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 26, 2014

Abstract

Easily implement a variety of simulation models, with a focus on spatially explicit agent based models. The core simulation components are built upon a discrete event simulation framework that facilitates modularity, and easily enables the user to include additional functionality by running user-built simulation modules. Included are numerous tools to visualize raster and other maps.

Website: `https://github.com/achubaty/SpaDES`

Contents

1 Introduction

1.1 Objectives and motivations

We have been building spatial simulation models for several years and got tired of rebuilding the wheel every time. So, we had been looking for a generic platform that could be used to make new models quickly, and provide a link to existing models, so that an already well described and mature model, can be used with a *de novo* one. Or better yet, use several *de novo* models and several existing models in combination. This approach requires a platform that allows for modular reuse of models (herein called "modules") as hypotheses that can be evaluated and tested in various ways.

1. Allow rapid building of models of a wide diversity of types (IBMs, raster models, differential equation models etc.).
2. Run faster and more memory efficiently than current systems for doing similar things (NetLogo, SELES, Repast etc.).
3. Take advantage of R's strengths as a data platform, its capabilities to call external software such as databases, its excellent visualization and graphics, and its abilities for high performance computing. We don't have to implement all of these from scratch ourselves!
4. Be open source, but also make it as easy as possible for many people to easily contribute modules and code. In particular, to make it easy for scientists already familiar with R but who aren't formally trained as computer programmers.
5. `SpaDES` is built around modularity so that models can be seen as modules that are easily replaceable, not just "in theory" replaceable.
6. Allow tight coupling between data and model simulations so that calibration is not actually something that one has to redesign every time there is a new data set.

1.2 Discrete event simulation and `SpaDES`

Discrete event simulation is

event driven: an activity changes the state of the system at particluar times. a particular activity may have several events assoriated with it. future events are scheduled in an event queue, and then processed in order.

rather than advancing time in fixed increments, time advances to the time of the next event in the queue (assume that state of the system only changes due to events, therefore no change between events, so no need to run the clock).

'time' as the core concept linking simulation components: activities schedule events (i.e. change the system according to their programmed rules) and do not need to know about each other, which allows for modularity of simulation components. thus, complex simulations involving multiple processes (activities) can be build fairly easily, provided these processes are modelled using a common DES framework.

`SpaDES` provides such a framework, facilitating interaction between multiple processes (built as "modules") that don't interact with one another directly, but are scheduled in the event queue and carry out operations on shared data objects in the global simulation evnivironment. This package provides tools for easily building modules natively in R that can be reused; however, because of the flexibilty R provides for interacting with other programming langugages and external datasources, modules can also be built using external tools.

1.3 `SpaDES` demos and sample modules

The static nature of PDFs does not allow us to really show off the simulation visualization components of this package, so we invite you to check out the included demos, to run the sample simulation provided in this vignette, and to view the source code for the sample modules included in this package.

Demos:

```
> library("SpaDES")
> # demo: randomLandscapes, fireSpread, caribouMovement
> demo("spades-simulation", package="SpaDES")
```

