So, it's finally time to generate tree maps. When we talked about generating tree maps last, we spoke about the fact that the minimum useful plot size in SORTIE-ND is 300m x 300m. Our validation plots, from the USGS, range from about a hectare to slightly larger. So, we can do a few things.

We could run simulations with our sample plots in the center, and bare ground all around them, but that wouldn't be feasible, because no forest plot is truly surrounded by bare ground.

We could also place our sample plots in the center, and then generate similar plots to fill in the surrounding space. That is the plan of action.

In the end, we need a tab-delimited file that has six columns: X, Y, Species, Type, Diam, Height. The first row is column headers.

So, we need to do a few things. First, we need to get our USGS plot maps ready, using only data from their origin years. We need to put those plot maps in the appropriate space (between 100-200 in x,y plot space). Then, we need to figure out what distribution matches the distribution of dbh's most closely for each species – probably normal or poisson, for ease – and randomly generate (x,y), species, and dbh for the remaining 8 hectares of space. We might be able to use "generatePlotMap" from MakeMyForests, if we modify it to match the tree-map that we need.

So first, let's convert the rotated plots that we already have into appropriate plot space. I'll import those CSVs and write them to an RData outside of this Sweave, then go from there.

```
> library(MakeMyForests)
> library(SortieTreeMaps)
> load("../data/rotPlots.RData")
```

Okay, let's figure out the true sizes of our plots, since ideally, we want to work with the square ones, first.

```
> plotnames <- unique(rotPlots$plot)
> getRange <- function(x){max(x)-min(x)}
> rangex <- aggregate(rotPlots$x, by=list(rotPlots$plot), FUN=getRange)
> rangey <- aggregate(rotPlots$y, by=list(rotPlots$plot), FUN=getRange)
> plotSizes <- data.frame(plot=rangex[,1], x=rangex[,2], y=rangey[,2], area=rangex[,2]*range
> plotSizes
```

```
        plot       x      y      area
1     bellow 141.100  73.76 10407.536
2   crackers 100.000 100.00 10000.000
3   distress  99.639  99.52  9916.073
4      gravy  99.769  99.74  9950.960
5     octane 199.000  50.00  9950.000
6    realtor  99.533  99.61  9914.482
7  reclusive 101.000 101.00 10201.000
8      rigid 224.000  50.00 11200.000
9     sodium  99.000  99.00  9801.000
```

```
10   trigger 124.000  75.00  9300.000
11   trinity  75.730 150.04 11362.529
```

Let's build an algorithm that will determine the density of trees per square meter and fill in the area with trees. The only things that will change is where we position the trees.

Trees will need to be separated by dbh and species within plot. Let's use crackers as an example, since it's first on the list.

```
> ##find center of map for our plot size
> plotSizes$centerx <- NA
> plotSizes$centery <- NA
> plotSizes[,c("centerx", "centery")] <- findPlotStart(300, 300, plotSizes$x, plotSizes$y)
> ## add those values to rotPlots by plot.
> ##
> ## for each plot...
>
> colnames(plotSizes) <- c("plot", "rangex", "rangey", "area", "centerx", "centery")
> newRots <- merge(plotSizes[,c(1, 4:6)], rotPlots, by=c("plot"))
> newRots$x <- newRots$centerx+newRots$x
> newRots$y <- newRots$centery + newRots$y
> head(newRots)

    plot     area centerx centery     X subplot treeid species ingrowth firstrec deathyear
1 bellow 10407.54   79.45  113.12 25214       8   7146    PIMO     2000     2000        NA 1
2 bellow 10407.54   79.45  113.12 25314       8   7146    PIMO     2000     2000        NA 1
3 bellow 10407.54   79.45  113.12 25414       8   7147    PIMO     2000     2000        NA 1
4 bellow 10407.54   79.45  113.12 25514       8   7147    PIMO     2000     2000        NA 1
5 bellow 10407.54   79.45  113.12 25614       8   7147    PIMO     2000     2000        NA 1
6 bellow 10407.54   79.45  113.12 25714       8   7148    PIMO     2000     2000        NA 1
     stage
1     tree
2     tree
3 seedling
4     tree
5     tree
6 seedling

> tail(newRots)

         plot     area centerx centery     X subplot treeid species ingrowth firstrec deathy
51211 trinity 11362.53 112.135   74.98 57631       8   3333    PIPO       NA     1996
51212 trinity 11362.53 112.135   74.98 57641       8   3333    PIPO       NA     1996
51213 trinity 11362.53 112.135   74.98 57651       8   3333    PIPO       NA     1996
51214 trinity 11362.53 112.135   74.98 57661       8   3333    PIPO       NA     1996
51215 trinity 11362.53 112.135   74.98 57671       8   3334    PIPO       NA     1996
51216 trinity 11362.53 112.135   74.98 57681       8   3334    PIPO       NA     1996
```

```
      dbh    stage
51211 1.5     tree
51212 2.7     tree
51213 3.4     tree
51214 4.8     tree
51215  NA seedling
51216 0.9     tree
```

