Appendix: Three Approaches

The first approach

```
creatBMLGrid=
  #
  #This function creat a grid and stored in a matrix. The red and blue
cars which have specific
  #number are randomly located. And define the class of grid as bot mat
rix and BMLGrid.
  #The parameter:
  #r is the number of row of the grid;
  #c is the number of column of the grid;
  #ncars is a vector including the number of red and blue cars separate
ly;
  #p is the density of the cars. And if we have p, we will ignore the v
alue of given ncars;
  #
 function(r = 100, c = 99, ncars = c(blue = 100, red = 100), p = 0)
  {
    if(p != 0){
      blue = red = ceiling(r*c*p/2)
      ncars = c(blue, red)
    }
    if (p<0|p>0.9) stop("Density should be positive and not greater tha
n 0.9")
    r = ceiling(r)
    c = ceiling(c)
    if(c<=0|r<=0) stop("Dimensions should be positive")


    grid = matrix(0,r,c)
    pos = sample(r*c, sum(ncars))

    grid[pos] = sample(rep(c(1,2),times = ncars))
    class(grid) = c("BMLGrid",class(grid))
    grid
  }


nextPos=
  #
  #This function is to find the next position of one given car.
  #The parameter:
  #pos is the current position of a specific car we want to get its nex
t position;
  #car has 1-2 values:1 represent for blue cars and 2 represent for red
```

```
 one,
  #when the car is blue one, it moves vertically upward,
  #Used mod to decide whether the car is at the edge of the grid,
  #when the car is red one, it moves horizontally rightward,
  #Used aliquot part to decide whether the car is at the edge of the gr
id;
  #grid is a matrix and BMLGrid.
  #
  function(pos, car,grid){
  if(car == 1){
    if(pos%%nrow(grid) == 1)
      pos = pos + nrow(grid) - 1
    else pos = pos - 1
  }
  else{
    if(pos-nrow(grid)*(ncol(grid)-1)>0)
      pos = pos-nrow(grid)*(ncol(grid)-1)
    else pos = pos + nrow(grid)
  }
  pos
}


moveCar.a=
  #
  #This is a function to move one car one time. If the next position of
 this car is empty,
  #we will return this new position. Otherwise we will let the next pos
ition equal to current
  #one.
  #The parameter:
  #posiont is the current location of the car in grid;
  #car has 1-2 values imply the two kind of cars;
  #grid is a matrix and BMLGrid;
  #
  function(pos,car,grid){
    pos.new = nextPos(pos,car,grid)
    if(grid[pos.new] != 0)  pos.new = pos
    else pos.new = nextPos(pos,car,grid)
  }

carLoc =
  #
  #This function will move all one specific type of car one time by usi
ng sapply.
  #The parameter:
  #grid is a matrix and BMLGrid;
  #n implies which type of car will move this step. When n is odd, blue
 cars will move, when n
  #is even, we will move red ones;
```

```
  #
  function(grid,n){
    car = which(grid == (2-n%%2))
    car.nwpos = sapply(1:length(car), function(i) moveCar.a(car[i],(2-
n%%2),grid))
    grid[car] = 0
    grid[car.nwpos] = (2-n%%2)
    grid
  }

runBMLGrid=
    #
    #Function to move car several steps
    #numSteps is the number of steps
    #
  function(grid,numSteps=10000){
    for(i in 1:numSteps)
    grid = carLoc(grid,i)
    grid
  }
```

The second approach

```
creatBMLGrid=
  #
  #This function is the same as which in the first approach
  #
  function(r = 100, c = 99, ncars = c(blue = 100, red = 100), p = 0)
  {
    if(p != 0){
      blue = red = ceiling(r*c*p/2)
      ncars = c(blue, red)
    }
    if (p<0|p>0.9) stop("Density should be positive and not greater tha
n 0.9")
    r = ceiling(r)
    c = ceiling(c)
    if(c<=0|r<=0) stop("Dimensions should be positive")


    grid = matrix(0,r,c)
    pos = sample(r*c, sum(ncars))

    grid[pos] = sample(rep(c(1,2),times = ncars))
    class(grid) = c("BMLGrid",class(grid))
    grid
  }
```

```
getCarloc =
  #
  #This function will give us the locations of all the cars(ignore the
type) in the grid.