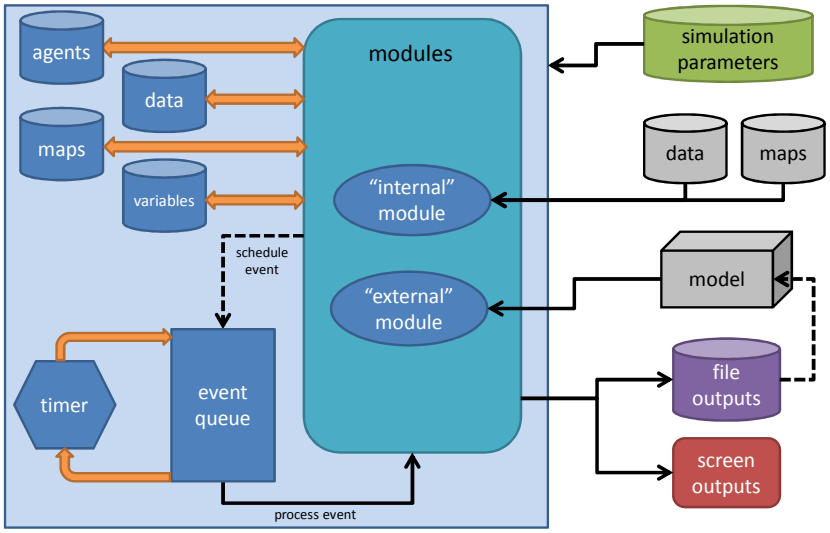
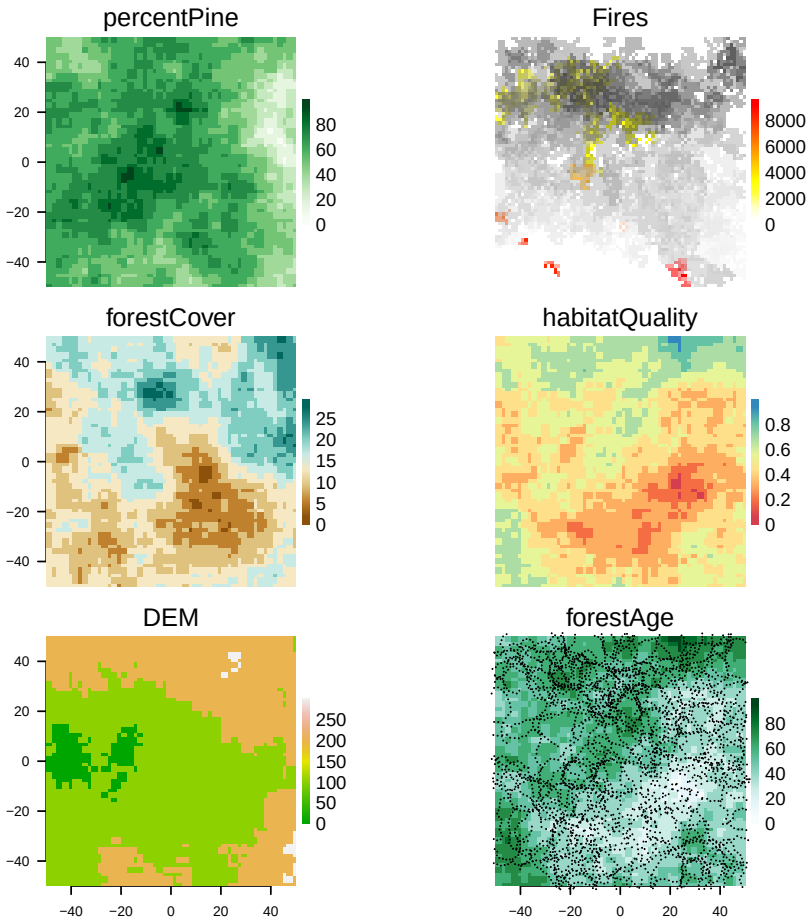


Figure 1: Schematic representation of a SpaDES simulation model.

Sample model:

```
> library("SpaDES")
> .cols[[5]] <- c("#FFFFFFFF", rev(heat.colors(9)))
> outputPath=file.path("~", "tmp", "simOutputs")
> times <- list(start=0,stop=100)
> parameters <- list(.globals=list(mapName="landscape", .outputPath=outputPath,
+                                  burnStats="nPixelsBurned"),
+                    .progress=list(NA),
+                    randomLandscapes=list(nx=1e2, ny=1e2, inRAM=TRUE),
+                    fireSpread=list(nFires= 1e1, spreadprob=0.225, its=1e6,
+                                    persistprob=0, returnInterval=10, startTime=0,
+                                    .plotInitialTime=0.1, .plotInterval=10),
+                    caribouMovement=list(N=1e2, moveInterval=1,
+                                         .plotInitialTime=1.01, .plotInterval=1)
+                    )
> modules <- list("randomLandscapes", "fireSpread", "caribouMovement")
> path <- system.file("sampleModules", package="SpaDES")
> mySim <- simInit(times=times, params=parameters, modules=modules, path=path)
> spades(mySim)
```


percentPine
40
20
0
−20
−40
80
60
40
20
0
Fires
8000
6000
4000
2000
0
forestCover
25
20
15
10
5
0
habitatQuality
0.8
0.6
0.4
0.2
0
DEM
250
200
150
100
50
0
−40
−20
0
20
40
forestAge
80
60
40
20
0

2 Using SpaDES to build 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. Plotting features are built using the `grid` package.