Okay, so we have our new positions in plot space, for 300m x 300m plots in SORTIE. Now we just need to populate the spaces around them. Since our endgame is to build 6-column tables ready for SORTIE, let's go ahead and create a new data.frame, starting that process. That way, we're not working in and screwing up rotPlots, which has its place.

```
> ## We need to remove any records beyond the first years for our plots.
>
> keepme <- vector()
> ##for each plot...
> for(i in 1:length(plotnames)){
+   minyear <- NULL
+   minyear <- min(newRots[newRots$plot==plotnames[i], "measyear"])
+   keepme <- c(keepme,
+               which(newRots$plot==plotnames[i] &
+                       newRots$measyear==minyear &
+                       newRots$firstrec==minyear))
+
+ }
> #X, Y, Species, Type, Diam, Height
>
> sortieTrees <- data.frame(X=newRots[keepme, "x"],
+                           Y=newRots[keepme, "y"],
+                           Species=newRots[keepme, "species"],
+                           Type=newRots[keepme, "stage"],
+                           Diam=newRots[keepme, "dbh"],
+                           Height=0,
+                           Plot=newRots[keepme, "plot"],
+                           stringsAsFactors=F)
> sortieTrees[sortieTrees$Type=="tree", "Type"] <- "Adult"
> sortieTrees[sortieTrees$Type=="seedling", "Type"] <- "Seedling"
> sortieTrees[sortieTrees$Type=="Seedling", "Diam"] <- 1
> ##arbitrary, because seedlings in USGS plots don't have a dbh or db10
> ##measurement.
>
> head(sortieTrees)

    X   Y Species  Type Diam Height     Plot
1 113 174    CADE Adult 27.0      0 crackers
```

```
2 114 181    CADE Adult 11.7      0 crackers
3 114 182    CADE Adult 20.8      0 crackers
4 115 182    CADE Adult 23.1      0 crackers
5 114 184    CADE Adult  6.7      0 crackers
6 115 184    CADE Adult  4.8      0 crackers
```

```
> tail(sortieTrees)
```

```
         X     Y Species  Type Diam Height  Plot
9601 177.1 125.5    CADE Adult 16.6      0 rigid
9602 256.8 128.2    ABCO Adult 31.2      0 rigid
9603 260.6 127.7    ABCO Adult  2.6      0 rigid
9604 260.9 127.1    ABCO Adult 20.4      0 rigid
9605 254.1 128.7    ABCO Adult 44.5      0 rigid
9606 260.3 127.6    ABCO Adult  3.1      0 rigid
```

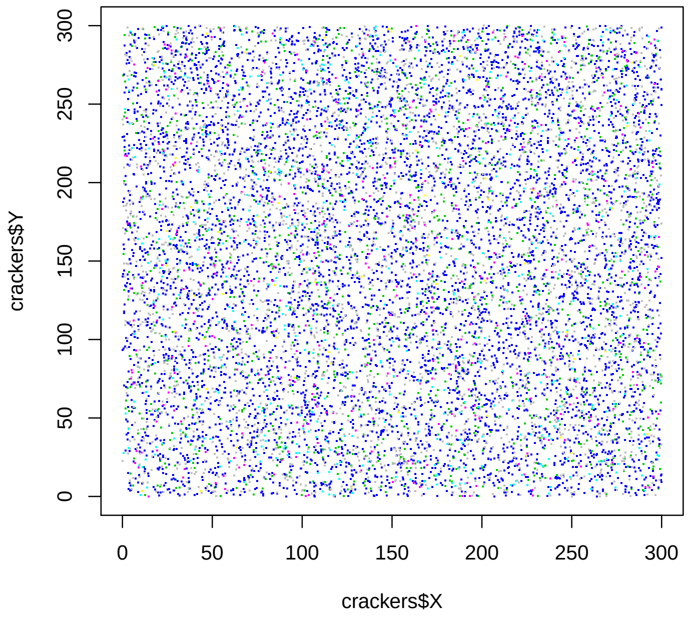
We will need to export by plot, but we can do that last. Ok, so we have our points, and they should be positioned correctly within the 300m x 300 m plot space. Now, we just need to determine the densities and distributions of trees in each plot, then fill the empty spaces with trees. First, let's get a data.frame started of the unique plot+species combinations. Then we have a looping function to work with. Let's use that to calculate individuals per square meter.

