How I made an agent-based model in R for the first time

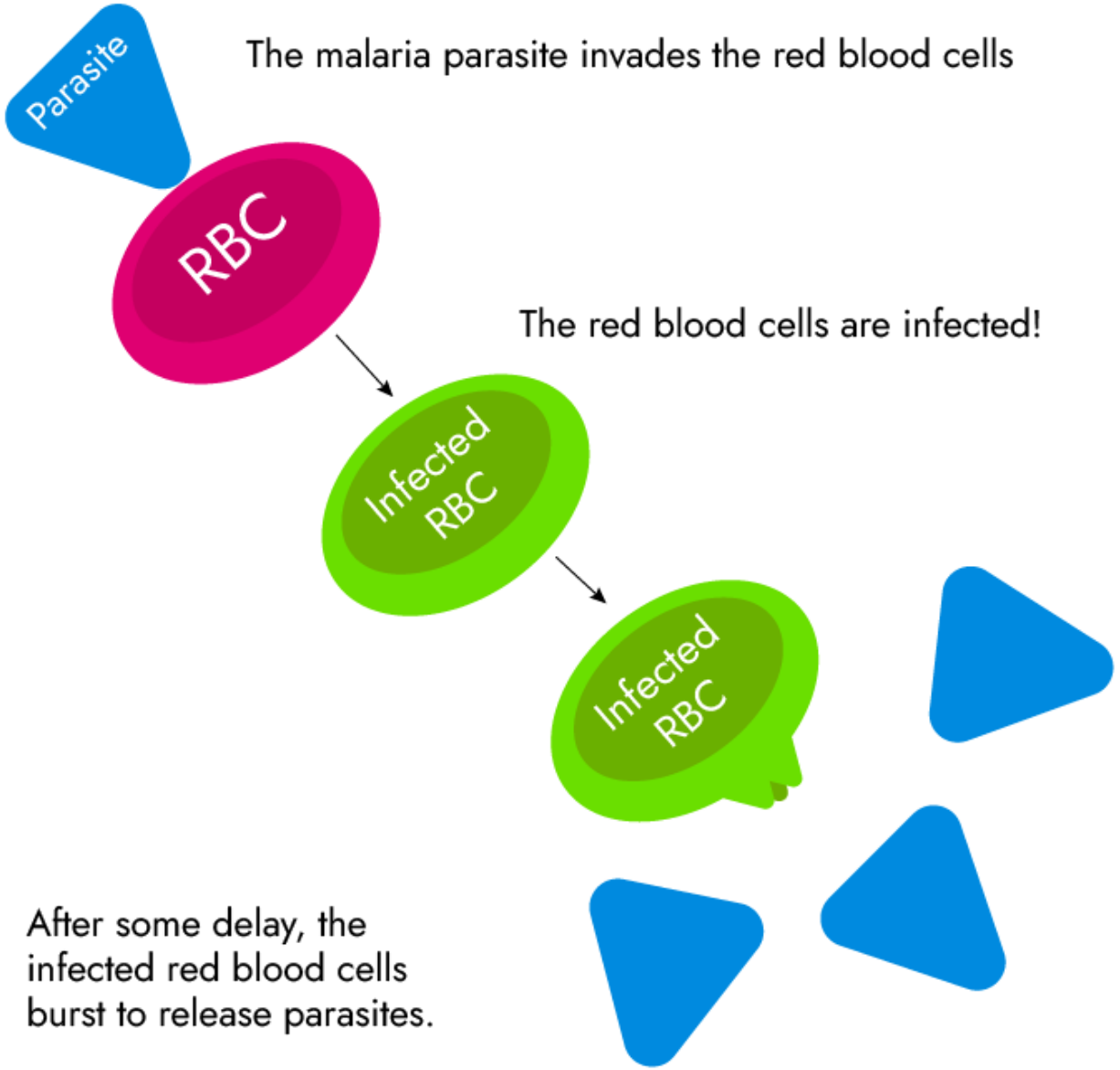
Inelegant and unhinged but you will laugh

Damie Pak
(dp495@cornell.edu)

DISCLAIMER:

1) Not the most efficient or best way
2) Is a side project/not a research project so have not been thoroughly error-proofed!
3) The subsetting method I'm doing is bound to lead to some errors- I'm trying to figure this out
4) The code is still messy!

