10. The End

Ken Rice
Tim Thornton

University of Washington

Seattle, July 2014

In this session

- Notes on the Special Exercise
- Cool interactivity (by example)
- What next?

Game of Life: the rules

As you will recall...

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.

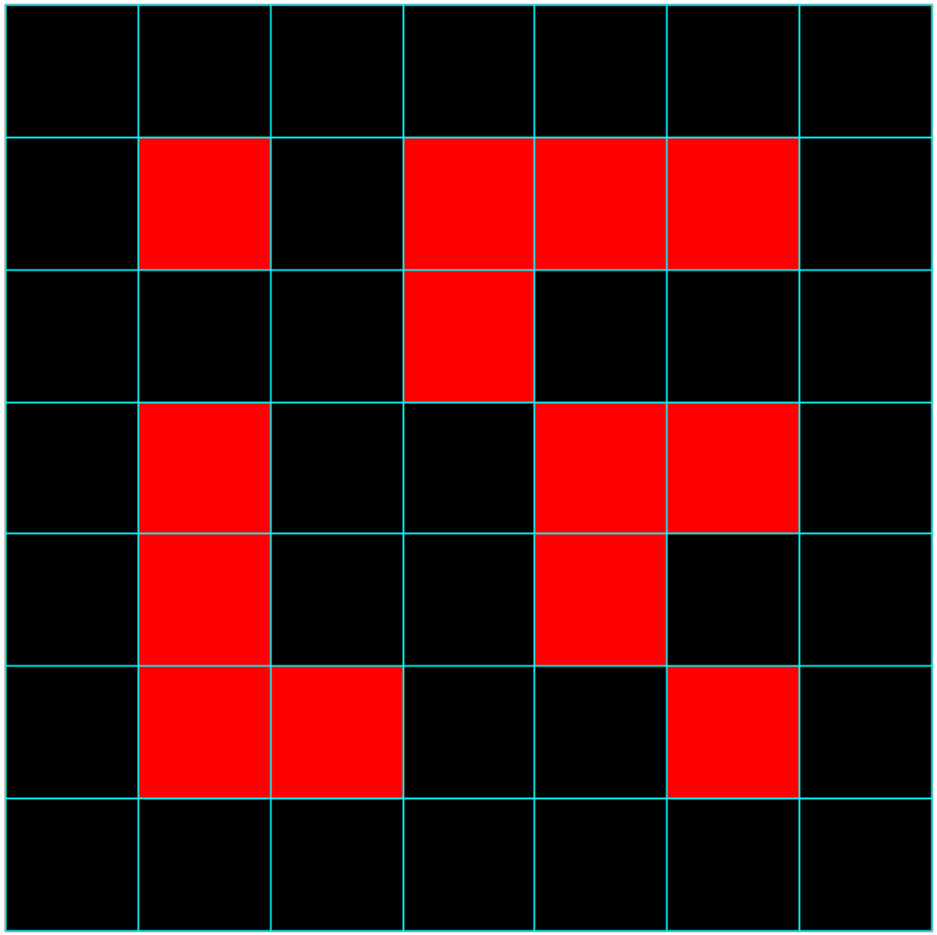
Cells live, die, and become alive according to these rules;

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.