While we're at it, we'll also get the mean and standard deviation of log(dbh). We're doing log(dbh) because the data look relatively normal under log transformation. Also, old ecology habits die hard.

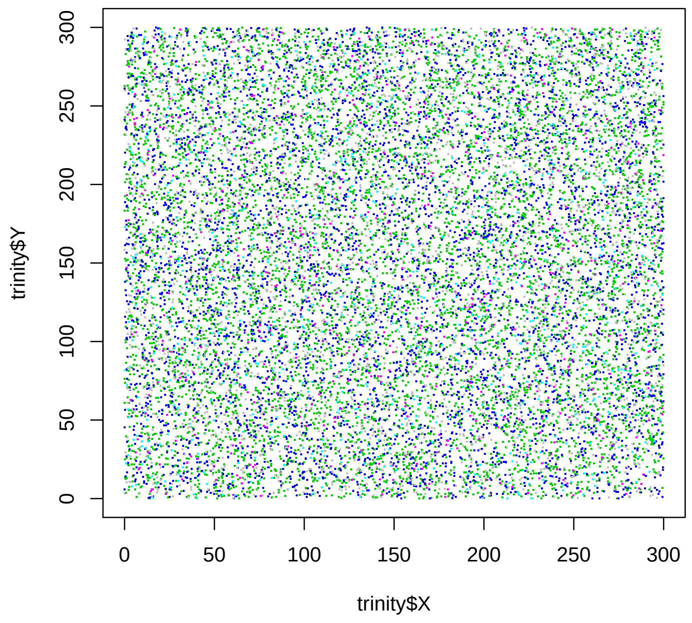
```
> sortiePlotTreeNum <- aggregate(sortieTrees$Diam, by=list(sortieTrees$Plot, sortieTrees$Spe
> colnames(sortiePlotTreeNum) <- c("Plot", "Species", "Count")
> colnames(plotSizes)[1] <- c("Plot")
> sortiePlotTreeNum <- merge(sortiePlotTreeNum, plotSizes[,c(1,4)], by=c("Plot"))
> sortiePlotTreeNum$density <- sortiePlotTreeNum$Count / sortiePlotTreeNum$area
> sortieTreesDBH <- aggregate(log(sortieTrees$Diam), by=list(sortieTrees$Plot, sortieTrees$S
> sortieTreesDBH$sddbh <- aggregate(log(sortieTrees$Diam), by=list(sortieTrees$Plot, sortieT
> sortieTreesDBH$counts <- aggregate(sortieTrees$Diam, by=list(sortieTrees$Plot, sortieTrees
> sortieTreesDBH[is.na(sortieTreesDBH$sddbh),"sddbh"] <- 0
> colnames(sortieTreesDBH) <- c("Plot", "Species", "meandbh", "sddbh", "n")
> sortieTreesFinal <- merge(sortieTreesDBH, sortiePlotTreeNum[,c("Plot", "Species", "density
> sortieTreesFinal$giantn <- floor(sortieTreesFinal$density * (300*300))
> response <- data.frame()
> for(i in 1:nrow(sortieTreesFinal)){
+   newresponse <- data.frame(Plot=sortieTreesFinal[i, "Plot"],
+                             Species=sortieTreesFinal[i, "Species"],
+                             X = runif(sortieTreesFinal[i, "giantn"], 0, 300),
+                             Y = runif(sortieTreesFinal[i, "giantn"], 0, 300),
+                             Type = "Adult",
+                             Diam = rlnorm(sortieTreesFinal[i, "giantn"],
+                                              sortieTreesFinal[i, "meandbh"],
+                                              sortieTreesFinal[i, "sddbh"]))
+   response <- rbind(response, newresponse)
+ }
> ##reorder response
> response <- response[,c("X", "Y", "Species", "Type", "Diam", "Plot")]
> response$Height <- 0
```

Okay, so we have transformed means and sd's ready, now we just need to write a function to find the missing spaces around the bounds of the filled in plot, and fill those in with trees. It may be easiest to generate for the whole 9 hectare area, then just delete any trees that fall within our already populated range.