The biological scenario: Within-host malaria infection

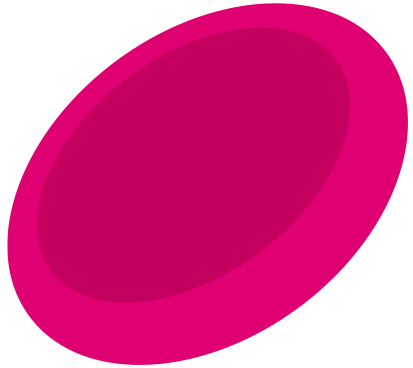


Malaria parasites invade red blood cells (RBCs) and the RBCs become infected.

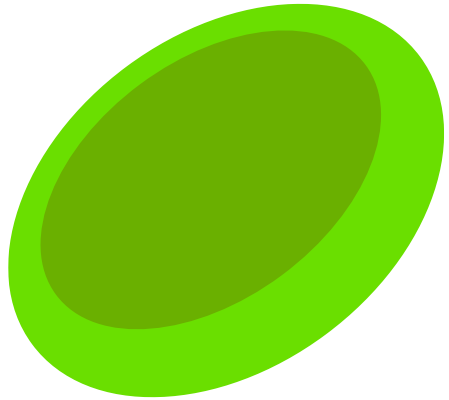
After some time-delay, the infected red blood cells release some number of parasites.

The agents in our model

Malaria parasite



Red blood cell



Infected
Red blood cell

In R, you must track your individuals!
One way is through a data.frame

"Features"

Individual ID	Type	X	Y	…	…
1	P	1	2		
2	P	3	4		
3	R	5	5		
4	R	6	7		

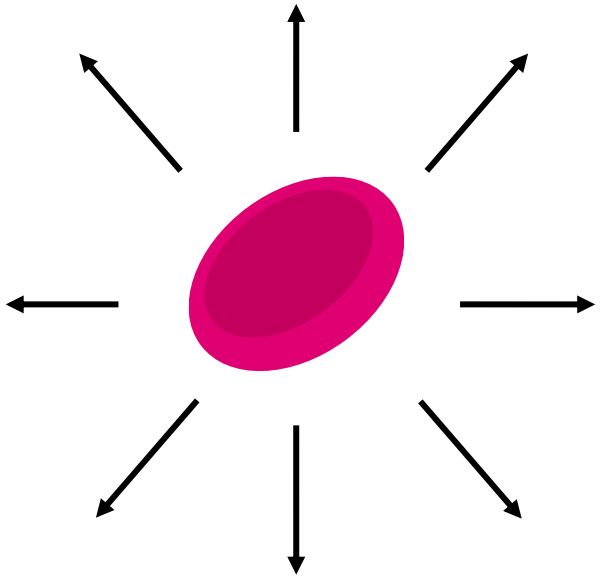
Each individuals has its own row

I call this dataframe Pop!

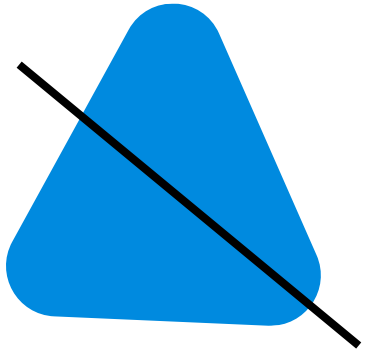
Individual: A unique ID number for each agent
Type: What kind of agent? (R for RBC, I for Infected RBC, and P for Parasite)
x : Current x-coordinate
y : Current y-coordinate
time_alive: How long have they been alive for
time_infected: How long they have been infected for
infected: Are they infected or not? (RBC/Infected RBC only)
dead: Are they dead or not?
infect: Have they infected or not (parasite only)
time: current time in the system

```
Pop<-
  data.frame(
    individuals = seq(1,200),
    type = c(rep("R",150), rep("P",50)),
    x = sample.int(100, 200,replace=TRUE),
    y = sample.int(100, 200,replace=TRUE),
    time_alive = rep(0,200),
    time_infected = rep(0,200),
    infected = rep("F",200),
    dead = rep("F",200),
    infect = rep("F",200),
    time = 0
  )
```

