```r
## http://arxiv.org/pdf/0709.3604v3.pdf
## http://mae.ucdavis.edu/dsouza/bml.html

## http://www.personality-project.org/R/makingpackages.html

## Set.seed() before applying functions!!!!

## Blue up, t = 1,3,... (Move first!)
## Red right, t = 2,4,...
setwd("/Users/shuhualiang/Documents/Davis MS/STA 242")

options(error = recover)
library(grDevices)

## Move Up (blue):
####################################################################################
## Generate the number of blue and red cars, then to them in a plane:
gen.cars = function(hor.grids, ver.grids, rho){
    total.grids = hor.grids*ver.grids
    red <- blue <- rho/2
    gen = sample(0:2, size=total.grids, replace=TRUE, prob=c((1-rho), red, blue))
    plane = matrix(gen, nrow=ver.grids, ncol=hor.grids)
}

#class(x) = "BML"
plot.BML = function(x,...){
    image(x, col=c("white","red","blue"),axes=FALSE)
}
    ## try: gen.cars(120,170,.3)
    ## 2 -> Blue, 1 -> Red

## Adjust directions, so matrix follows the direction of the plane:
turn.mat.90left = function(gen.cars){
    apply(gen.cars,1,rev)
}

turn.back.90right = function(mat){
    list.back = tapply(mat,rep(1:ncol(mat),each=nrow(mat)),function(i)i)
    revert = lapply(list.back,rev)
    matrix(unlist(revert),byrow = TRUE,nrow=length(revert))
}
## For partial vectors, move up one and make the last one zero:
one.up = function(vec){
    if(length(vec) <= 1) return(vec)
    else c(vec[2:length(vec)],0)
}

## Use when have a partial vector that comes after a 1:
two.zero.and.more = function(part){
    if(all(part==2) || length(part) <= 1) return(part)
    else {
        first.zero = min(which(part == 0))
        part[first.zero:length(part)] = one.up(part[first.zero:length(part)])
        return(part)
    }
}

## Rotate the entire vector by one element up:
rot <- function(x) (1:x %% x) +1
rotvec <- function(vec){vec[rot(length(vec))]}

## Operate the sequence in between two ones:
run.two = function(vec, ones){
    for(i in 1:(length(ones)-1)){
        if(0 %in% vec[(ones[i]):(ones[i+1])]){
            vec[(ones[i]+1):(ones[i+1]-1)] = two.zero.and.more(vec[(ones[i]+1):(ones[i
 +1]-1)])}
    }
    return(vec)
}

### Move up function when there is more than one 1:
move.up.more1 <- function(vec){
    ones = which(vec==1)
    if(vec[1]==1 && vec[length(vec)]==1){
        vec = run.two(vec,ones)
        return(vec)
    }
    if(vec[1]==1 && vec[length(vec)]!=1){
        vec[(ones[length(ones)]+1):(length(vec))] =
 two.zero.and.more(vec[(ones[length(ones)]+1):(length(vec))])
        vec = run.two(vec,ones)
        return(vec)
    }
    if(vec[1]!=1 && vec[length(vec)]==1){
        vec[1:(ones[1]-1)] = two.zero.and.more(vec[1:(ones[1]-1)])
        vec = run.two(vec,ones)
        return(vec)
    }
    if(vec[1]==2 && vec[length(vec)]!=1){
        vec[(ones[length(ones)]+1):(length(vec))] =
 two.zero.and.more(vec[(ones[length(ones)]+1):(length(vec))])
        if(vec[length(vec)]==0){
        vec[1]=0
        vec[1:(ones[1]-1)] = two.zero.and.more(vec[1:(ones[1]-1)])
        vec[length(vec)]=2}
        vec = run.two(vec,ones)
        return(vec)
    }
    else{   #(vec[1]!=1 && vec[length(vec)]!=1)
        vec[(ones[length(ones)]+1):(length(vec))] =
 two.zero.and.more(vec[(ones[length(ones)]+1):(length(vec))])
        vec[1:(ones[1]-1)] = two.zero.and.more(vec[1:(ones[1]-1)])
        vec = run.two(vec,ones)
        return(vec)
    }
}

## More a vector up 1 if there is only one 1 in the vector:
move.up.one1 = function(vec){
    one = which(vec==1)
    if(one==1){
        vec[2:length(vec)] = two.zero.and.more(vec[2:length(vec)])
        return(vec)}
    if(one==length(vec)){
        vec[1:(length(vec)-1)] = two.zero.and.more(vec[1:(length(vec)-1)])
        return(vec)}
    if(one!=1 && one!=length(vec)){
        if(vec[1]==2){
            vec[1]=0
            vec[1:(one-1)] = two.zero.and.more(vec[1:(one-1)])
            vec[(one+1):length(vec)] = two.zero.and.more(vec[(one+1):length(vec)])
            if(vec[length(vec)]==0) vec[length(vec)]=2
        return(vec)
