Introduction to `SpaDES`

Alex M. Chubaty
Natural Resources Canada, Pacific Forestry Centre
email: achubaty@nrcan.gc.ca

Eliot McIntire
Natural Resources Canada, Pacific Forestry Centre
email: emcintir@nrcan.gc.ca

August 8, 2014

Contents

1 Spatial Discrete Event Simulation (SpaDES)

Requirements This packages makes heavy use of the `raster` and `sp` packages, so familiarity with these packages and their classes and methods is recommended.

2 Discrete event simulation

Lorem ipsum ...

3 SpaDES modules

3.1 Module overview

SpaDES modules are event-based, meaning that different actions (calculations) are performed on data objects based on the order of scheduled events. Basically, a module consists of a collection of events which are scehduled depending on the rules of your simulation. Each event may evaluate or modify a simulation data object.

3.2 Events

Simulation event list Lorem ipsum ...

Module events Lorem ipsum ...

Module event dependencies Typically, each module schedules its own events (e.g., a ”fire” module may schedule ”burn” events) and only uses its own data objects. Modules that behave in this way are indepedent of each other, and this is generally the prefered way to design and implement modules. A module that schedules events from another module is said to depend on that module. Module event dependencies compilcate the construction of simulation models, and hinder the ability to develop and deploy models with modularity. If two modules are actually depedent on each others' events, then you should consider whether they really are separate modules or should be merged into a single module.

3.3 Objects

As you build your module / simulation, you can use any of R's data types to store your objects / data. In particular, matrices (including vectors) and lists work well for this purpose because they are pass-by-reference, which reduces your model's memory footprint and speeds up your codes execution. Other useful datatypes include Raster* and SpatialPoints* objects.

Global objects Use ■- to assign global objects to reduce copying large objects (such as maps), which slows model execution.

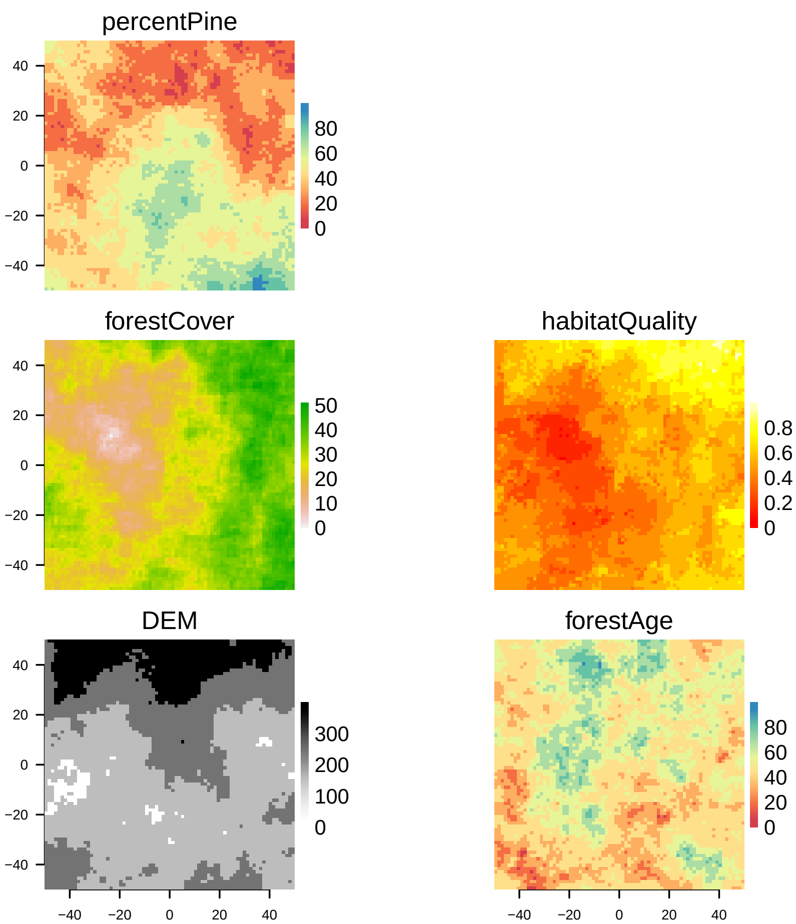
Module object dependencies As noted above, modules can depend on one another for event scheduling. Modules can also be design to rely on outputs (data objects) from other modules. A module that relies on a global simulation data object that previously used by another module is said to be dependent on that other module. It is often useful to develop collections of modules that interact indirectly and are dependent in this way. Note that modules need not be inter-dependent on one another: module B may depend on module A (for example to initialize a data object), without module A depending on module B.

