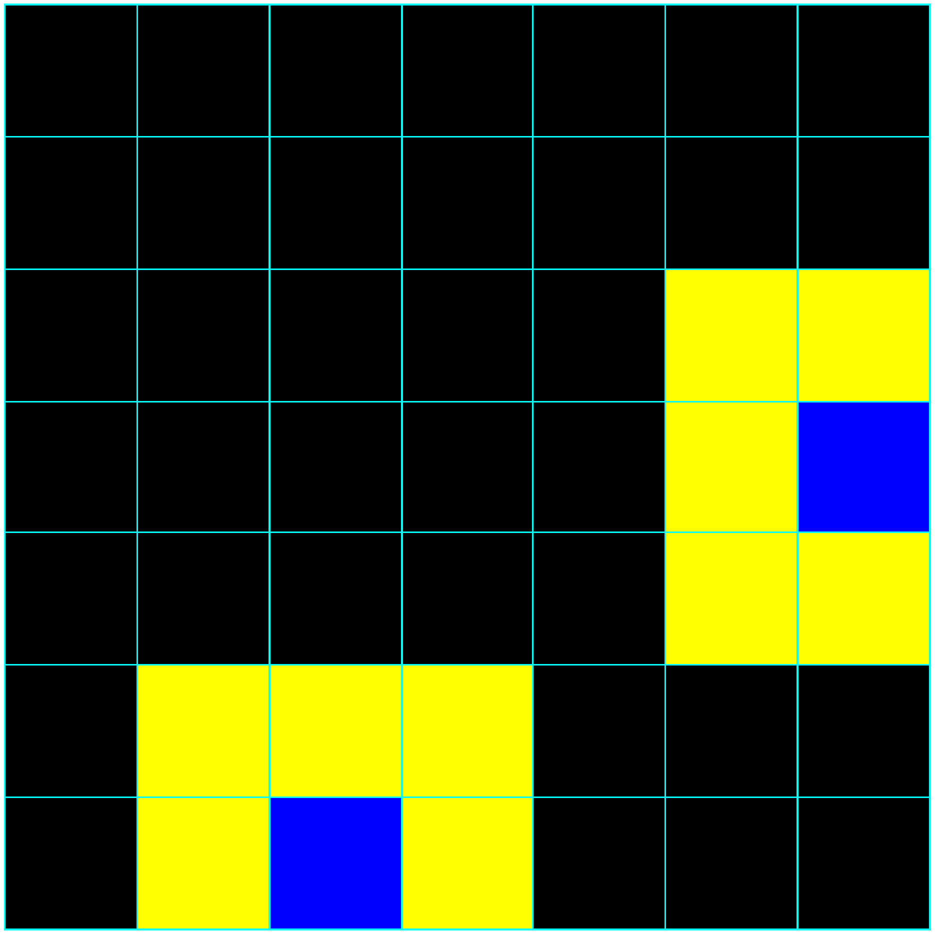
Game of Life: Counting neighbors

First option for neighbors; count based on what’s visible.



Game of Life: Counting neighbors

First option for neighbors; count based on what's visible.