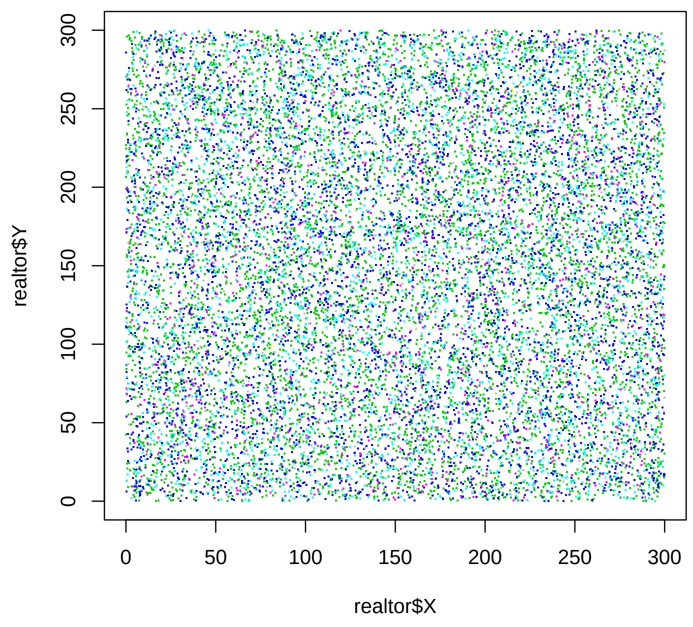
```
> crackers <- response[response$Plot=="crackers",]
> plot(crackers$X,
+      crackers$Y,
+      col=as.factor(crackers$Species),
+      pch=".")
```



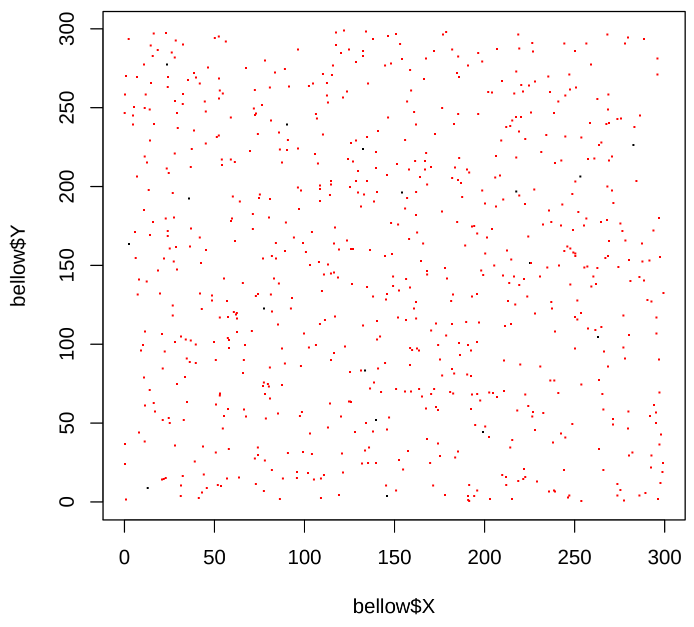
```
> trinity <- response[response$Plot=="trinity",]
> plot(trinity$X,
+      trinity$Y,
+       col=as.factor(trinity$Species),
+       pch=".")
```