4 Working with maps

A raster map Sample map of habitat quality.

Plotting maps

```
> # Give dimensions of dummy raster
> nx <- 1e2
> ny <- 1e2
> template <- raster(nrows=ny, ncols=nx, xmn=-nx/2, xmx=nx/2, ymn =-ny/2, ymx=ny/2)
> # Make dummy maps for testing of models:
> # - digital elevation model (DEM)
> # - forest age
> # - forset cover
> # - percent pine
> DEM <- round(GaussMap(template, scale=300, var=0.03, speedup=1), 1)*1000
> forestAge <- round(GaussMap(template, scale=10, var=0.1, speedup=1), 1)*20
> forestCover <- round(GaussMap(template, scale=50, var=1, speedup=1),2)*10
> percentPine <- round(GaussMap(template, scale=50, var=1, speedup=1),1)
> # Scale them as needed
> forestAge <- forestAge/maxValue(forestAge)*100
> percentPine <- percentPine/maxValue(percentPine)*100
> # Make layers that are derived from other layers
> habitatQuality <- (DEM+10 + (forestCover+5)*10)/100
> habitatQuality <- habitatQuality/maxValue(habitatQuality)
> # Stack them into a single stack for plotting
> habitat <- stack(list(DEM, forestAge, forestCover, habitatQuality, percentPine))
> names(habitat) <- c("DEM","forestAge", "forestCover", "habitatQuality", "percentPine")
> library(RColorBrewer)
> cols <- list(
+   transparent.greys <- c("#00000000",paste(brewer.pal(8,"Greys"),"66",sep="")[8:1]),
+   grey <- brewer.pal(9,"Greys"),
+   spectral <- brewer.pal(8,"Spectral"),
+   terrain <- rev(terrain.colors(100)),
+   heat <- heat.colors(10),
+   topo <- topo.colors(10)
+ )
> simPlot(habitat, col=cols[c(2:5,3)])
```


percentPine
forestCover
habitatQuality
DEM
forestAge

5 Simulating “agents”

Lorem ipsum ...

5.1 Spatial agents

5.1.1 Point agents

agents represented by a single set of coordinates indicating their current position.

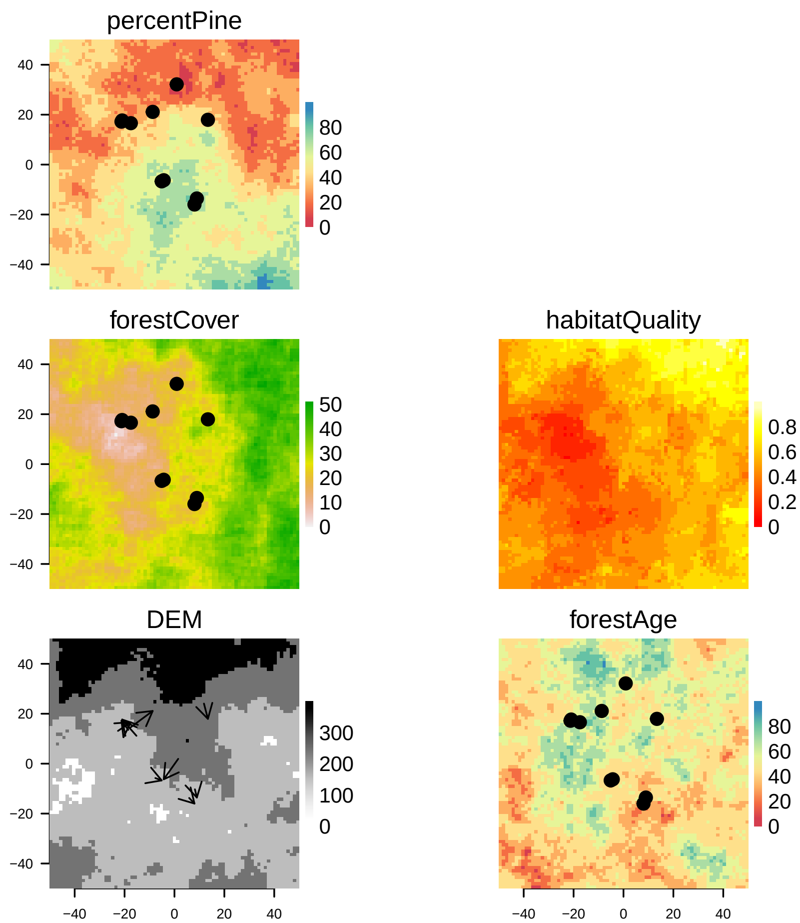
Use a `SpatialPointsDataFrame` with additional columns as needed.