2.1 Setting up a simulation:

As you can see in the sample simulation code provided above, setting up and running a simulation in `SpaDES` is straightforward using existing modules. You need to specify somethings about the simulation environment, including all parameter values passed to the simulation, which modules to use for the sim, and any global data objects that should be used to stored the simulation state. Each of these are passed as named lists to the simulation object upon initilization.

Initializing a simulation: The details of each simulation are stored in an S4 `simList` object, including the simulation parameters and modules used, as well as storing the current state of the simulation and the future event queue. A list of all competed events is also stored, which can provide useful debugging information.

A new simulation is initialized using the `simInit` function, which does all the work of creating the `simList` object for your simulation, setting all the slot values appropriately. Furthermore, this function tries to provide additional feedback to the user regarding parameters that may be improperly specified.

You can inspect the contents of a `simList` object as you would any other R object (e.g., `print(mySim)`).

2.2 Running a simulation:

Once a simulation is properly initialized it is executed using the `spades` fuction. By default, a progress bar is displayed in the console (this can be customized), and [WHAT ELSE HAPPENS BY DEFAULT?]. Debugging mode (setting `spades(mySim, debug=TRUE)`) prints the contents of the `simList` object after the completion of every event during simulation.

3 SpaDES modules

`SpaDES` modules are event-driven, meaning that different actions are performed on data objects based on the order of scheduled events. Basically, a module describes the processes or activities that drive simulation state changes. Each activity consists of a collection of events which are scheduled 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 or plotting.

The power of `SpaDES` is in modularity and the ease with which existing modules can be modified and new modules created. This vignette will highlight general use of the package and its features using the sample modules provided. Creating and customizing modules is a whole topic unto itself, and for that reason we have created a separate `modules` vignette with more details on module development.

4 Simulation and data

importance of linking simulation and data

Bringing your data into R is easy, and you can use any of the built in data import tools. To facilitate this, we have provided additional functionality to easily load maps or data from files via the load module. To automatically import a list of files, simply provide it as a parameter named `.loadFileList` when initializing the simulation. See `?loadFiles` and the modules vignette for more information on the load module.

```
> ### Example: loading habitat maps
>
> # use all built-in maps from the SpaDES package
> pathToMaps <- file.path(find.package("SpaDES", quiet=FALSE), "maps")
> fileList <- data.frame(files=dir(pathToMaps, full.names=TRUE, pattern= "tif"),
+                        functions="rasterToMemory", packages="SpaDES",
+                        stringsAsFactors=FALSE)
> # this list can be passed to simInit() as an entry in the parameter list
> mySim <- simInit(times=list(start=0.0, stop=10.0),
+                  params=list(
+                    .loadFileList=fileList,
+                    .progress=list(.graphical=FALSE, .progressInterval=10),
+                    .globals=list(mapName="landscape", burnStats="nPixelsBurned"),
+                    fireSpread=list(nFires=1e1, spreadprob=0.225, persistprob=0,
+                                    its=1e6,returnInterval=10, startTime=0,
+                                    .plotInitialTime = 0.1, .plotInterval=10)
+                  ),
+                  modules=list("stackFileList", "fireSpread"),
+                  path=system.file("sampleModules", package="SpaDES"))
> spades(mySim, debug=TRUE)
```