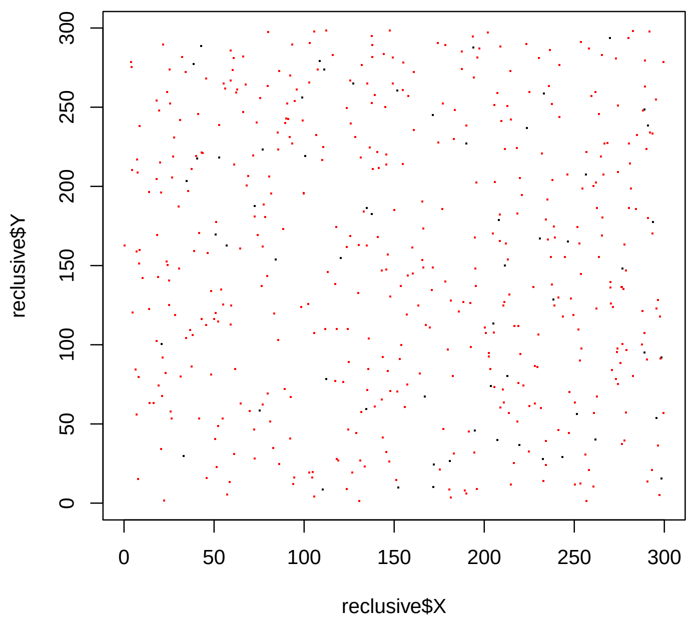
```
> realtor <- response[response$Plot=="realtor",]
> plot(realtor$X,
+       realtor$Y,
+       col=as.factor(realtor$Species),
+       pch=".")
```



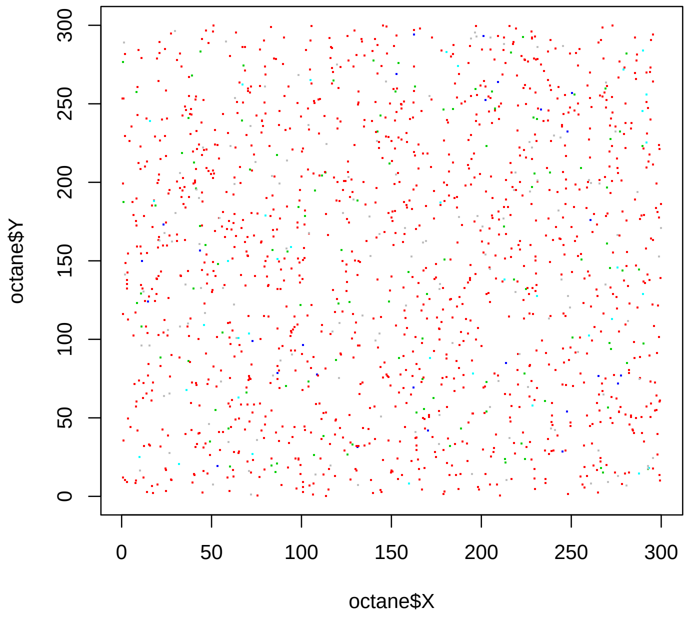
```
> bellow <- response[response$Plot=="bellow",]
> plot(bellow$X,
+       bellow$Y,
+       col=as.factor(bellow$Species),
+       pch=".")
```



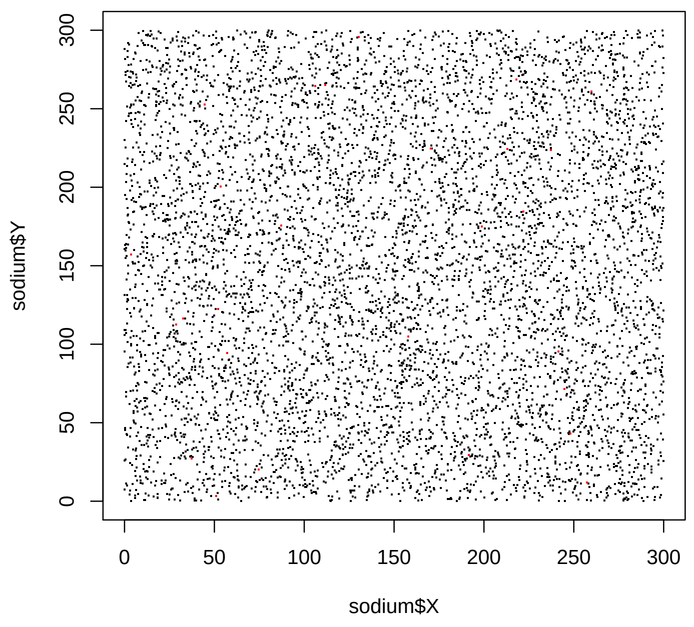
```
> reclusive <- response[response$Plot=="reclusive",]
> plot(reclusive$X,
+       reclusive$Y,
+       col=as.factor(reclusive$Species),
+       pch=".")
```