Non-mobile point agents e.g., plants

Mobile point agents e.g., animals use a `SpatialPointsDataFrame`, with additonal columns for agents' previous `n` positions, and any other columns such as age, sex, group membership, etc.

```
> N <- 1e1 # number of agents
> # caribou data vectors
> IDs <- c("Alice", "Bob", "Clark", "Daisy", "Eric",
+          "Franz", "Gabby", "Hayley", "Igor", "Jane")
> sex <- c("female", "male", "male", "female", "male",
+          "male", "female", "female", "male", "female")
> age <- round(rnorm(N, mean=8, sd=3))
> prevX <- runif(N, xmin(habitat)+(ncol(habitat)*0.2), xmax(habitat)-(ncol(habitat)*0.2)) # previous X l
> prevY <- runif(N, ymin(habitat)+(nrow(habitat)*0.2), ymax(habitat)-(nrow(habitat)*0.2)) # previous Y l
> # create the caribou agent object
> caribou <- SpatialPointsDataFrame(coords=cbind(x=rnorm(N, prevX, ncol(habitat)/20),
+                                                y=rnorm(N, prevY, ncol(habitat)/20)),
+                                   data=data.frame(prevX, prevY, sex, age))
> row.names(caribou) <- IDs # alternatively, add IDs as column in data.frame above
> heading(SpatialPoints(cbind(x=prevX,y=prevY)),caribou)

    Alice       Bob     Clark     Daisy      Eric     Franz     Gabby    Hayley
359.04302  56.15213 318.61165 136.39021 323.25320 215.29489  53.47163 110.29392
     Igor      Jane
166.85858 165.51945

> coordinates(caribou)

                 x          y
Alice    0.8974701  32.151768
Bob    -17.4771796  16.596777
Clark  -20.9706302  17.674747
Daisy    8.0323081 -15.986443
Eric   -21.2326637  17.196287
Franz   -4.2999827  -6.272255
Gabby   -8.7226061  21.106907
Hayley  -5.1215720  -6.708804
Igor     8.9780478 -13.574948
Jane    13.4029546  17.948136

> ## conventional plotting method - agents don't plot properly when it is a raster stack
> #plot(habitat)
> #plot(caribou, add=TRUE)
```

```
>
> # convenient plotting using simPlot
> simPlot(habitat, col=cols[c(2:5,3)])
> simPlot(caribou, on.which.to.plot=c(2,3,5), pch=19, size=unit(0.1,"inches"))
> drawArrows(from=SpatialPoints(cbind(x=prevX, y=prevY)),
+            to=caribou,
+            on.which.to.plot="DEM")
```