Game of Life: Counting neighbors

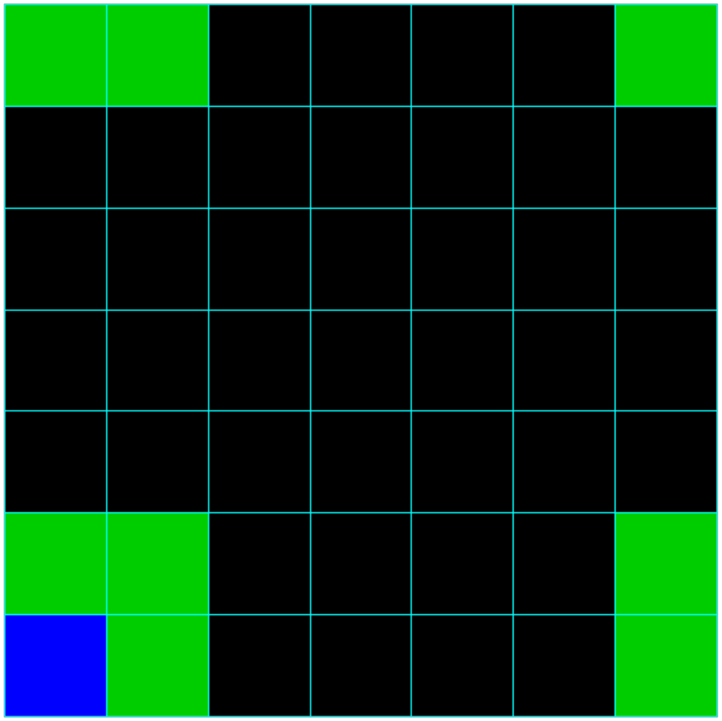
Pseudo-code for counting neighbors;

```
neebs <- matrix(NA, nrows, ncols)
for(i in 1:nrows){
  for(j in 1:ncols){
    if(<in corner>){ add up over 3 relevant cells }
    if(<on side>){   add up over 5 relevant cells }
    if(<other>){     add up over 8 relevant cells }
    neebs[i,j] <- #neighbors we just counted
  }
}
```

- `neebs` stores the number of neighbors
- Which cells are relevant depends on i and j

Game of Life: Counting neighbors

Doing the ‘wrap-around’ version, always count 8 neighbors;



Same basic commands, ‘wrap’ with modular arithmetic;

```
> 1:13 %% 7
 [1] 1 2 3 4 5 6 0 1 2 3 4 5 6
> ((1:13 - 1) %% 7) + 1
 [1] 1 2 3 4 5 6 7 1 2 3 4 5 6
```

Game of Life: Updating status

Not-so-pseudo code;

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

- Other `alive==0` cells stay dead, so no need for another line
- Possible to do fewer updates, if start with
  `alive.new <- alive`

Game of Life: Plotting status

There are many ways to do this. `rect()` offers one simple way; if i indexes rows and j columns, we need e.g.

`xleft`	$j - 1/2$
`ybottom`	$i - 1/2$
`xright`	$j + 1/2$
`ytop`	$i - 1/2$

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

```
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,j] + 1, border="cyan")
   }
}
```

Also need to set up an ‘empty’ plot first; `type="n"`

Game of Life: Putting it together

Other notes;

- Do each task separately, on a small grid (i.e. a small dataset) and make sure it works right
- As per session 9, it helps to write a function for each task

Final pseudo-code;

```
alive <- <initial setup>
<setup empty plot>
refresh.grid(alive) # plot the initial status

for(k in 1:n.steps){
    alive <- do.update(alive) # counting and updating
    refresh.grid(alive)       # plotting
    }
```