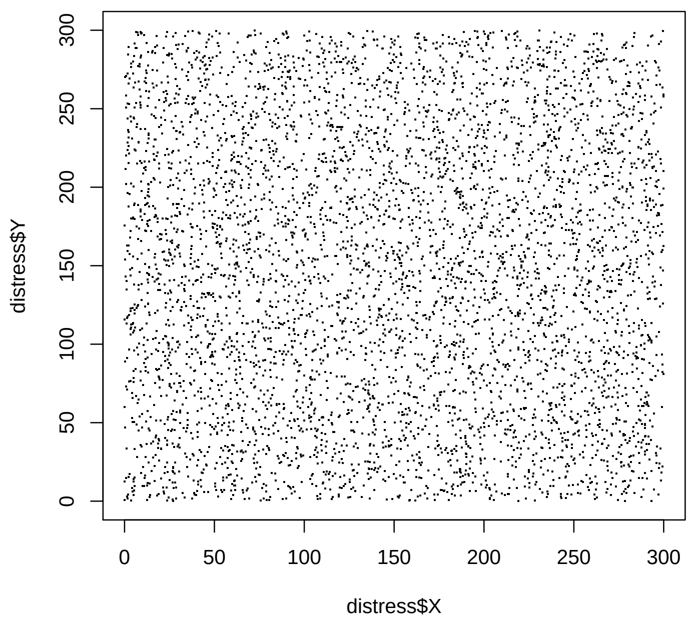
```
> octane <- response[response$Plot=="octane",]
> plot(octane$X,
+      octane$Y,
+      col=as.factor(octane$Species),
+      pch=".")
```



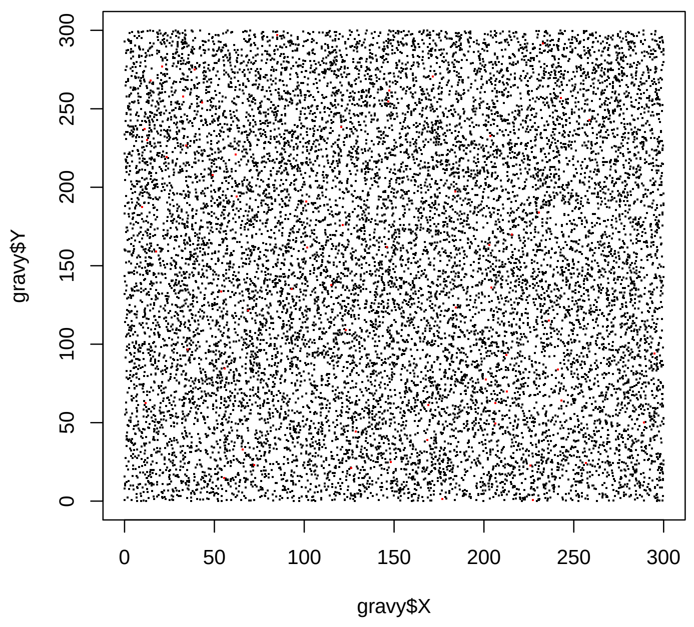
```
> sodium <- response[response$Plot=="sodium",]
> plot(sodium$X,
+      sodium$Y,
+      col=as.factor(sodium$Species),
+      pch=".")
```