  #Not like 1th approach, this function will store the location in the
form (x,y) rather than
  #a index of an array.
  #The parameter grid is a matrix and BMLGrid.
  #Return is a dataset contain coordinate x, coordinate y, and correspo
nding type of car.
  #
  function(grid){
    x = row(grid)[grid != 0]
    y = col(grid)[grid != 0]
    pos = cbind(x,y)
    data.frame(x,y,car = grid[pos])
  }

getNextloc =
  #
  #This function uses vectorie to find the next position of all cars on
e time.
  #When we move blue cars("blue" cars move vertically upward), we can r
emain coordinate y
  #same, and miner coordinate x by 1L. If x-1L equal to 0, then this ca
r gets to the edge of
  #the grid, and wraps around.
  #When we move red cars("red" cars move horizontally rightwards), we c
an remain coordinate x
  #same, and plus coordinate y by 1L. If y+1L larger than the number of
 columns, then this car
  #gets to the edge of the grid, and wraps around.
  #The parameter:
  #pos is the positions of all cars;
  #time=n indicates this is the nth step to run, and implies which type
 of cars will move.
  #
  function(pos,grid,time=1){

    if(time%%2 == 1){
    indexBlue = which(pos$car == 1)
    pos[indexBlue, ]$x = pos[indexBlue, ]$x - 1L
    pos[indexBlue, ]$x[pos[indexBlue, ]$x == 0] = nrow(grid)
    }

    else {
    indexRed = which(pos$car == 2)
    pos[indexRed,]$y = pos[indexRed,]$y + 1L
```

```
    pos[indexRed,]$y[ pos[indexRed, ]$y > ncol(grid) ] = 1L
    }
    pos
  }

moveCar.b =
  #
  #This is going to move all one specific type of cars one time.
  #We can get the cars next loctions first, and check whehter these loc
tions are empty.
  #If the next loctions are empty, we will move the cars to them and le
t the current locations
  #of these cars to be empty. Otherwise, we will stay the cars.
  #time=n indicates this is the nth step to run, and implies which type
 of cars will move.
  #
  function(grid,time=1){
      pos = getCarloc(grid)
      pos.new = getNextloc(pos, grid, time)

      index =which( grid[ cbind(pos.new$x, pos.new$y) ] == 0 )
      newLoc = pos.new[index,]
      oriLoc = pos[index,]

      grid[cbind(newLoc$x,newLoc$y)] = 2-time%%2
      grid[cbind(oriLoc$x,oriLoc$y)] = 0
      grid
  }

runBMLGrid =
  #
  #This function will move cars "numSteps" steps.
  #
  function(grid, numSteps){
    for(i in 1:numSteps)
      grid = moveCar.b(grid,i)
    grid
  }
```

The thired approach

```
creatBMLGrid=
  #
  #This function is a liitle bit different from previous ones. We use 3
 to denote red car
  #instead of 2.
  #
  function(r = 100, c = 99, ncars = c(blue = 100, red = 100), p = 0)
  {
    if(p != 0){
```

```
      blue = red = ceiling(r*c*p/2)
      ncars = c(blue, red)
    }
    if (p<0|p>0.9) stop("Density should be positive and not greater tha
n 0.9")
    r = ceiling(r)
    c = ceiling(c)
    if(c<=0|r<=0) stop("Dimensions should be positive")


    grid = matrix(0,r,c)
    pos = sample(r*c, sum(ncars))

    grid[pos] = sample(rep(c(1,3),times = ncars))
    class(grid) = c("BMLGrid",class(grid))
    grid
  }




moveCar.c =
    #
    #This function is used to move car.
    #In this function, we create a new matrix based on the original one.
 This new matrix can
    #be considered as all elements in the matrix move upwards(rightward
s). Thus, by finding
    #the difference between two matrix, we can easily find which cars c
an move. When it's time
    #to move the blue cars, we move downwards the orignal grid(matrix)
to get the new one and
    #calculate a difference matrix. The locations in that matrix which
has value -1 are the
    #locations we should move the car. For red cars, we need to move le
ftwards the grid
    #and find locations which have value -3 in difference matrix. And t
hese are the locations
    #we should move the car.