Game of Life: Some cute tricks

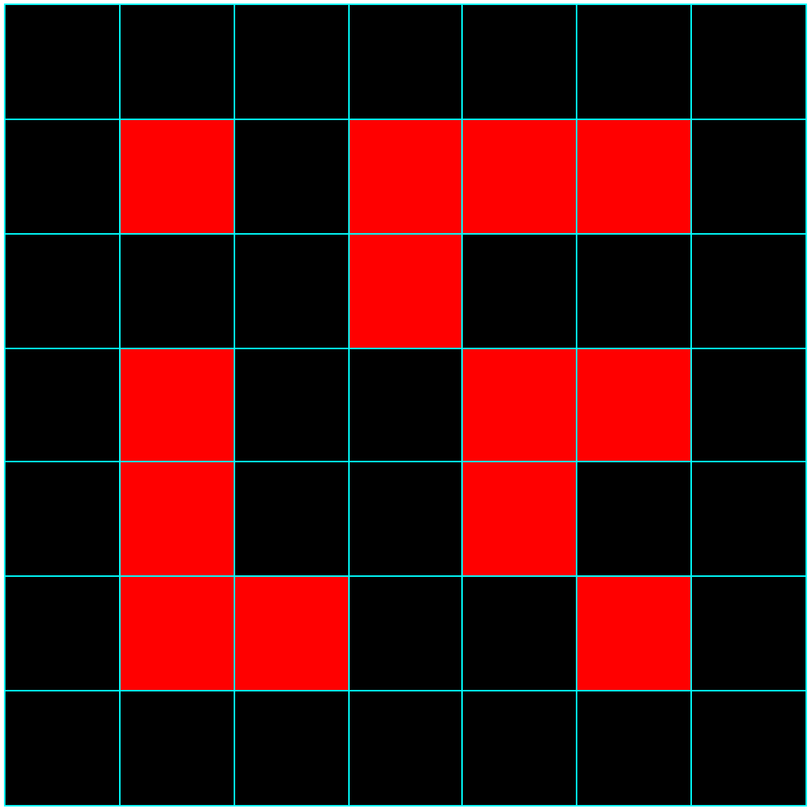
If the arguments to `rect()` are vectors, it draws multiple rectangles. To do this, we need to make a copy of `alive` in ‘long’ format, i.e. in a long tall matrix, where each row corresponds to a cell

```
a.long <- cbind( alive=as.vector(alive),
                 expand.grid(row=1:side, clm=1:side) )
# this is a side^2 x 3 matrix
rect(a.long$clm-0.5,
     a.long$row-0.5,
     a.long$clm+0.5,
     a.long$row+0.5, col=a.long$alive+1, border="cyan")
```

This is slightly easier to type, but doesn't actually speed things up much.