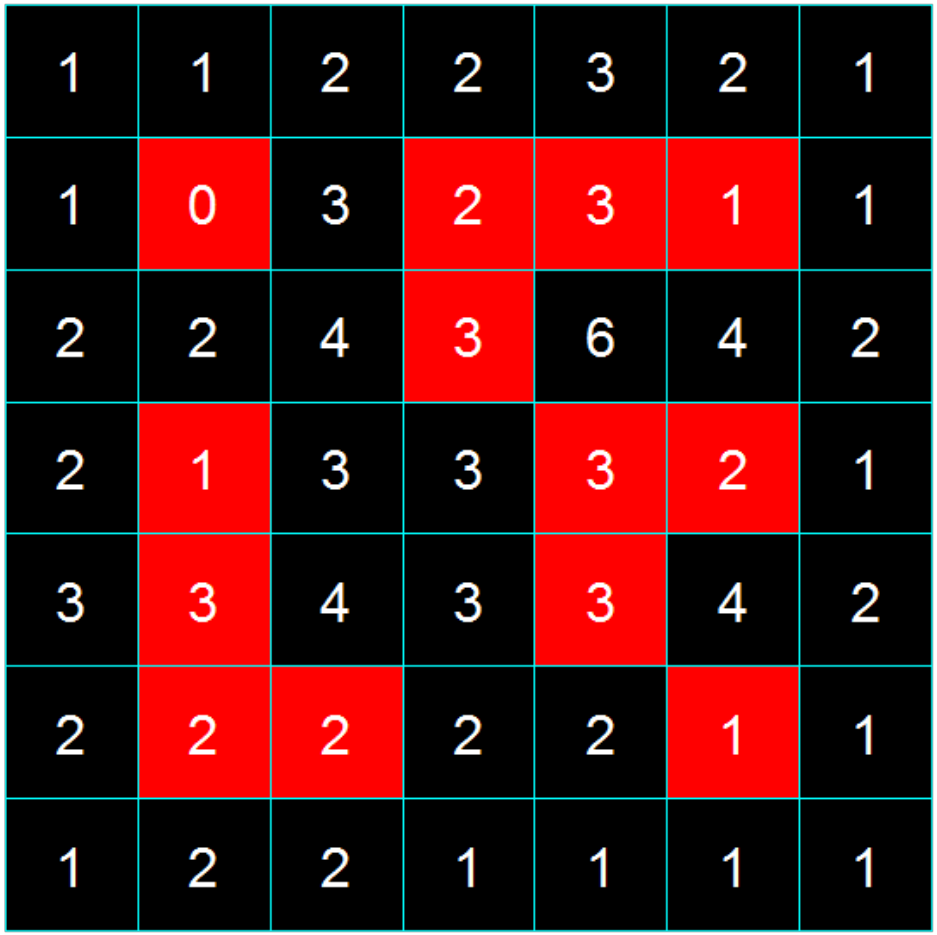
Game of Life: the rules

An example update;



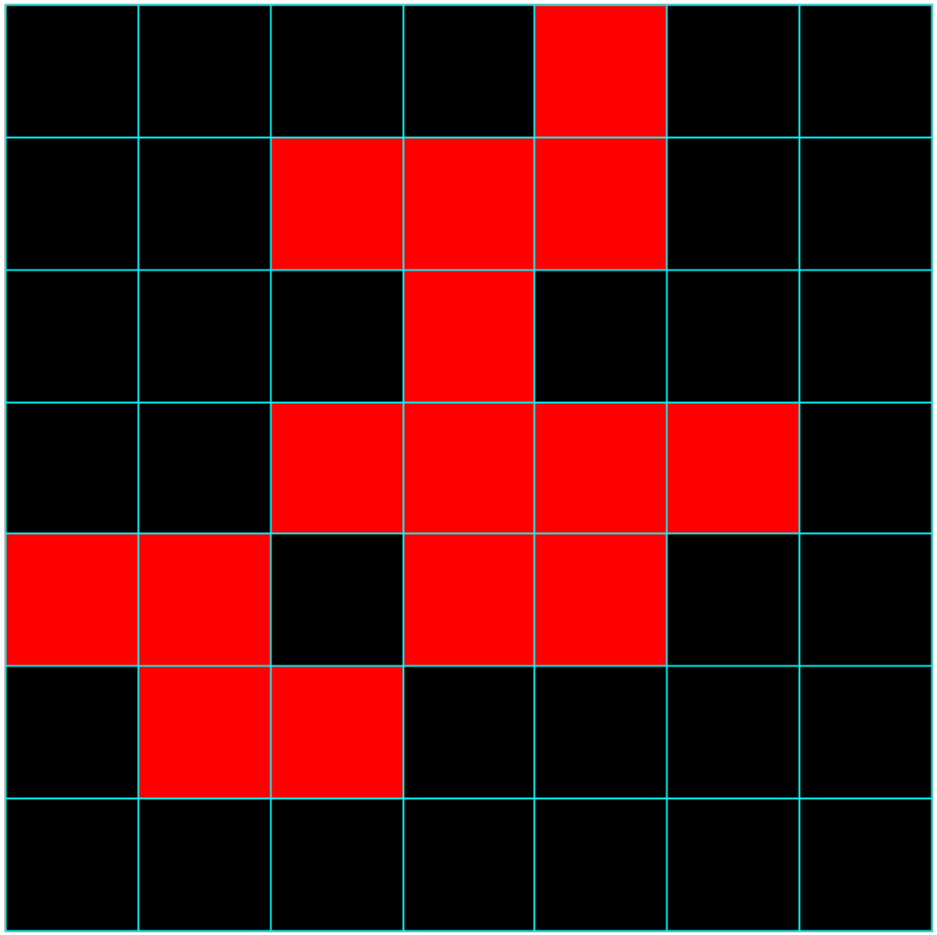
Game of Life: the rules

An example update;