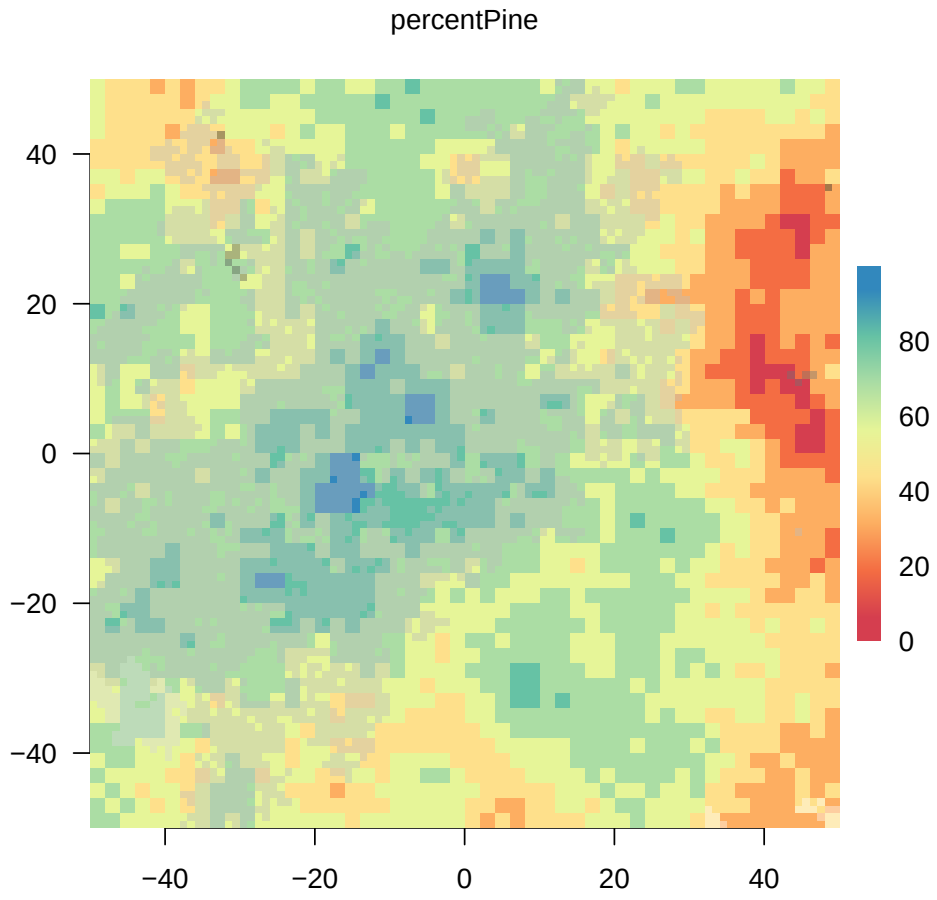
5 Modelling spread processes

5.1 A simple fire model

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
> landscape[["Fires"]] <-
+   spread(landscape[["percentPine"]],
+          loci=as.integer(sample(1:ncell(landscape), nFires)),
+          spreadProb=landscape[["percentPine"]]/(maxValue(landscape[["percentPine"]])*5)+0.1,
+          persistance=0,
+          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]])
```



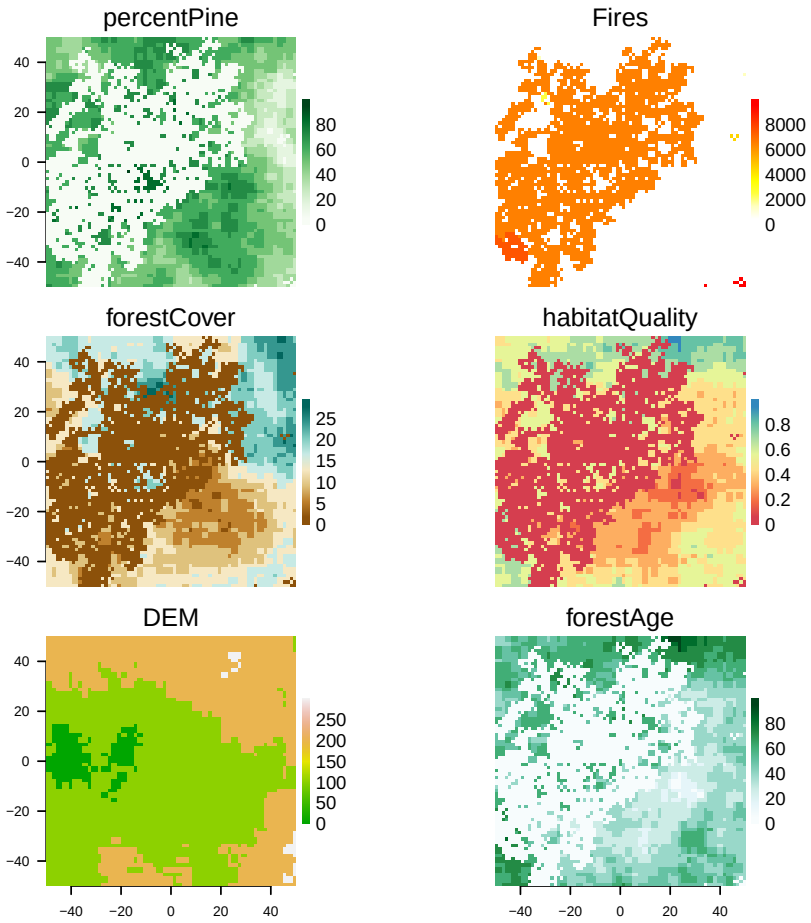
We can see that the fires tend to be in the Pines because we made it that way, using an arbitrary weighting with pine abundance