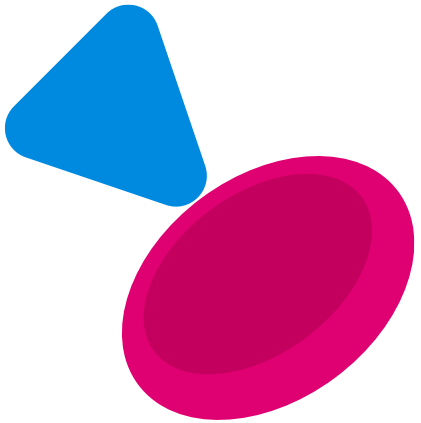
Processes that drive our model



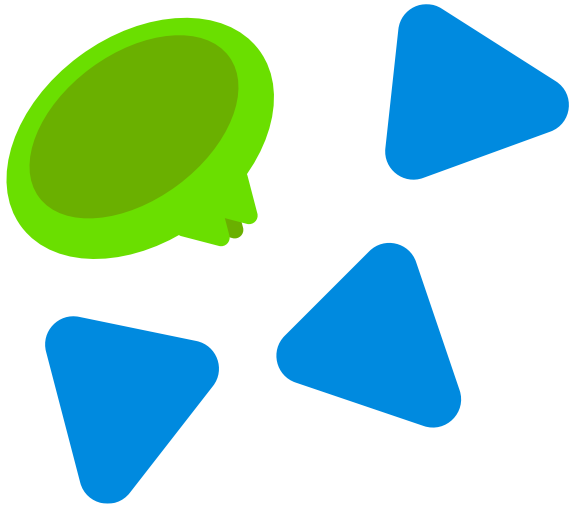
Movement
(All agents)



Background mortality
(Parasite)



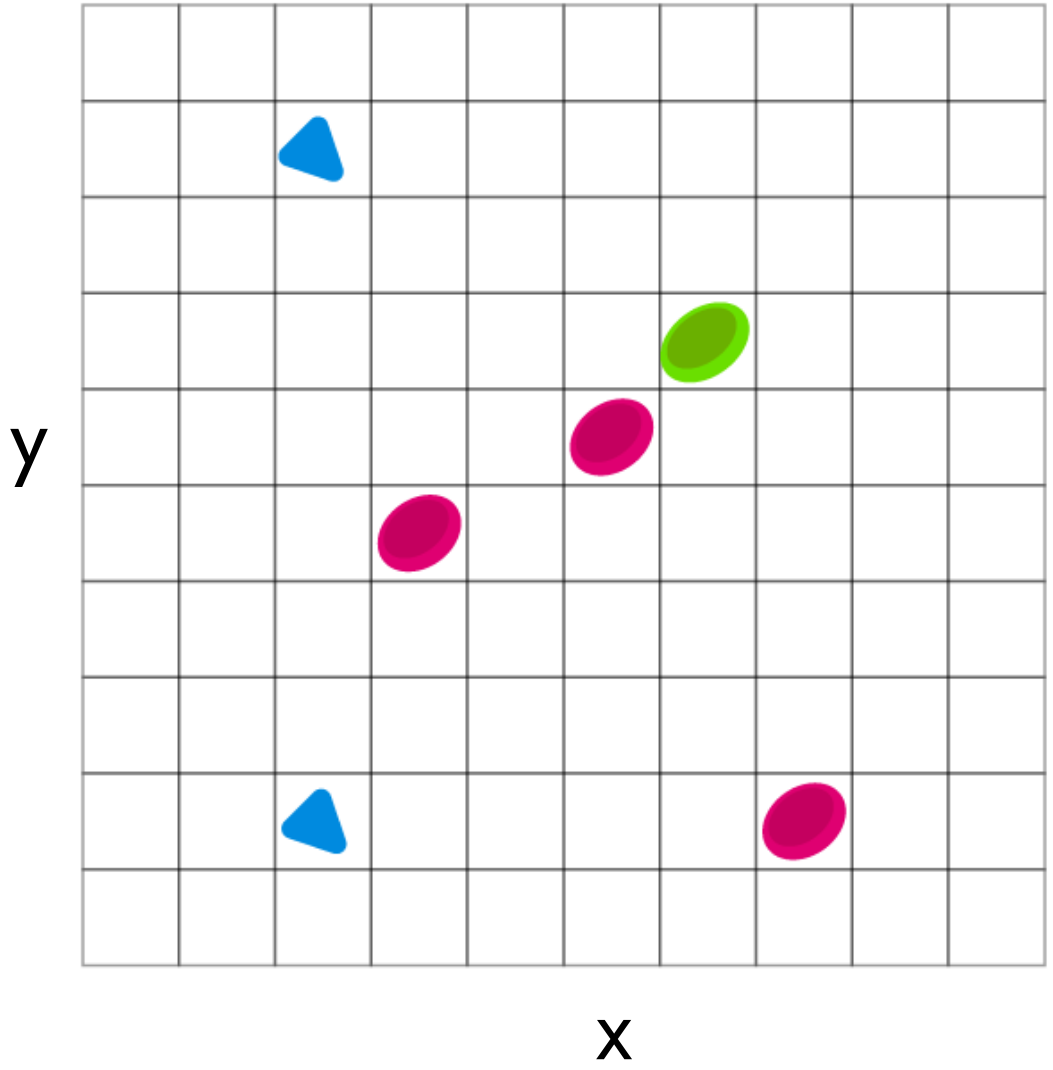
Infection
(Parasite and RBCs)