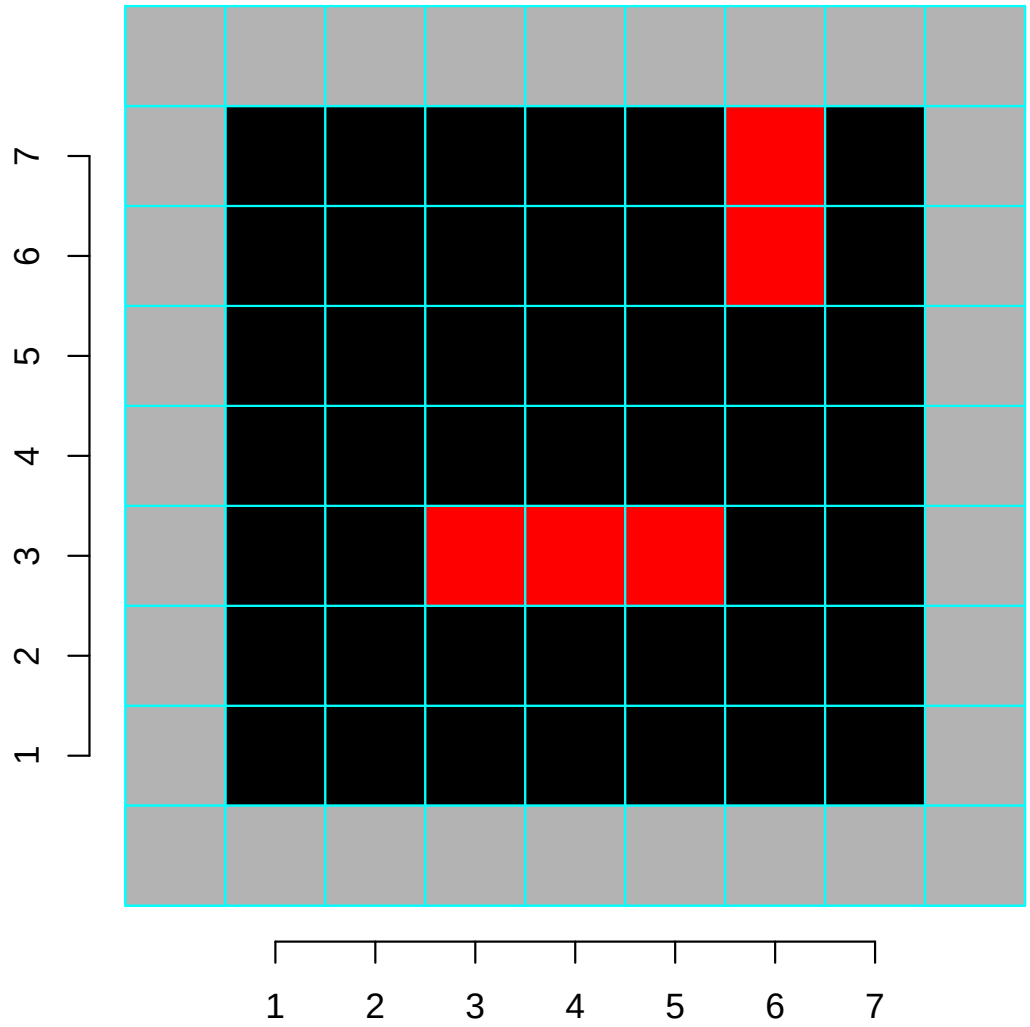
Game of Life: the rules

An example update;



Game of Life: the rules

And a trick to make ‘edge-cases’ easier;



Game of Life: with functions

We first need to set up a matrix of 1s/0s, to indicate alive/dead in a specified number of rows and columns.