```
> # 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
-35.749  -15.534   -7.479   10.311   32.282

Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept)  6.196484   0.025355  244.39   <2e-16 ***
fire1       -1.679562   0.051669  -32.51   <2e-16 ***
pine         0.045855   0.004486   10.22   <2e-16 ***
fire1:pine   0.216908   0.007917   27.40   <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: 10155.6  on 19  degrees of freedom
Residual deviance:  7673.3  on 16  degrees of freedom
AIC: 7819.8

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(4, 7, 10, 3, 8, 5)])
```


percentPine
40
20
0
−20
−40
80
60
40
20
0
Fires
8000
6000
4000
2000
0
forestCover
25
20
15
10
5
0
habitatQuality
0.8
0.6
0.4
0.2
0
DEM
250
200
150
100
50
0
−40
−20
0
20
40
forestAge
80
60
40
20
0

6 Agent based modelling

A primary goal of developing 'SpaDES' was to facilitate the development of agent-based models (ABMs), also known as individual-based models (IBMs).

6.1 Point agents

As ecologists, we are usually concerned with modelling individuals (agents) in time and space, and whose spatial location (position) can be represented as a single point on a map. These types of agents can easily be represented most simply by a single set of coordinates indicating their current position, and can simulated using a `SpatialPoints` object. Additionally, a `SpatialPointsDataFrame` can be used, which provides storage of additional information beyond agents' coordinates as needed.

these objects to to be named to work properly with SpaDES plotting, so you need to used the `SpatialPoint*Named` classes, which is easily done using `name`. This converts a `SpatialPoints*` object to its equivalent `SpatialPoint*Named`.

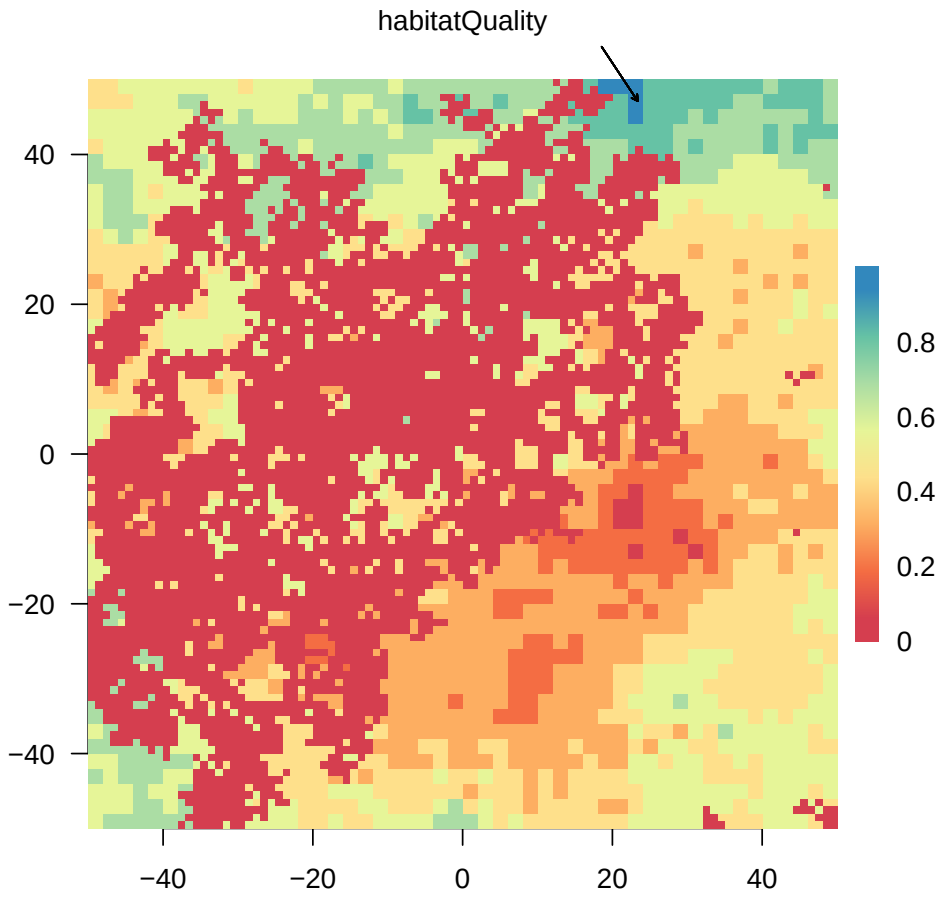
To model mobile point agents, e.g., animals (as opposed to non-mobile agents such as plants), use a `SpatialPointsDataFrame` containing additional columns for storing agents' previous `n` positions.

