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 14, 2014

Contents

1 Spatial Discrete Event Simulation (SpaDES)

1.1 Overview

Why SpaDES exists

1.2 What is Discrete Event Simulation

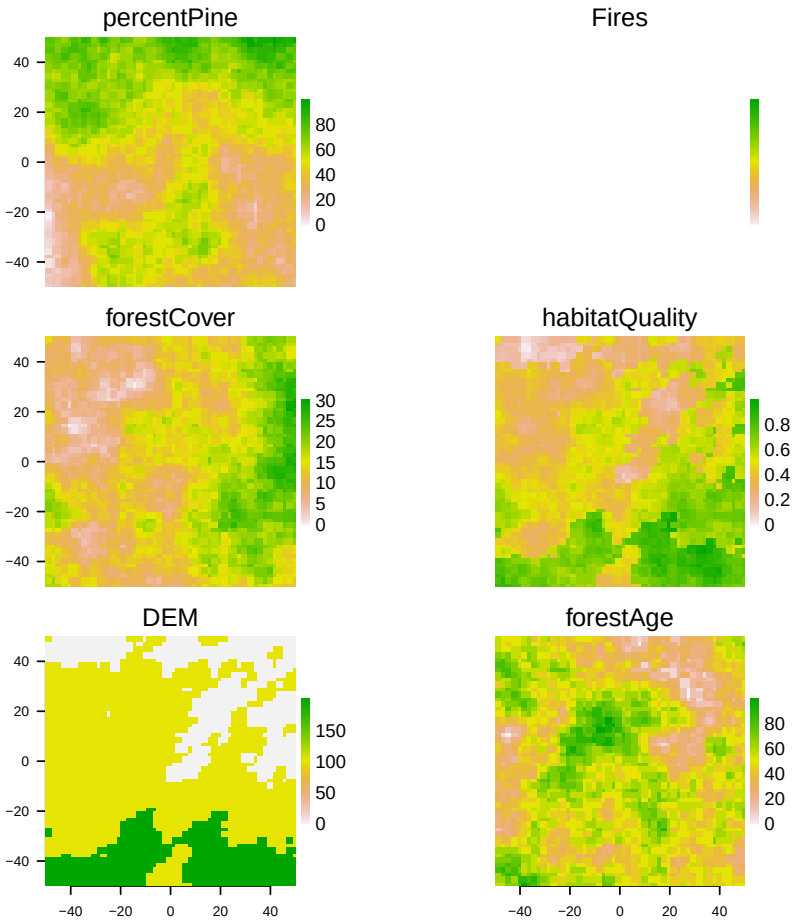
1.3 SpaDES is a DES

2 Using SpaDES to build discrete event simulations

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

Talk about doSim and simInit.

```
> times = list(start=0,stop=10.02)
> parameters = list(
+                     .globals=list(mapName="landscape"),
+                     .progress=list(NA),
+                     randomLandscapes = list(nx=1e2, ny=1e2, .saveObjects="landscape",
+                                             .plotInitialTime = 0, .plotInterval=1e3,
+                                             startTime=0),
+                     fireSpread=list(nFires = 1e1, spreadprob=0.225,
+                                     persistprob=0, its=1e6,
+                                     .plotInitialTime = 0.1, .plotInterval=10,
+                                     returnInterval = 10, startTime=0),
+                     caribouMovement=list(N=1e2,
+                                 .plotInitialTime = 1.01, .plotInterval=1,
+                                 moveInterval=1, startTime=0)
+                   )
> modules = list("randomLandscapes", "fireSpread", "caribouMovement")
> path = system.file("sampleModules", package="SpaDES")
> mySim <- simInit(times=times, params=parameters, modules=modules, path=path)
> doSim(mySim)
```


percentPine
80
60
40
20
0
Fires
forestCover
30
25
20
15
10
5
0
habitatQuality
0.8
0.6
0.4
0.2
0
DEM
150
100
50
0
forestAge
80
60
40
20
0
40
20
0
−20
−40

3 SpaDES modules

3.1 Module overview

`SpaDES` modules are event-based, meaning that different actions 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, or perform other operations such as saving and loading data objects.

3.2 Events

3.2.1 Simulation event list

Lorem ipsum ...

3.2.2 Module events

Lorem ipsum ...

3.2.3 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.

For more information about module dependencies please see the `modules` vignette.

3.3 Objects

3.3.1 Data types

As you build your modules for your simulation, you can use any of R's data types to store your objects and data. In particular, matrices (including vectors) and lists work well for this purpose because as of R version 3.1 they are more efficient and reduce your model's memory footprint and speeds up your codes execution. Other useful datatypes include `Raster*` and `SpatialPoints*` objects.

3.3.2 Global objects

Use the superassignment operator (`<<-`) to assign global objects to reduce copying large objects (such as maps), which slows model execution.

3.3.3 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 is also 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 on shared data objects.

For more information about module dependencies please see the `modules` vignette.

4 Reading data or maps from files

There is a function, simLoad, that can use the simList or a fileList directly, to load maps or data from files. This can ease loading of many types of files into R.