Bursting
(Parasites and infected RBCs)

Each one is a function()

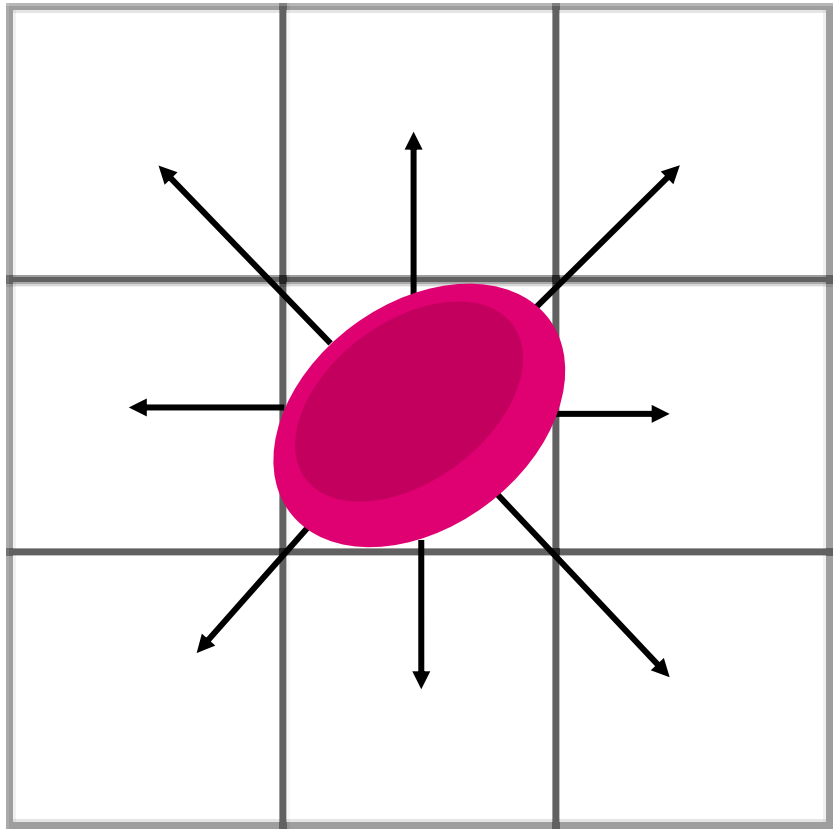
Getting everyone to move around



Imagine that the agents are moving around on a grid system.