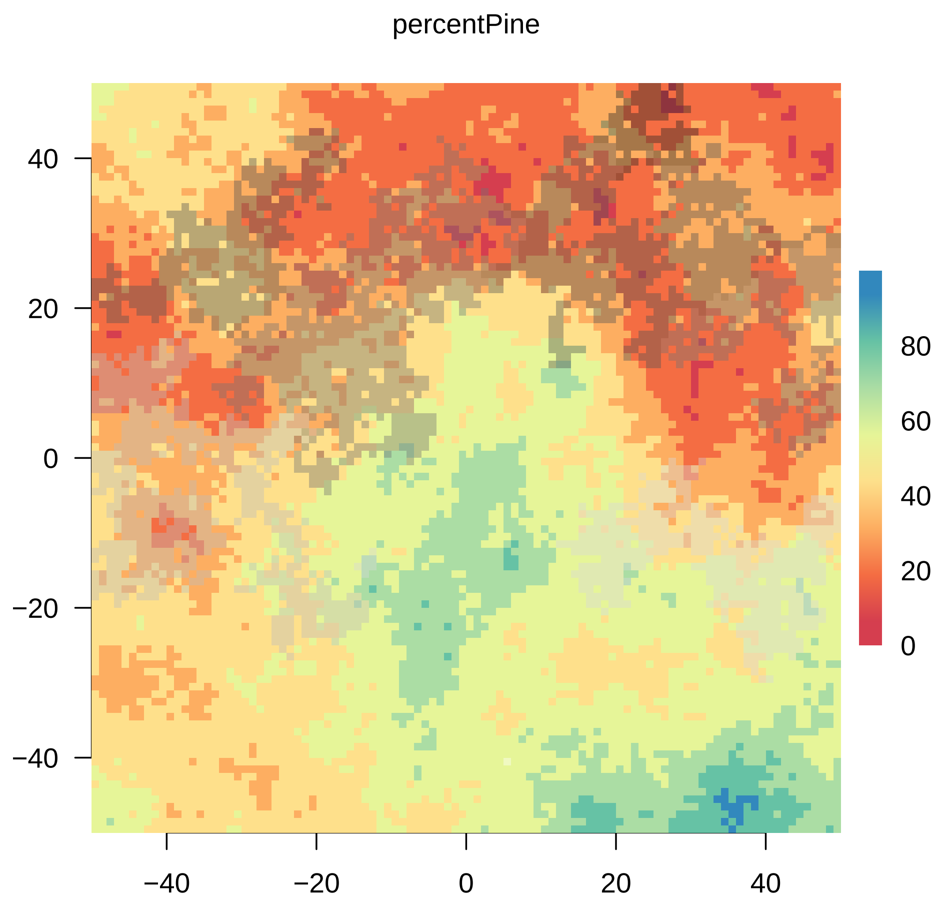
6 A simple fire model

Burn some of the forest Using the spread function, we can simulate fires, and subsequent changes to the various map layers. Here, spreadProb can be a single probability or a raster map where each pixel has a probability. In the example below, each cell's probability is taken from the Percent Pine map layer.

```
> nFires <- 10 # number of agents
> habitat[["Fires"]] <- spread(habitat[[1]],
+                              loci=as.integer(sample(1:ncell(habitat), nFires)),
+                              spreadProb=0.225,
+                              #spreadProb=habitat[["percentPine"]]/(maxValue(habitat[["percentPine"]])*
+                              persistance=0,
+                              mapFireID=TRUE,
```

```
+                                 mask=NULL,
+                                 maxSize=1e8,
+                                 directions=8,
+                                 iterations=1e6,
+                                 plot.it=FALSE,
+                                 mapID=TRUE)
> simPlot(habitat[["Fires"]])
```

```
> # Show the burning more strongly over abundant pine
> simPlot(habitat[["percentPine"]], col=cols[[3]])
> simPlot(habitat[["Fires"]], add=TRUE, delete.previous=FALSE, col=cols[[1]])
```



We can see that the fires tend to be in the Pines because we made it that way, using an arbitrary weighting with pine abundance