Then, within the loop (over many generations) there are 3 major jobs to do;

- Plot the current 'alive' status
- Count the neighbours
- Update the 'alive' status

... we will write functions to do each one – as per Session 9!

Game of Life: plot current status

```
# first, just set up some empty axes;
do.basic.plot <- function(nrows, ncols){
   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)
  invisible() # no output goes to command line
}

update.plot <- function(alive, nrows=dim(alive)[1]-2,
                            ncols=dim(alive)[2]-2){
   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]*6 + 1, border="blue")
      }   # NB cols here are 0/1*6 = 1 or 7 - black or yellow
   }
  invisible()
}
```

Game of Life: count neighbours

```
get.neebs <- function(alive, nrows=dim(alive)[1]-2,
                      ncols=dim(alive)[2]-2){
   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
   neebs   # return the matrix of counts
}
```

Game of Life: update status

```
update.alive <- function(alive, neebs, nrows=dim(alive)[1]-2,
                         ncols=dim(alive)[2]-2){
   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.new # return the new status
}
```

Game of Life: get on with it!

First some set up: here with a random starting position;

```
nrows <- 40
ncols <- 40
n.updates <- 100
set.seed(4)
alive <- matrix(rbinom((nrows+2)*(ncols*2),1, 0.3), nrows+2,
                ncols+2) # "+2" is adding the gray border
```

And actually doing the work; (this is 'high level' code)

```
do.basic.plot(nrows, ncols) # sets up axes
update.plot(alive)          # plots initial status
for(k in 1:n.updates){
  neebs <- get.neebs(alive)             # count neighbors
  alive <- update.alive(alive, neebs)   # update status
  update.plot(alive)                    # plot new status
}
```

Game of Life: get on with it!

Some suggested extras:

- Add a counter, showing index `k` increasing;
  `legend("bottomright", bg="white", pch=NA, legend=k, cex=0.7)`

- Wait before continuing to next iteration;
  `cat("Press [enter] to continue")`
  `line <- readline()`

- Store the 'lifespan' of each cell, e.g. 0/1/2/3/4/5+, and show this with color coding – this is more work

Speed-ups *are* possible, but they require avoiding use of `for()` loops. (Details available on request... or come back and take a later module!)

Game of Life: not yet rated?

To show off your new-found prowess in R, you'll want a file for your website. The `saveGIF()` function in the `animation` package makes GIFs where each 'still' is an R plot;

```
install.packages("animation")
# NB this requires ImageMagick, http://www.imagemagick.org
# ... and won't work without it
library("animation")
nrows <- 40 # usual setup
ncols <- 40
n.updates <- 100
set.seed(4)
alive <- matrix(rbinom((nrows+2)*(ncols*2),1, 0.3), nrows+2, ncols+2)
saveGIF(expr={                          # 'expr' is the earlier high level code
   do.basic.plot(nrows, ncols)
   update.plot(alive)
   for(k in 1:n.updates){
      neebs <- get.neebs(alive)
      alive <- update.alive(alive, neebs)
      do.basic.plot(nrows, ncols)
      update.plot(alive)}
}, movie.name = "conway.gif", interval=0.1)
```

Shiny

It's also possible to display data analyses on websites – and have them be interactive. The `shiny` package, by RStudio, builds 'apps' that do this.

The syntax is (roughly) a hybrid of R and HTML, so we give just a short example, showing off the `salary` data again*.

To make an app, in a directory named for your app, you need two files;

- **ui.R** This R script controls the layout and appearance of your app
- **server.R** This script contains the instructions that your computer needs to build your app

NB `shiny` is temperamental about which version of R you use.