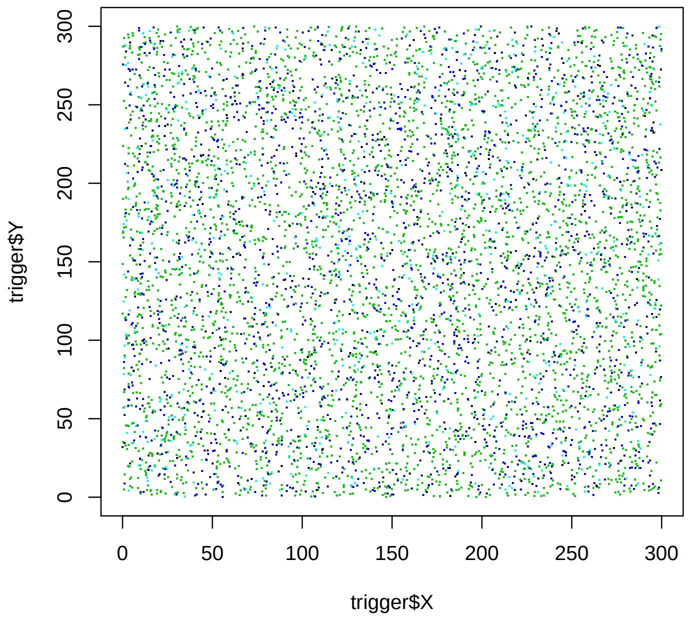
```
> distress <- response[response$Plot=="distress",]
> plot(distress$X,
+       distress$Y,
+       col=as.factor(distress$Species),
+       pch=".")
```



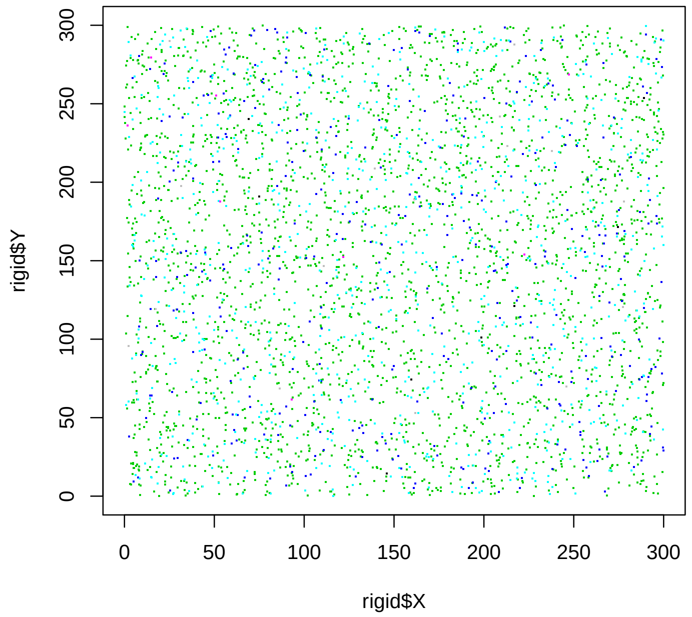
```
> gravy <- response[response$Plot=="gravy",]
> plot(gravy$X,
+      gravy$Y,
+      col=as.factor(gravy$Species),
+      pch=".")
```



```
> trigger <- response[response$Plot=="trigger",]
> plot(trigger$X,
+      trigger$Y,
+      col=as.factor(trigger$Species),
+      pch=".")
```



```
> rigid <- response[response$Plot=="rigid",]
> plot(rigid$X,
+       rigid$Y,
+       col=as.factor(rigid$Species),
+       pch=".")
```



As a final bit, we can write to tab delimited files after removing the "plot" column.

```
>  write.table(crackers[,-c(6:7)], file="crackers.txt", sep="\t", row.names=FALSE, quote=F)
>  write.table(trinity[,-c(6:7)], file="trinity.txt", sep="\t", row.names=FALSE, quote=F)
>  write.table(realtor[,-c(6:7)], file="realtor.txt", sep="\t", row.names=FALSE, quote=F)
>  write.table(bellow[,-c(6:7)], file="bellow.txt", sep="\t", row.names=FALSE, quote=F)
>  write.table(reclusive[,-c(6:7)], file="reclusive.txt", sep="\t", row.names=FALSE, quote=
>  write.table(octane[,-c(6:7)], file="octane.txt", sep="\t", row.names=FALSE, quote=F)
>  write.table(sodium[,-c(6:7)], file="sodium.txt", sep="\t", row.names=FALSE, quote=F)
>  write.table(distress[,-c(6:7)], file="distress.txt", sep="\t", row.names=FALSE, quote=F)
>  write.table(gravy[,-c(6:7)], file="gravy.txt", sep="\t", row.names=FALSE, quote=F)
>  write.table(trigger[,-c(6:7)], file="trigger.txt", sep="\t", row.names=FALSE, quote=F)
>  write.table(rigid[,-c(6:7)], file="rigid.txt", sep="\t", row.names=FALSE, quote=F)
```