```
> N <- 10 # number of agents
> # caribou data vectors
> IDs <- letters[1:N]
> sex <- sample(c("female", "male"), N, replace=TRUE)
> age <- round(rnorm(N, mean=8, sd=3))
> x1 <- runif(N, 0, 100) # previous X location
> y1 <- runif(N, 0, 100) # previous Y location
> # caribou (current) coordinates
> x0 <- rnorm(N, x1, 5)
> y0 <- rnorm(N, y1, 5)
> # create the caribou agent object
> caribou <- SpatialPointsDataFrame(coords=cbind(x=x0, y=y0),
+                                   data=data.frame(x1, y1, sex, age))
> row.names(caribou) <- IDs
>
> # caribou needs to be a named object for plotting
> #name(caribou) <- "caribou"
```

Using a simple landscape-depedent correlated random walk, we 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 (t in 1:10) {
+   #crop any caribou that went off maps
+   caribou <<- crop(caribou,landscape)
+   drawArrows(from=SpatialPoints(cbind(x=caribou$x1, y=caribou$y1)),
+              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)
+ }
```



6.1.1 Polygons agents

Analogously, possible to use `SpatialPolygons*`, but we don't have `simPlot` plotting methods yet.

7 Putting it all together

Running multiple simulations with different parameter values is a critical part of sensitivity analysis, simulation experiments, optimization, and pattern oriented modelling. Below is a greatly simplified example, using the sample `randomLandscapes` and `fireSpread` modules. *NB only two parameters are varied; no outputs are saved; and the analyses done here are kept simple for illustrative purposes. A substantial portion of the simulation time is spent generating the random maps!*

```
> ### WARNING this can take a while to run, especially for large mapSizes.
>
> rasterOptions(maxmemory=1e9)
> # list all parameter values to run sims with
> parameters <- list(mapSize=round(sqrt(c(1e4, 1e5, 1e6, 1e7, 1e8))),
+                    pSpread=seq(0.05, 0.25, 0.05))
> # create data.frame with all parameter combinations
> paramsdf <- expand.grid(parameters)
> # outputs
> nPixelsBurned <- numeric()
> meanPixelsBurned <- cbind(paramsdf, pmean=NA, psd=NA)
> set.seed(42)
> for (i in 1:nrow(paramsdf)) {
+   # initialize each simulation with a param combo from paramsdf
+   mySim <- with(paramsdf,
+                 simInit(times=list(start=0.0, stop=20.0),
+                         params=list(
+                           .progress=list(.graphical=NA, .progressInterval=NA),
+                           .globals=list(mapName="landscape", burnStats="nPixelsBurned"),
+                           randomLandscapes=list(nx=mapSize[i], ny=mapSize[i],
+                                                 inRAM=TRUE),
+                           fireSpread=list(nFires=1000, spreadprob=pSpread[i],
+                                           persistprob=0, its=1e6,
+                                           returnInterval=10, startTime=0)
+                           ),
+                         modules=list("randomLandscapes", "fireSpread"),
+                         path=system.file("sampleModules", package="SpaDES")))
+   mySim <- spades(mySim)
+
+   # collect stats for each run
+   proportionBurned <- with(paramsdf, nPixelsBurned / (mapSize[i]^2))
+   meanPixelsBurned[i, "pmean"] <- mean(proportionBurned)
+   meanPixelsBurned[i, "psd"] <- sd(proportionBurned)
+
+   # cleanup between runs
+   rm(landscape, mySim, nPixelsBurned)
+   for (j in 1:10) gc()
+ }
> # overall statistics
> pch <- c(21:25)
> col <- brewer.pal(5, "Set1")
> with(meanPixelsBurned, plot(pmean ~ pSpread, xlab="Spread probability",
+                             ylab="Mean proportion of pixels burned",
+                             ylim=c(0,1), pch=pch, cex=1.5, col=col))
> with(parameters, legend("topleft", legend=formatC(mapSize^2, digits=0),
+                               pch=pch, col=col, cex=1.2))
```