*The online tutorial is excellent

Shiny: ui.R

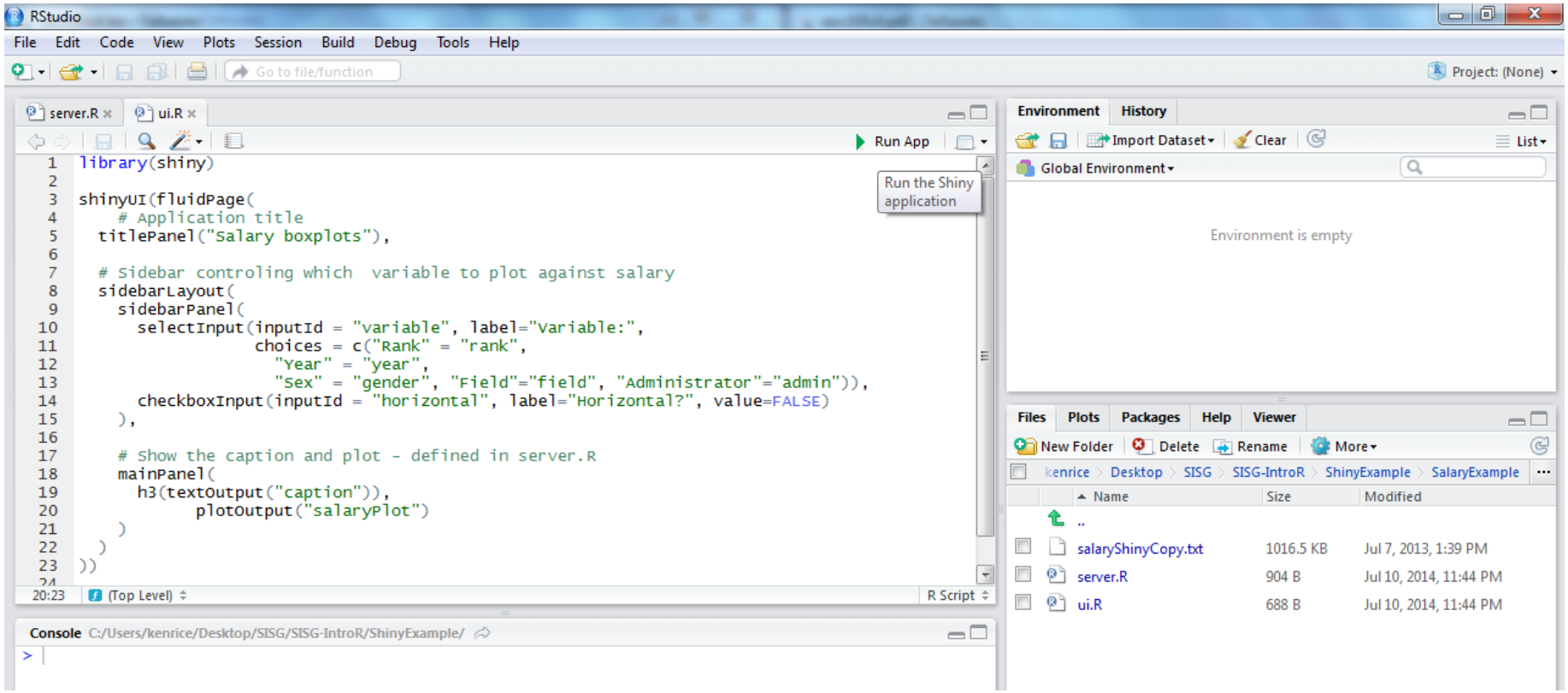
```
library("shiny") # after installing it
shinyUI(fluidPage(
    # Application title
  titlePanel("Salary boxplots"),

  # Sidebar controlling which  variable to plot against salary
  sidebarLayout(
    sidebarPanel(
      selectInput(inputId = "variable", label="Variable:",
                  choices = c("Rank" = "rank", "Year" = "year",
                              "Sex" = "gender", "Field"="field",
                              "Administrator"="admin")
                  ),
      checkboxInput(inputId = "horizontal", label="Horizontal?", value=FALSE)
      ),
    # Show the caption and plot - defined in server.R
    mainPanel(
      h3(textOutput("caption")),
            plotOutput("salaryPlot")
    ) # close main Panel
  )   # close sidebarLayout
))
```

Shiny: server.R

```
library("shiny")
# first, a local copy of salary data sits in same directory
salary <- read.table("salaryShinyCopy.txt", header=T)

# make some variable factors - for prettiness
salary$year  <- factor(salary$year)
salary$admin <- factor(salary$admin)

# Define server "logic" required to plot salary vs various variables
shinyServer(function(input, output) {

  # Compute the forumla text in a "reactive expression"
  # it is shared by  output$caption and output$mpgPlot, below
  formulaText <- reactive({ paste("salary ~", input$variable) })

  # Return the formula text for printing as a caption
  output$caption <- renderText({ formulaText() })

  # Do the boxplot, using the formula syntax, and setting horizontal=T/F
  output$salaryPlot <- renderPlot({
    boxplot(as.formula(formulaText()),
            data = salary, horizontal = input$horizontal) })
}) # close function
```