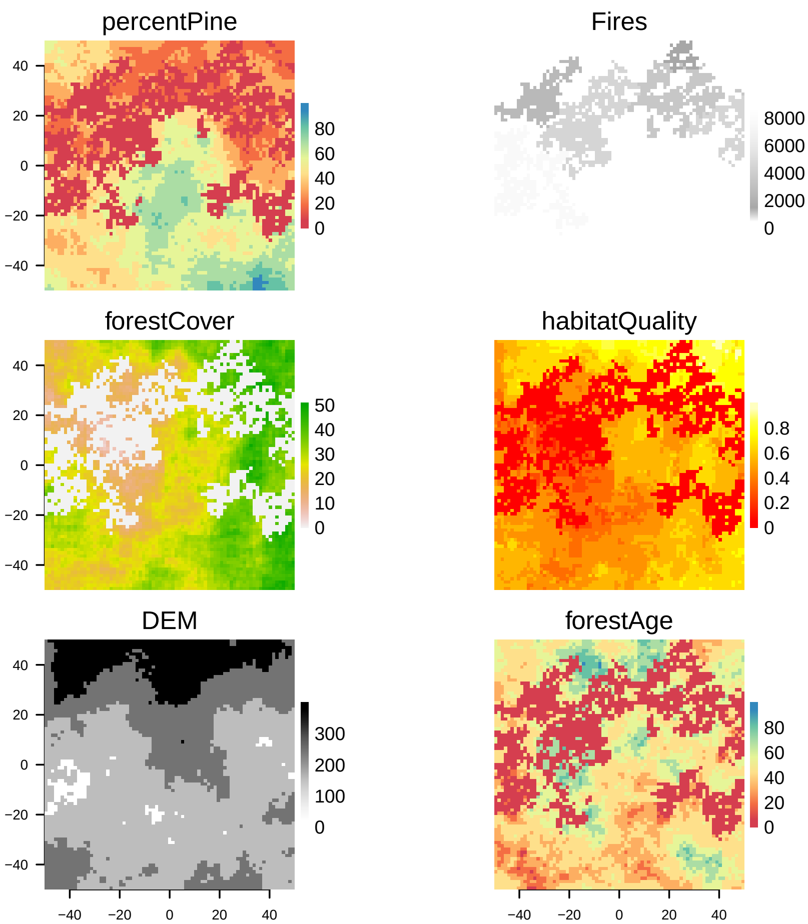
```
> # Show the burning more strongly over abundant pine
> fire <- reclassify(habitat[["Fires"]],rcl= cbind(0:1,c(0,ncell(habitat)),0:1))
> pine <- reclassify(habitat[["percentPine"]],rcl= cbind(0:9*10, 1:10*10, 0:9))
> PineByFire <- crosstab(fire, pine, long=TRUE)
> colnames(PineByFire) <- c("fire", "pine", "freq")
```

```
> PineByFire$pine <- as.numeric(as.character(PineByFire$pine))
> summary(glm(freq ~ fire*pine, data=PineByFire, family="poisson"))

Call:
glm(formula = freq ~ fire * pine, family = "poisson", data = PineByFire)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-31.185  -16.143   -0.408   15.296   20.611

Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept)  6.927794   0.019797 349.933   <2e-16 ***
fire1       -0.570639   0.035808 -15.936   <2e-16 ***
pine        -0.082486   0.004162 -19.817   <2e-16 ***
fire1:pine  -0.084710   0.009627  -8.799   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for poisson family taken to be 1)

    Null deviance: 7171  on 17  degrees of freedom
Residual deviance: 5182  on 14  degrees of freedom
AIC: 5324.6

Number of Fisher Scoring iterations: 5
```

Sure enough, there are more fires as the abundance of pine goes up, as seen by the positive interaction term (the negative `fire1` term means that there are more pixels without fires than with fires).