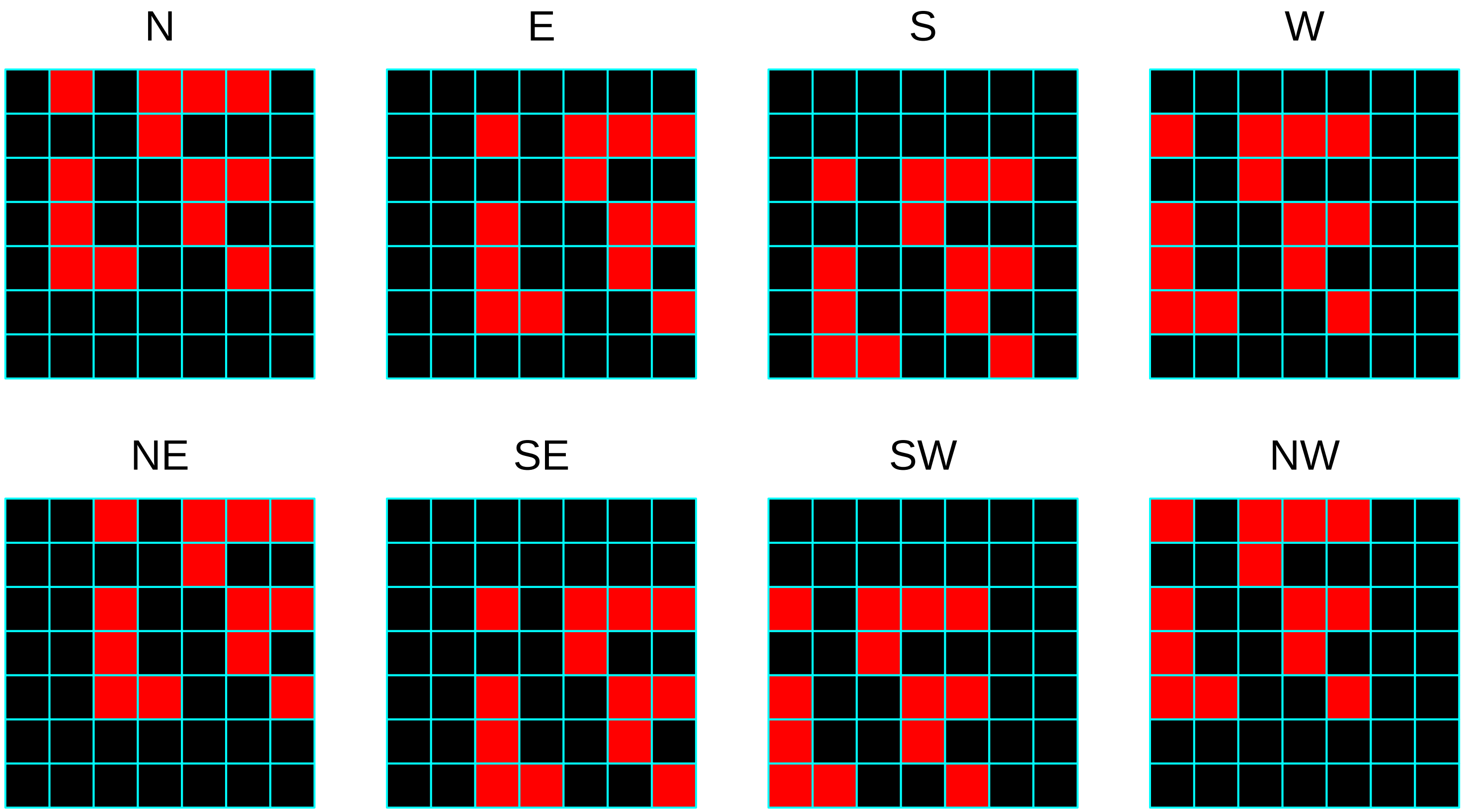
Game of Life: Some cute tricks

A much cuter trick; to count neighbors, slide the grid in all 8 directions, and add;



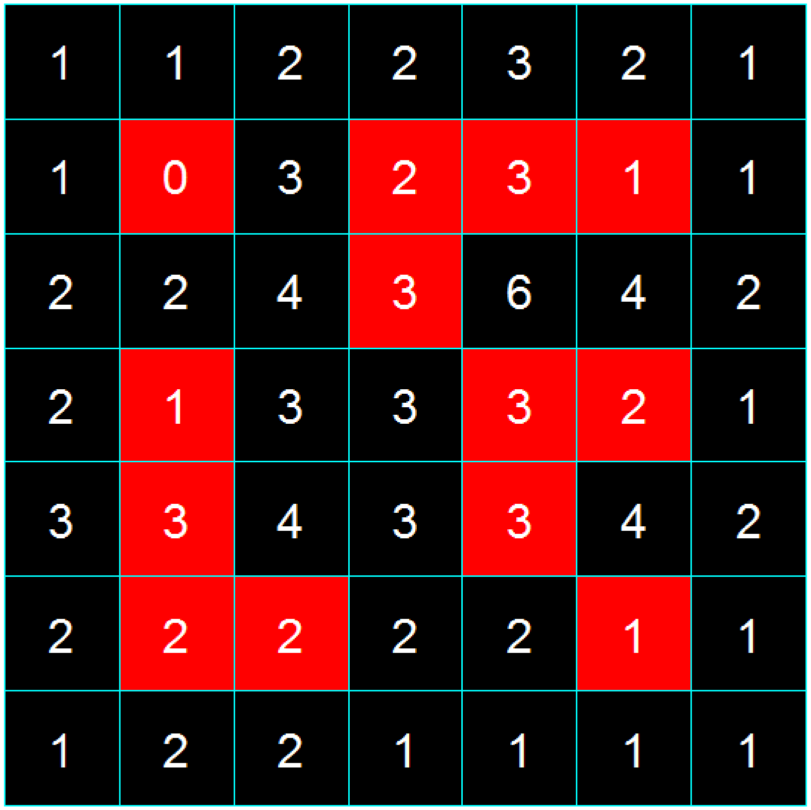
Game of Life: Some cute tricks

A much cuter trick; to count neighbors, slide the grid in all 8 directions, and add;



Game of Life: Some cute tricks

A much cuter trick; to count neighbors, slide the grid in all 8 directions, and add; (it works!)