We know where everyone is with the x-y coordinate

Let’s imagine that the agent moves randomly



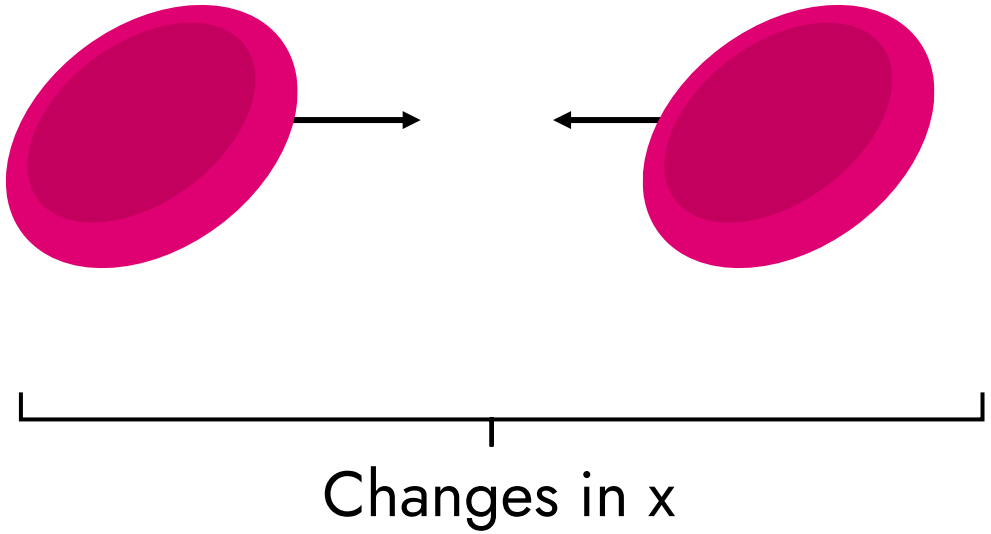
An agent usually have eight ways of moving!

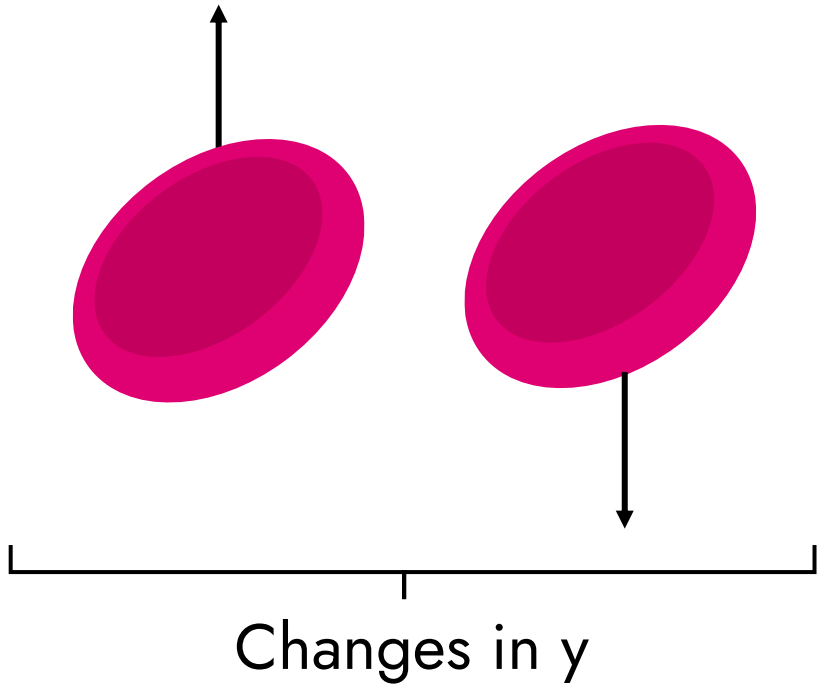
You can get an agent to move by taking the original coordinates and adding or subtracting some values.

We’re assuming that the agent can move only one square at a time and it’s entirely random!

Remember we have the x and y coordinate

Let's say the new movement can either be (-1,0,+1)