    #time implies which type of car we need to move.
    #
    #
function(grid,time){
    if(time%%2 == 1){
      grid.new = rbind(grid[nrow(grid),], grid[1:(nrow(grid)-1),])

      findMov = which(grid.new - grid == -1 ) #this is the index of blu
e cars which can move
      grid[findMov] = 0
```

```
      rule = (findMov - 1L)%%nrow(grid) #rule to judge whether the cars
 get to the edge
      findMov[rule==0] =
        findMov[rule==0] + nrow(grid) - 1
      findMov[rule != 0] =
        findMov[rule != 0] - 1

      grid[findMov] = 1
    }
      else{
        grid.new = cbind(grid[,2:ncol(grid)],grid[,1])

        findMov = which(grid.new - grid == -3 )#this is the index of re
d cars which can move
        grid[findMov] = 0

        rule = findMov - (ncol(grid)-1)*nrow(grid) #rule to judge wheth
er the cars get to the edge
        findMov[rule>0] =
          findMov[rule>0] - (ncol(grid)-1)*nrow(grid)
        findMov[rule <= 0] =
          findMov[rule <= 0] + nrow(grid)

        grid[findMov] = 3
  }
  grid
}

runBMLGrid =
  #
  #move car numSteps times
  #
  function(grid, numSteps){
    for(i in 1:numSteps)
      grid = moveCar.c(grid,i)
    grid
  }
```

Plot and Summary function(S3 methods and classes)

```
plot.BMLGrid =

  function(grid, main){
    image(t(grid[nrow(grid):1,]),col=c("white","blue","red"),main = mai
n)
    box()
  }

numBlocked =
```

```
  #
  #This function is used to find which cars are blocked
  #we need function gerCarloc and getNextloc in 2nd approach
  #carType is 1-2 value indicate the type of the car.
  #
  function(grid, carType){
    carPos = getCarloc(grid)

    carnxtPos = subset(getNextloc(carPos, grid, carType), car == carTyp
e)
    numNotmov = length(
      which( grid[cbind(carnxtPos$x, carnxtPos$y)] != 0) )
    numNotmov
  }

carVelocity =
  #
  #This function calculate the cars velocity.
  #We define velocity by fomula:
  #the number of red(blue) cars which can move / the number of whole re
d(blue) cars
  #This function is for approaches one and two not for three!
  #
  function(grid, carType){
    totalCar = length(which(grid == carType))

    velocity = (totalCar - numBlocked(grid,carType)) / totalCar
    velocity
  }

carVelocity =
  #
  #This function calculate the cars velocity.
  #We define velocity by fomula:
  #the number of red(blue) cars which can move / the number of whole re
d(blue) cars
  #This function is ONLY for the third approach
  #
  function(grid,carType){
    if(carType == 1){
      grid.new = rbind(grid[nrow(grid),], grid[1:(nrow(grid)-1),])
      blueMov = length(which(grid.new - grid == -1 ))
      velocity = blueMov/length(which(grid==1))
    }
    else {
      grid.new = cbind(grid[,2:ncol(grid)],grid[,1])
      redMov = length(which(grid.new - grid == -3 ))
      velocity = redMov/length(which(grid==3))
    }
```

```
    velocity
  }

summary.BMLGrid =
  #
  #summary for class 'BMLGrid'
  #return:
  #The size of the grid, how many cars in the grid, how many red or blu
e cars there,
  #how many red or blue cars are blocked separately.