1	1	2	2	3	2	1
1	0	3	2	3	1	1
2	2	4	3	6	4	2
2	1	3	3	3	2	1
3	3	4	3	3	4	2
2	2	2	2	2	1	1
1	2	2	1	1	1	1

Game of Life: Some cute tricks

This enables counting neighbors without any explicit loops;

```
  al.E <- alive[,c(ncols, 1:(ncols-1))] # moved E
  al.W <- alive[,c(2:nrows, 1)] # moved W
  al.N <- alive[c(nrows, 1:(nrows-1)),]
  al.S <- alive[c(2:nrows, 1),]
  al.SW <- rbind( alive[2:(nrows), c(2:nrows,1)],
                  alive[1,         c(2:ncols,1)] )
  <etc>
  neebs <- al.W + al.NW + al.N + al.NE +
           al.E + al.SE + al.S + al.SW
```

This does notably speed up the code – faster than the graphics can cope, on my laptop. Much of the processor time is spent managing the `for()` loop, and this ‘vectorized’ version means all that work is done in C/Fortran instead.

Game of Life: Some cute tricks

Our final trick is ‘logical subscripting’. We can reassign elements of a matrix indexed by the TRUE elements of another matrix, without losing the original matrix structure.

```
  alive.new <- alive
  alive.new[alive==0 & neebs==3] <- 1
  alive.new[alive==1 & neebs<2] <- 0
  alive.new[alive==1 & neebs>3] <- 0
  alive <- alive.new
```

A simpler example – to show this really is a trick;

```
> x <- matrix(1:10, 2, 5)
> x
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    3    5    7    9
[2,]    2    4    6    8   10
> y <- x>5
> x[y]
[1]  6  7  8  9 10
> x[y] <- 0
> x
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    3    5    0    0
[2,]    2    4    0    0    0
```

Game of Life: Nicer output

Standard R graphics are not built for animations. If your code goes too fast for them, export to a file format that does permit animations – e.g. animated GIFs, which the `caTools` package will write.

Storing every version of `alive` in a nrows × ncols × nsteps `array` object.

```
all.alive <- array(NA, c(nrows, ncols, nsteps))
alive <- <initial setup>

for(k in 1:n.steps){
    all.alive[,,k] <- alive   # store current status
    alive <- do.update(alive) # counting and updating
    }
install.packages("caTools")
library("caTools")
write.gif(image=all.alive, filename="gol.gif", scale="always",
          col="jet")
```

Game of Life: Nicer output

R can't display animated GIFs. So, to open this file in the default application on your computer;

```
shell.exec("gol.gif")
```

Assuming your machine knows what to do with URLs, also try

```
shell.exec("http://www.google.com/")
```

And having done that, try this last `mammals` example;

```
mammals <- read.table("mammals.txt", header=TRUE)
plot(log(brain)~log(body), data=mammals) # usual plot

repeat({
   mychoice <- identify(y=log(mammals$brain), x=log(mammals$body),
                        labels=row.names(mammals), n=1)
   shell.exec(
      paste("http://images.google.com/images?q=",
            row.names(mammals)[mychoice], sep=""))
})
```