5 Simulating "agents"

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.

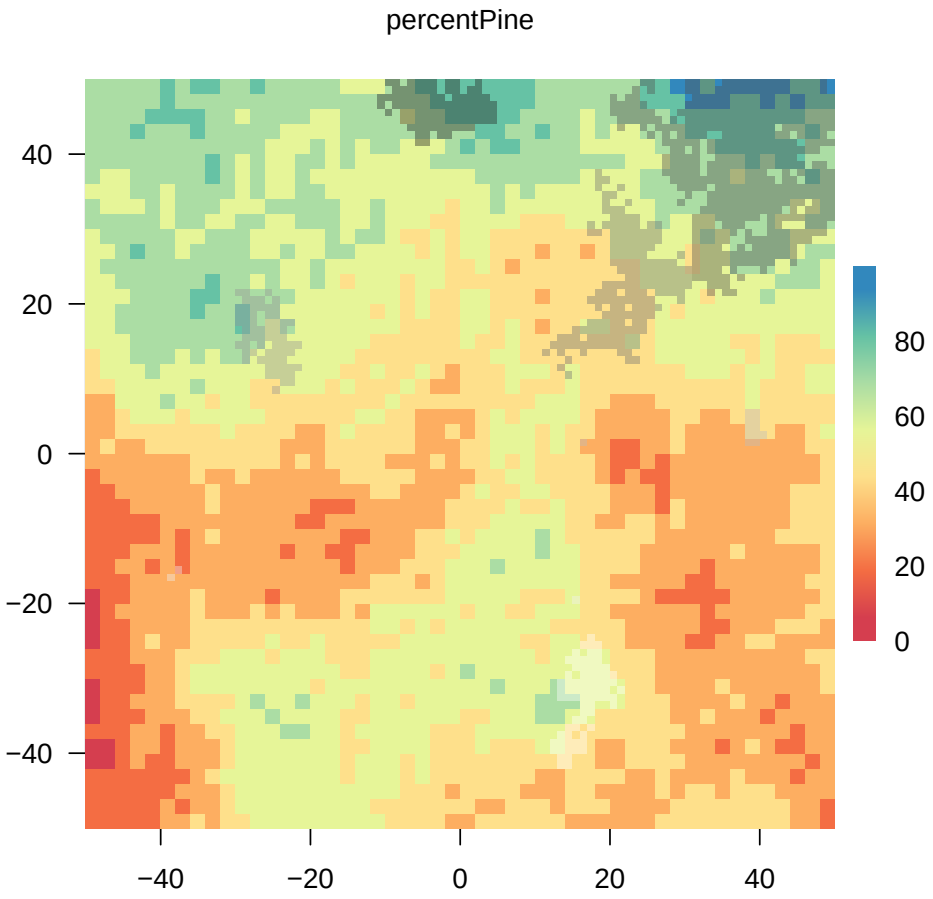
Non-mobile point agents e.g., plants

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.

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
> landscape[["Fires"]] <-
+   spread(landscape[[1]],
+          loci=as.integer(sample(1:ncell(landscape), nFires)),
+          spreadProb=landscape[["percentPine"]]/(maxValue(landscape[["percentPine"]])*5)+0.1,
+          persistance=0,
+          mapFireID=TRUE,
+          mask=NULL,
+          maxSize=1e8,
+          directions=8,
+          iterations=1e6,
+          plot.it=FALSE,
+          mapID=TRUE)
> simPlot(landscape[["Fires"]])

> # Show the burning more strongly over abundant pine
> simPlot(landscape[["percentPine"]], col=.cols[[3]])
> simPlot(landscape[["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