        }
        else{
        vec[1:(one-1)] = two.zero.and.more(vec[1:(one-1)])
        vec[(one+1):length(vec)] = two.zero.and.more(vec[(one+1):length(vec)])
        return(vec)
        }
    }
}

## The move up function! Given a vector, the cars move up by 1:
move.up = function(vec){
    ones = which(vec == 1)
    if(length(ones)==0){
        vec = rotvec(vec)
        return(vec)
    }
    if(length(ones) == 1){move.up.one1(vec)}
    else{move.up.more1(vec)}
}

## Input a matrix and move blues up one:
oneUp.matrix = function(mat){
    li = tapply(mat,rep(1:ncol(mat),each=nrow(mat)),function(i)i)
    sapply(li, move.up)
}

####################################################################################

## Move Right (Red):
####################################################################################
## Switch numbers:
swap = function(mat){
    mat[which(mat==1)] <- 3
    mat[which(mat==2)] <- 1
    mat[which(mat==3)] <- 2
    return(mat)
}

## Turn to right direction as will be plotted:
Turn2 = function(mat){
    bat = turn.mat.90left(mat)
    cat = turn.mat.90left(bat)
    return(cat)
}

TurnBack2 = function(mat){
    bat = turn.back.90right(mat)
    cat = turn.back.90right(bat)
    return(cat)
}

###################################################################################
## Final functions:

## Function that moves blue cars when given a matrix:
Blue = function(matrix){
    rightDirection = turn.mat.90left(matrix);rightDirection
    moved.up = oneUp.matrix(rightDirection)
    rotBack = turn.back.90right(moved.up)
    return(rotBack)
}

## Function that moves red cars when given a matrix:
Red = function(matrix){
    changed.num.ratBack = swap(matrix);changed.num.ratBack
    Ready2Up = Turn2(changed.num.ratBack);Ready2Up
    Moved.right = oneUp.matrix(Ready2Up);Moved.right
    Origin.Dir = TurnBack2(Moved.right); Origin.Dir
    ready2plot = swap(Origin.Dir)
    return(ready2plot)
}

## Function that takes in time, and alternates between blue and red car movements:
Drive = function(time,hgrid, vgrid, rho,...){
    carPlane = gen.cars(hgrid, vgrid, rho)
    #mat = carPlane
    class(carPlane) = "BML"
    png(file = "Anna0")
    print(plot(carPlane))
    dev.off()

    Annas = rep("Anna",time)
    Annas = paste(Annas, 1:time, sep="")
    moved = c()
    unmoved = c()
    for(i in 1:time){
        if(i %% 2 !=0){
            plane = Blue(carPlane)
            moved[i] = length(which((carPlane == plane)==FALSE))
            unmoved[i] = hgrid*vgrid*rho - moved[i]
            carPlane = plane
            class(carPlane) = "BML"
            png(Annas[i])
            print(plot(carPlane))
            dev.off()
        }
        if(i %% 2 ==0){
            plane = Red(carPlane)
            moved[i] = length(which((carPlane == plane)==FALSE))
            unmoved[i] = hgrid*vgrid*rho - moved[i]
            carPlane = plane
            class(carPlane) = "BML"
            png(Annas[i])
            print(plot(carPlane))
            dev.off()
        }
    }
    par(mfrow=c(1,2))
    (plot(seq(time),moved,type="l",xlab = "Time",ylab="Number of Car Moves",main="Number
 of Car Moves at Each Time Point"))
    (plot(seq(time),unmoved,type="l",xlab = "Time",ylab="Number of Car
 Unmoves",main="Number of Car Unmoves at Each Time Point"))
}

## Functions that creates a GIF and cleans up the working directory:
video = list.files(pattern="Anna");video
new = paste(video,".png",sep="");new
Rename = file.rename(video, new)
GIF = system("convert -delay 1 *.png Moves.gif")

Remove = file.remove(new)

#####################################################################################
## Creating the package:
mylist = c(
"gen.cars",
"plot.BML",
"turn.mat.90left",
"turn.back.90right",
"one.up",
"two.zero.and.more",
"rot",
"rotvec",
"run.two",
"move.up.more1",
"move.up.one1",
"move.up",
"oneUp.matrix",
"swap",
"Turn2",
"TurnBack2",
"Blue",
"Red",
"Drive",
"video",
"new",
"Rename",
"GIF",
"Remove")

package.skeleton("BMLpkg", mylist, environment = .GlobalEnv, path = ".", force = TRUE)
```