Shiny: making it work in Rstudio

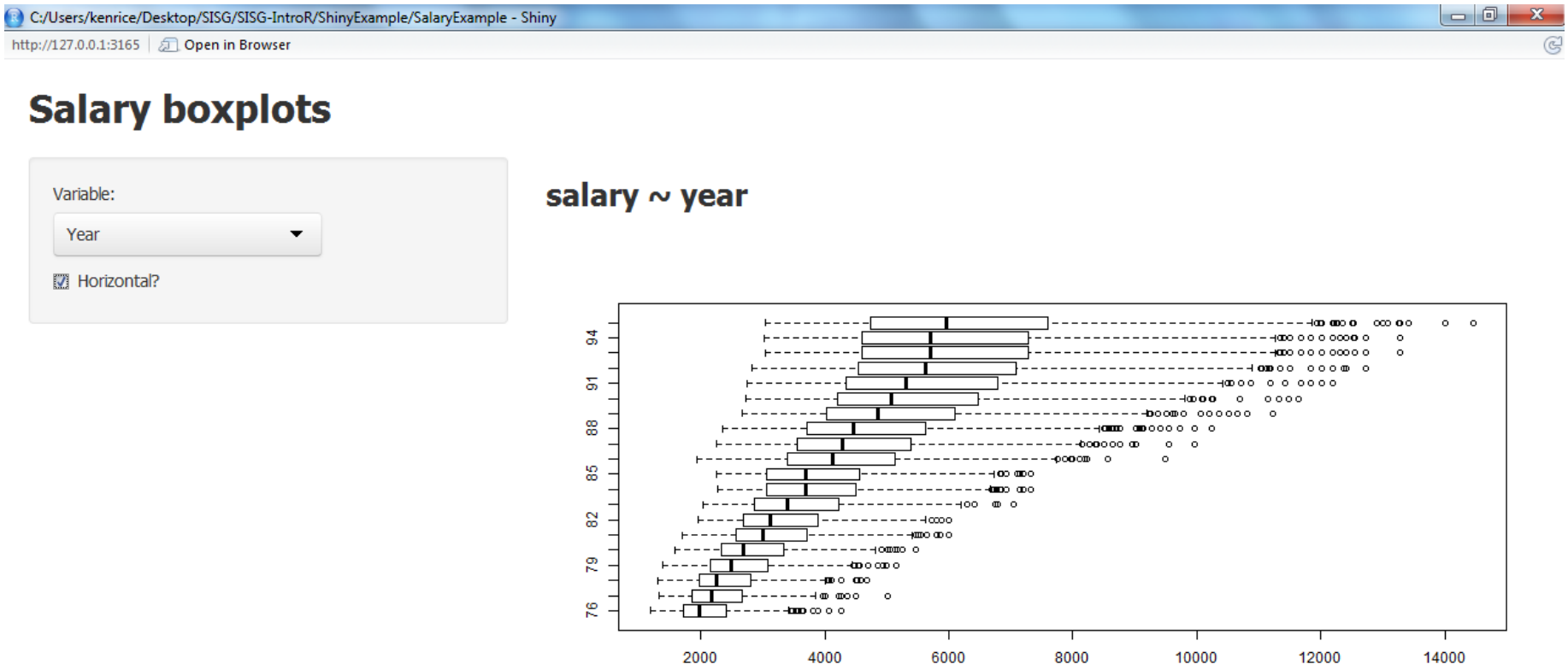
This is remarkably straightforward;



- Hit 'Run App' – and it (should) run
- Note that `ui.R`, `server.R` and the `salaryShinyCopy.txt` data file are *all* in the `SalaryExample` directory

Shiny: making it work in Rstudio

The (interactive) output should look something like this;



- Expect mild differences, across systems
- To share your app online, go to http://my.shinyapps.io/ – needs registration, and (unavailable!) `shinyapps` package
- Be careful with personal data!

Shiny: making it work in Rstudio

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

```
shell.exec("conway.gif")
shell.exec("notepad") # opens the most basic text editor
```

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