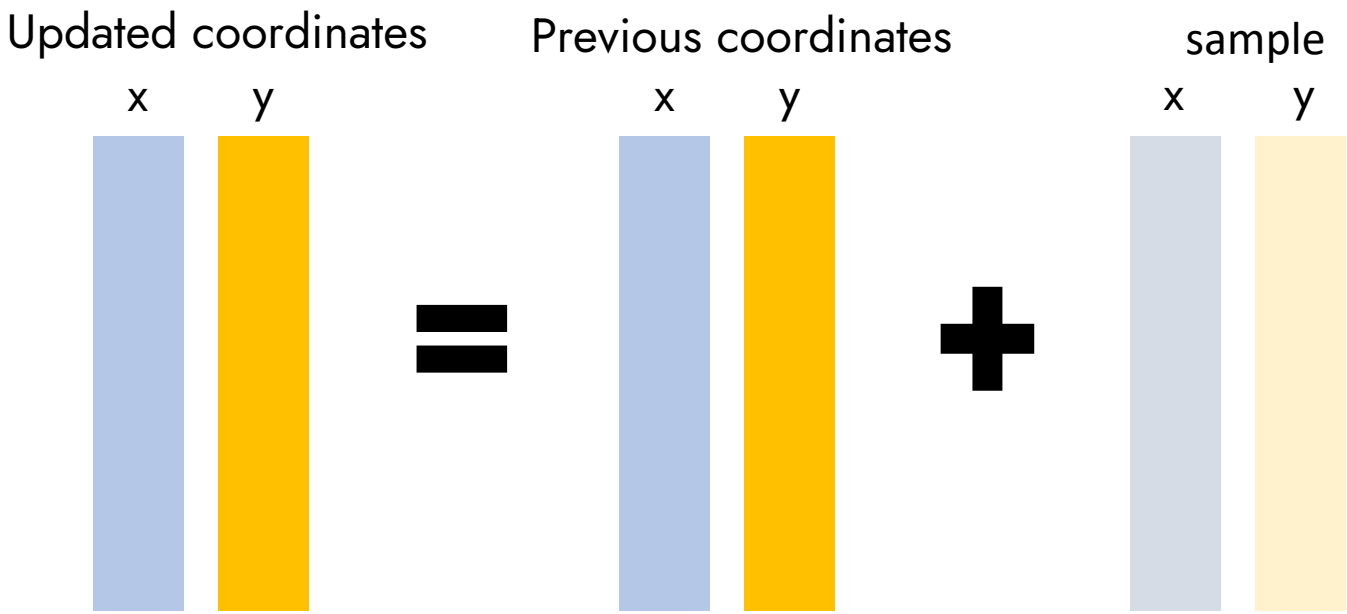


How to randomize movement?

```
sample(x = c(-1, 0, 1), size = nrow(Pop),replace=TRUE)
```

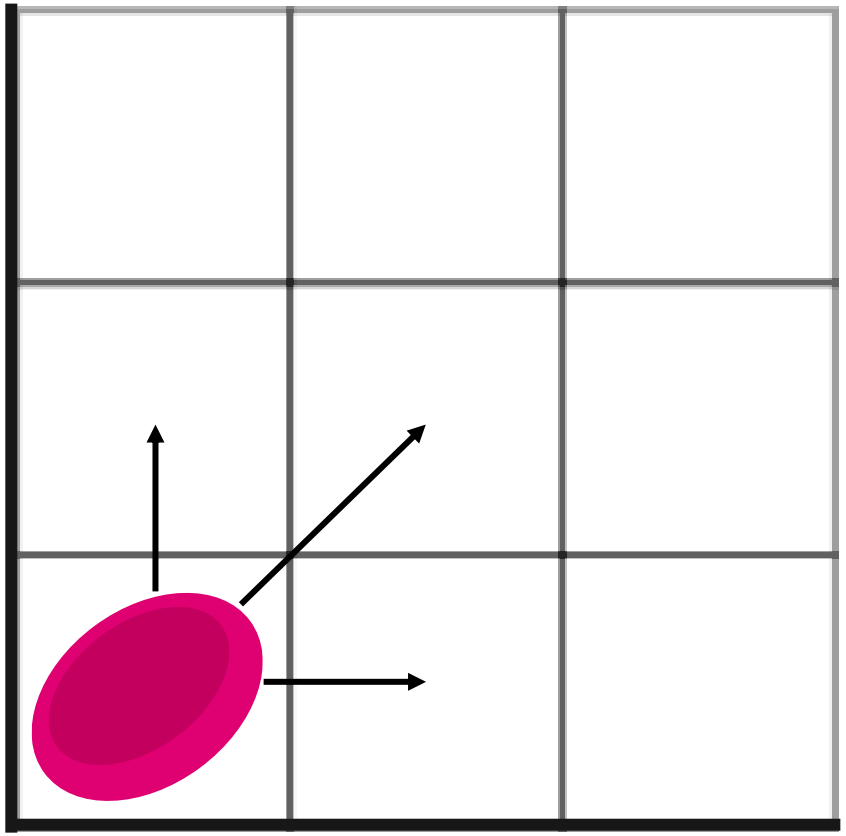
Randomly choose -1, 0, or 1.
The size tells us how many times to do this
Replace means that the options are the same for all individuals

Take the original x,y coordinates of all individuals and add the new vectors to both the x column and y column



```r
move_x ← sample(x = c(-1, 0, 1), size = nrow(Pop),replace=TRUE)
move_y ← sample(x = c(-1, 0, 1), size = nrow(Pop),replace=TRUE)

Pop[,'x'] ← Pop[,'x'] + move_x
Pop[,'y'] ← Pop[,'y'] + move_y
```

Bonk! Hitting the wall!



