title: “Assignment#5” author: “Nora Cook” date: “April 25, 2017” output: html_document —

These are the parameters of the game. We will play the game for 50 generations the grid is 12 rows by 12 columns. So there are 144 cells.

life3<-function(grid.size=12,start=50,generations=50,slow=FALSE) {

require(reshape2) ###Check for and load the reshape2 package

par(mar=c(1, 1, 4, 1)) ###Set plot margins to small

Set up initial population

Now we set up the initial population. The initial seed parameter rep.int(1,times=start) shows when a cell lives

ep.int(0,times=(grid.size^2)-start)) for dead. This is how the random sampler picks the values of the cell for the grid size^2 cells.

init<-sample(c(rep.int(1,times=start),rep.int(0,times=(grid.size$^{2)-start)),size=grid.size}$2,replace=FALSE)

sim.grid<-matrix(data=init,nrow=grid.size)

Function to count neighbours

```
#Here we are calculating the whether the cells go up or down on the column
#or left or right on the rows. We expand the grid
neighbours<-function(R,C) {
  Rs<-c(R-1,R,R+1)
  Cs<-c(C-1,C,C+1)
```

Now we correct for the boundaries

```
  Rs<-ifelse(Rs==0,grid.size,Rs)    #what to do if it's on an edge
  Rs<-ifelse(Rs==grid.size+1,1,Rs)
  Cs<-ifelse(Cs==0,grid.size,Cs)
  Cs<-ifelse(Cs==grid.size+1,1,Cs)

count<-ifelse(sim.grid[R,C]==1,sum(sim.grid[Rs,Cs])-1,sum(sim.grid[Rs,Cs]))  #count the neighbours. -1 for the on
e in the middle
```

count }

Loop each generation

Here we calculate the number of neighbors for each point in the grid. We run the code for each Generation. We save the current grid into a variable called the last grid so that you can compute the current grid by comparing it against the last grid. We create we create an iterator for the generation and this then repeats.

```
  for (i in 1:generations) {

  sim.grid.last<-sim.grid
```

Calculate new matrix with number of neighbours

neighbours.mat<-matrix(data=NA,nrow=grid.size,ncol=grid.size)

#p and q are indexes for the rows and the columns

for (p in 1:grid.size){ for (q in 1:grid.size){ neighbours.mat[p,q]<-neighbours(R=p,C=q) } }

Generate sim.grid for next generation

Here we set up the definitions for whether a cell is created, destroyed, or stays the same

sim.grid[sim.grid==0 & neighbours.mat==3]<-1 sim.grid[sim.grid==1 & neighbours.mat<2]<-0 sim.grid[sim.grid==1 & neighbours.mat>3]<-0

Plot distribution of live cells

plot.mat<-melt(sim.grid) ## convert matrix into long data frame

plot(plot.mat$Var1[plot.mat$value==1],plot.mat $Var2[plot.mat$value==1],cex=50*1/grid.size,col=“steelblue”,pch=16,xlim=c(0.5,grid.size+0.5),ylim=c(0.5,grid.size+0.5),xaxs=“i”,yaxs=“i”,xaxt=“n”,yaxt=“n”,xlab=

Introduce a 0.2 second pause for each iteration if slow=TRUE

If all dead break and show message

if(sum(sim.grid)==0) { text(x=grid.size/2,y=grid.size/2,“When you play the game of life,you win or you die.There is no middle ground”,cex=1.5)

break }

Data frame

I used the debugger and entered a break here because it includes the all the different variables in this data set. Further down if we check our environment their will be less variables. We have the neighbours.mat which is a 12x12 matrix which measures how many neighbor each cell has. plot.mat has 144 observations with three variables. Those variables are defined by three situations which are when a cell has three or more neighboring cells and a cell is destroyed, when there are no neighboring, cells, and lastly when two cells neighbour one and a new cell is created. It is created by the neighbours function which adds up rows and columns sim.grid is another 12x12 matrix and shows the initial grid in the game of life. sim.grid.last is the last grid created by game of life whether it be at the 50th iteration or when the game ends. As I said there are 50 generations which means that this program repeats itself until the cells enter a steady state or when we have hit the maximum iterations allowed in this code which is 50. i is includes 50 layers of integers Grid.size is 12. init is a list of of 144 numbers which are compiled of zeros and 1s. P and q are indices for the rows and columns of the matrix and are a 12 layers of integers. We have slow which a logical function telling us whether to slow down the process between iterations or not.

If steady state break and show message.

We define the End of the Game when the comparison of the current grid and past grid equal zero or when no cells are created or removed.As nothing will change for generations to come

if(sum(abs(sim.grid-sim.grid.last))==0) { text(x=grid.size/2,y=grid.size/2, “Unlike the universe,The Game of Lifehas reached a steady state”,cex=1.5)

```
break
```

}

} par(mar=c(5, 4, 4, 2) + 0.1) ##Reset margins }

life3()

We can repeat the same game we replace the sample function

p and q are indexes for the rows and the columns

for (p in 1:grid.size){

for (q in 1:grid.size){

neighbours.mat[p,q]<-neighbours(R=p,C=q) in order to repeat the exact same game.

I think that Life4() was much faster then Life3()

The iteration for Life4() basically uses matrices representing North, South, East, West, Northeast, Northwest, Southeast, Southwest are layered on top of eachother. Because these matrices cover are more expansive this means it has less calculations then Life3() where all points are calculated individually in Life3() and each cell is evaluated one at a time.