  function(grid){
    r = nrow(grid)
    c = ncol(grid)

    car.num = length(which(grid != 0))
    blue.num = length(which(grid == 1))
    red.num = length(which(grid == 2))

    notmovBlue = numBlocked(grid, 1)
    notmovRed = numBlocked(grid, 2)

    velocityBlue = round(carVelocity(grid, 1), 3)
    velocityRed = round(carVelocity(grid, 2), 3)

    cat(' This is a ',r,'*',c,'GRID:',
          '\n','There are total',car.num,'CARS:',
          '\n','                  BLUE              RED              ',
          '\n','NUM           ',blue.num,'               ',red.num,
          '\n','BLOCKED NUM   ',notmovBlue,'               ',notmovRed,
          '\n','VELOCITY      ',velocityBlue,'              ',velocityR
ed,'\n')
  }
```

Analysis for codes

```
setwd("e://2015 spring/242/assignment2/")
library(profr)
library(ggplot2)
library(reshape2)
library(proftools)

source('h2.R')
file = 'runBMLGile.out'
Rprof(file)
grid = creatBMLGrid(100,100,p=0.5)
grid = runBMLGrid(grid,100)
Rprof(NULL)
rprof1 = summaryRprof(file)
plot(parse_rprof(file),minlabel = 0.07)
```

```r
plotProfileCallGraph(readProfileData(file))



source('h2(2).R')
##using Rprof() to profile the R code
file = 'runBMLGile_b.out'
Rprof(file)
grid = creatBMLGrid(100,100,p=0.5)
grid = runBMLGrid(grid,100)
Rprof(NULL)
summaryRprof(file)
plot(parse_rprof(file),minlabel = 0.05,angle = 30)
plotProfileCallGraph(readProfileData(file),score = "total")


source('h2(3).r')
file = 'runBMLGile_c.out'
Rprof(file)
grid = creatBMLGrid(100,100,p=0.5)
grid = runBMLGrid(grid,10000)
Rprof(NULL)
summaryRprof(file)
par(mfrow = c(1,2))
plot(parse_rprof(file))
plotProfileCallGraph(readProfileData(file),score = "total")


#Calculate the time utilization for each three approches and plot them.
grid = creatBMLGrid(100,100,c(100,100))

source("h2.R")
time1 = integer(50)
aa = grid
for(i in 1:50){
  time1[i] = system.time({runBMLGrid(aa,50*i)})
}

source("h2(2).R")
time2 = integer(50)
aa = grid
for(i in 1:50){
  time2[i] = system.time({aa = runBMLGrid.b(aa,50*i)})
}

source("h2(3).r")
time3 = integer(50)
aa=grid
aa[which(aa=2)]=3
```

```
for(i in 1:50){
  time3[i] = system.time({aa = runBMLGrid.c(aa,50*i)})
}

plot(time1,xlim = c(0,50),ylim = c(0,10),type = 'l',xlab = 'STEPS/50',y
lab = 'TIME',
     main = 'Comparing for 3 approaches')
lines(time2,col = 'red')
lines(time3,col = 'blue')
legend("topleft",legend = c('Approach 1','Approach 2','Approach 3'),
       col=c("black","red","blue"),lty =1)
```

Analysis for stochastic process

```
####plot the density
plotVel =
  function(times,r,c,p){
    par(mfrow = c(1,2))
    grid = creatBMLGrid(r,c,p=p)

    blu.v = integer(times/20)
    red.v = integer(times/20)

    for(i in 1:(times/20)){
        grid = runBMLGrid(grid,20)
        blu.v[i] = carVelocity(grid,1)
        red.v[i] = carVelocity(grid,3)
  }
    plot(blu.v,ylim = c(0,1),type = "l",col ="blue",
         main = paste0(" Grid:",r,"*",c," p:",p," Times:",times),
         xlab="numSteps/20", ylab = "Velocity")
    lines(red.v,col = "red")
    grid[which(grid==3)]=2
    plot(grid,main = paste0(" Grid:",r,"*",c," p:",p," Times:",times))
}


###############################plot velocity aginsit number of steps an
d density of cars.
plotVel2 =
  function(times,r,c){
    p = seq(0.2,0.7,0.01)
    velocity = integer(length(p)*times)
    k=1
    for (j in p){
      grid = creatBMLGrid(r,c, p =j)
      for(i in 1:times){
        grid = moveCar.c(grid,i)
        velocity[k] = carVelocity(grid,times)
        k=k+1
```

```
      }
    }
    velocity = matrix(velocity,nrow = times)

    require(plot3D)
    persp3D(1:nrow(velocity),1:ncol(velocity),velocity,
            theta = 140, phi = 20 ,xlab = "Numsteps", ylab = "Density",
            zlab = "velocity",main = paste0("Grid:",r,"*",c," Numsteps:
",times," Rho:[0.2,0.7]"))
    return(velocity)
  }
```