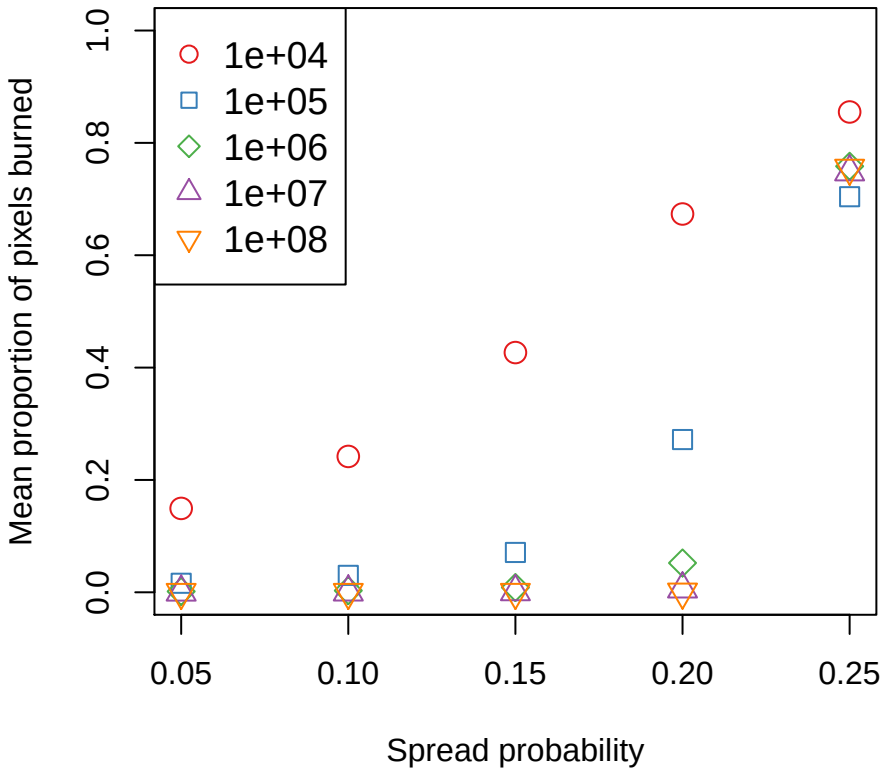


Figure 2: Mean proportion of pixels burned for maps of various sizes and fire spread probabilities.

8 Additonal resources

8.1 SpaDES documentation and vignettes:

Vignettes:

- `introduction`: Introduction to SpaDES: A package to develop and run spatially explicit discrete event simulation models. [This vignette.]
- `modules`: Building modules in `SpaDES`.
- `plotting`: Simulation visualization and plotting in `SpaDES`.

8.2 Reporting bugs:

Contact us via the package GitHub site: `https://github.com/achubaty/SpaDES/issues`.