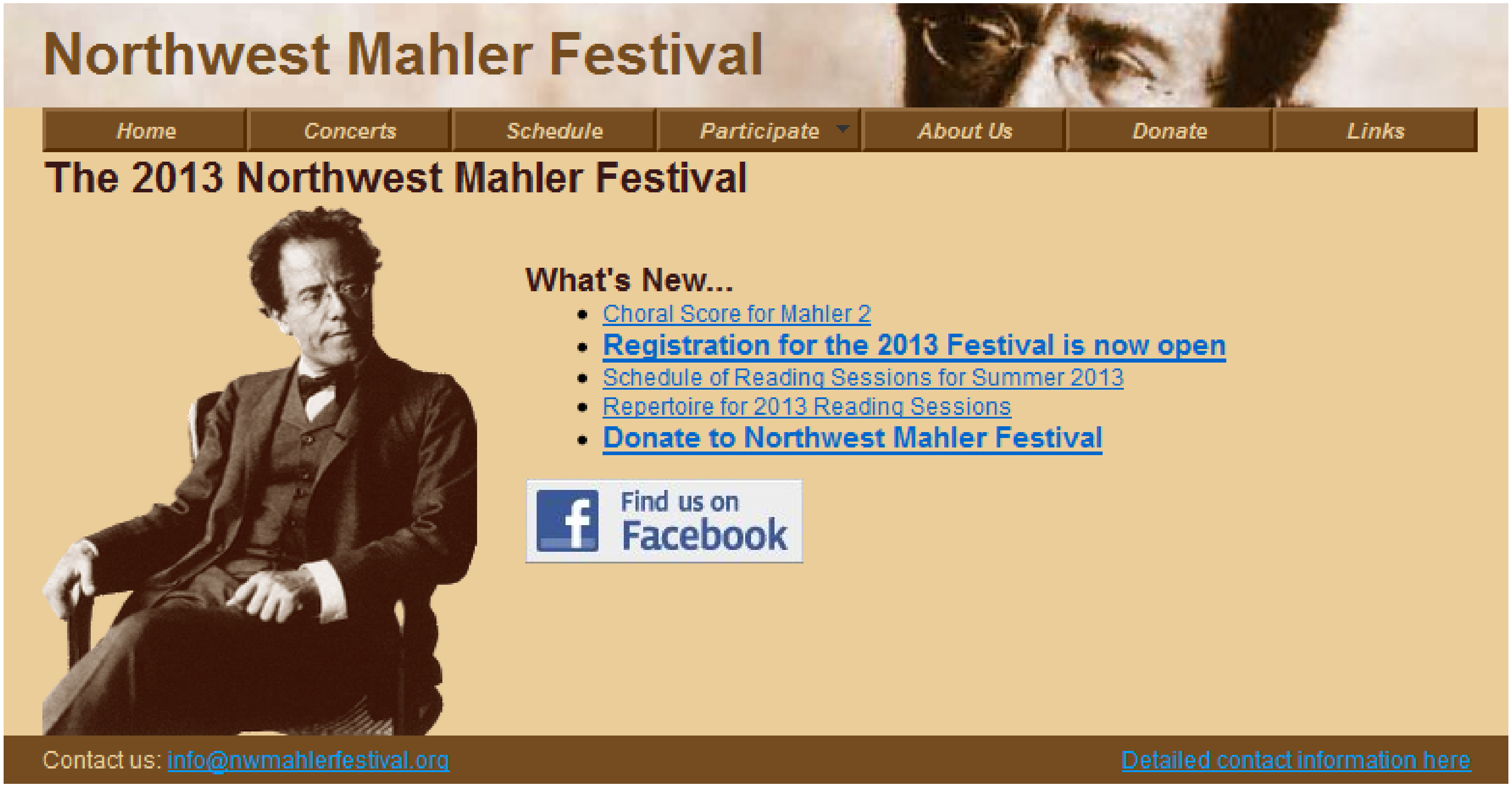
What next?

This concludes our course. To learn more;

- Take the next one! ‘Elements of R’ follows on, with genetics/bioinformatics examples (and lots of programming)
- See the recommended books, on the course site
- To find simple examples/functions, ask Google (in a web browser
- There are several R mailing lists; R-help is the main one. *But* contributors expect you to have read the documentation – all of it! CrossValidated is friendlier to beginners
- Emailing package authors may also work
- For questions about *any* software, say;
  - What you did (ideally, with an example)
  - What you expected it to do
  - What it did instead

What next week?

Calling anyone who can sing (or just read music) – no auditions for two ‘open reading’ sessions.



Northwest Mahler Festival

The 2013 Northwest Mahler Festival

What's New...

- Choral Score for Mahler 2
- Registration for the 2013 Festival is now open
- Schedule of Reading Sessions for Summer 2013
- Repertoire for 2013 Reading Sessions
- Donate to Northwest Mahler Festival

Contact us: info@nwmahlerfestival.org

Detailed contact information here

www.nwmahlerfestival.org