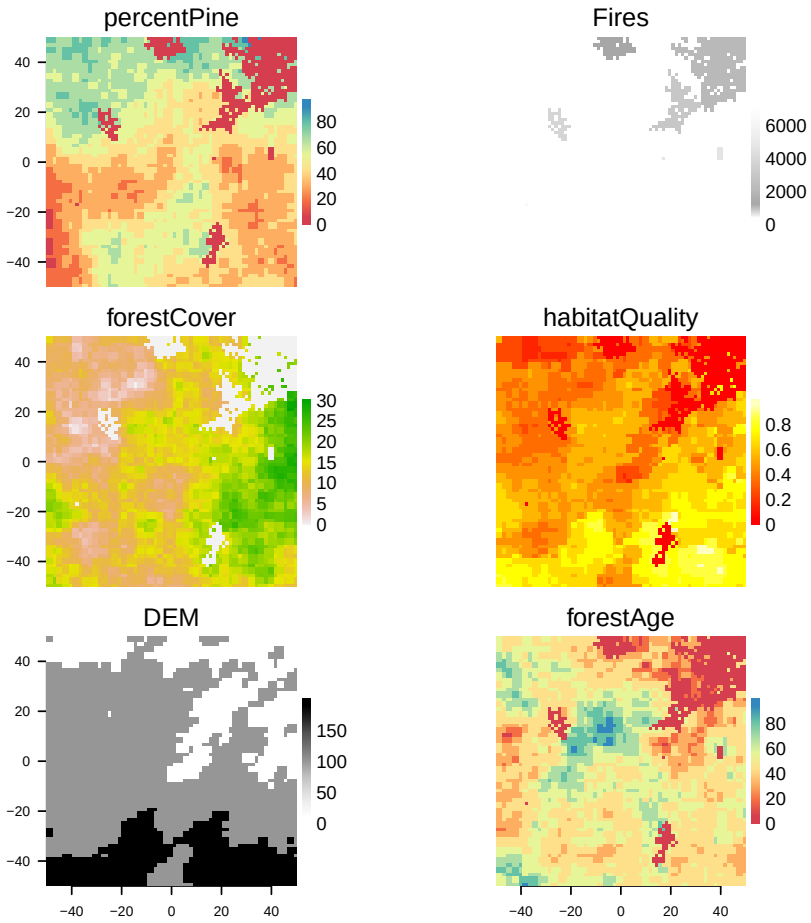
```
> # Show the burning more strongly over abundant pine
> fire <- reclassify(landscape[["Fires"]],rcl= cbind(0:1,c(0,ncell(landscape)),0:1))
> pine <- reclassify(landscape[["percentPine"]],rcl= cbind(0:9*10, 1:10*10, 0:9))
> PineByFire <- crosstab(fire, pine, long=TRUE)
> colnames(PineByFire) <- c("fire", "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
-40.631  -10.346   -2.539    8.542   34.637

Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept)  7.027853   0.018319  383.64   <2e-16 ***
fire1       -3.478185   0.093695  -37.12   <2e-16 ***
```

```
pine        -0.050061   0.003674  -13.63   <2e-16 ***
fire1:pine   0.239021   0.014292   16.72   <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: 14668.2  on 18  degrees of freedom
Residual deviance:  7326.6  on 15  degrees of freedom
AIC: 7463

Number of Fisher Scoring iterations: 6
```

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

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

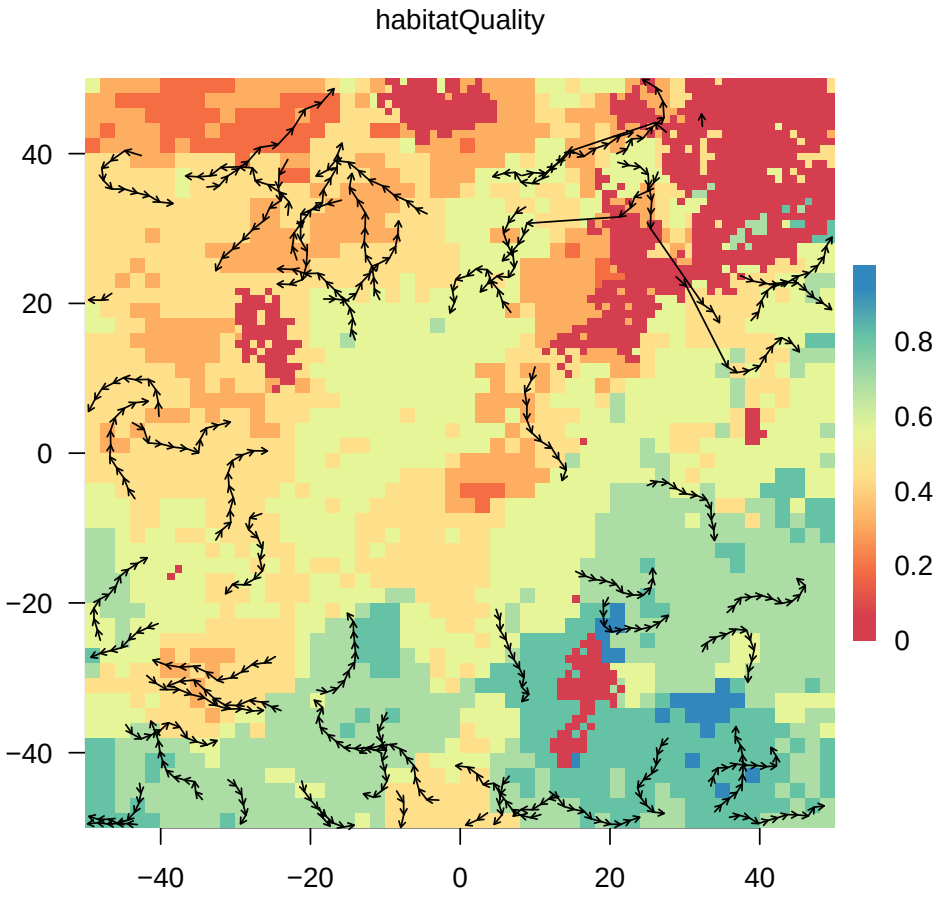


7 A simple individual based model (IBM)

Move some agents Using a simple landscape-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(landscape[["habitatQuality"]], col=.cols[[3]])
> for (i in 1:10) {
+   #crop any caribou that went off maps
+   caribou <<- crop(caribou,landscape)
+   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 <- landscape[["habitatQuality"]][caribou]
+
+   #step length is a function of current cell's landscape 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`