Impact some of the forest

```
> habitat[["forestAge"]][habitat[["Fires"]]>0] <- 0
> habitat[["forestCover"]][habitat[["Fires"]]>0] <- 0
> habitat[["habitatQuality"]][habitat[["Fires"]]>0] <- 0.1
> habitat[["percentPine"]][habitat[["Fires"]]>0] <- 0
> simPlot(habitat, col=cols[c(2:5,3,1)])
```

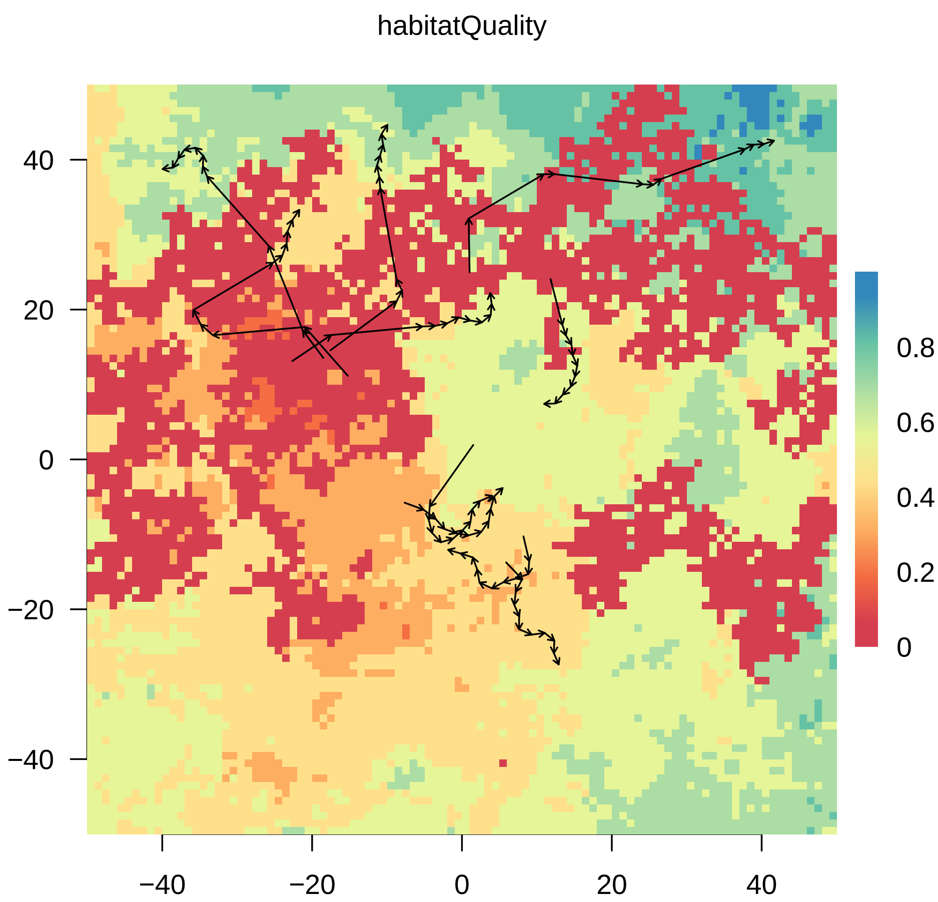


7 A simple individual based model (IBM)

Move some agents Using a simple habitat-depedent correlated random walk, simulate the movement of caribou across a heterogeneous landscape. Because we had just had fires, and we assume that fires have a detrimental effect on animal movement, we can see the long steps taken in the new, low quality, post-burn sections of the landscape.

```
> simPlot(habitat[["habitatQuality"]], col=cols[[3]])
> for (i in 1:10) {
+   #crop any caribou that went off maps
+   caribou <<- crop(caribou,habitat)
+   drawArrows(from=SpatialPoints(cbind(x=caribou$prevX, y=caribou$prevY)),
+              to=caribou,length=0.04,
+              on.which.to.plot=1)
+
+   # find out what pixels the individuals are on now
+   ex <- habitat[["habitatQuality"]][caribou]
+
+   #step length is a function of current cell's habitat quality
+   sl <- 0.25/ex
+
```

```
+   ln <- rlnorm(length(ex), sl, 0.02) # log normal step length
+   sd <- 30 # could be specified globally in params
+
+   caribou <<- crw(caribou, stepLength=ln, stddev=sd, lonlat=FALSE)
+ }
```



8 Further reading

8.1 Other `SpaDES` vignettes:

- `modules`: Building modules in `SpaDES`
- `plotting`: Plotting with `simPlot` in `SpaDES`