What if the red blood cell is at the boundary?

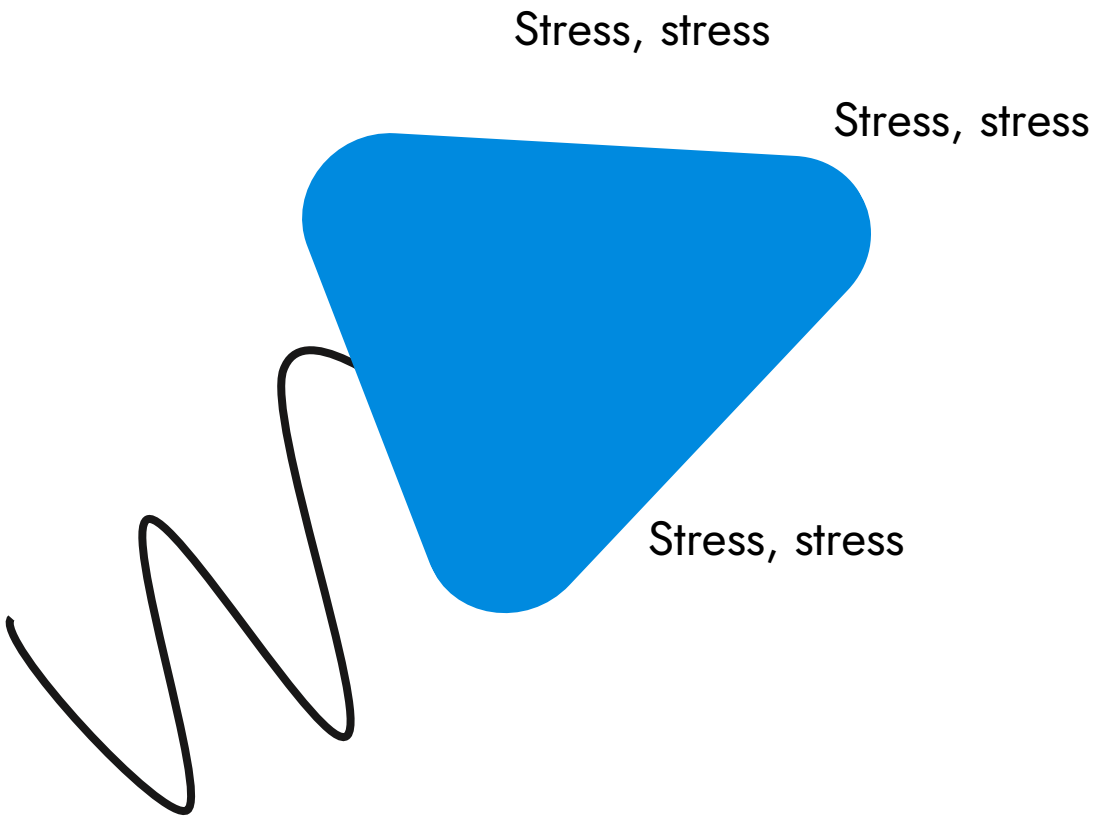
The best way is that if the RBC goes out of bounds, gently guide it back

```
#If going out of the right boundary
if (Pop[p,'x'] > 100){
  Pop[p,'x'] ← 99
}
#If going out of the left boundary
if (Pop[p,'x'] < 1){
  Pop[p,'x'] ← 2
}
#If going out of the top boundary
if (Pop[p,'y'] > 100){
  Pop[p,'y'] ← 99
}
#If going out of the bottom boundary
if (Pop[p,'y'] < 1){
  Pop[p,'y'] ← 2
}
```

Loop through everyone, and
make sure no one is out of bounds.

Because we're doing a 100 x 100
grid, the agents are out of bounds
if they're above 100 or less than 1 in the
x and y coordinates.

Getting some to die when their time is up



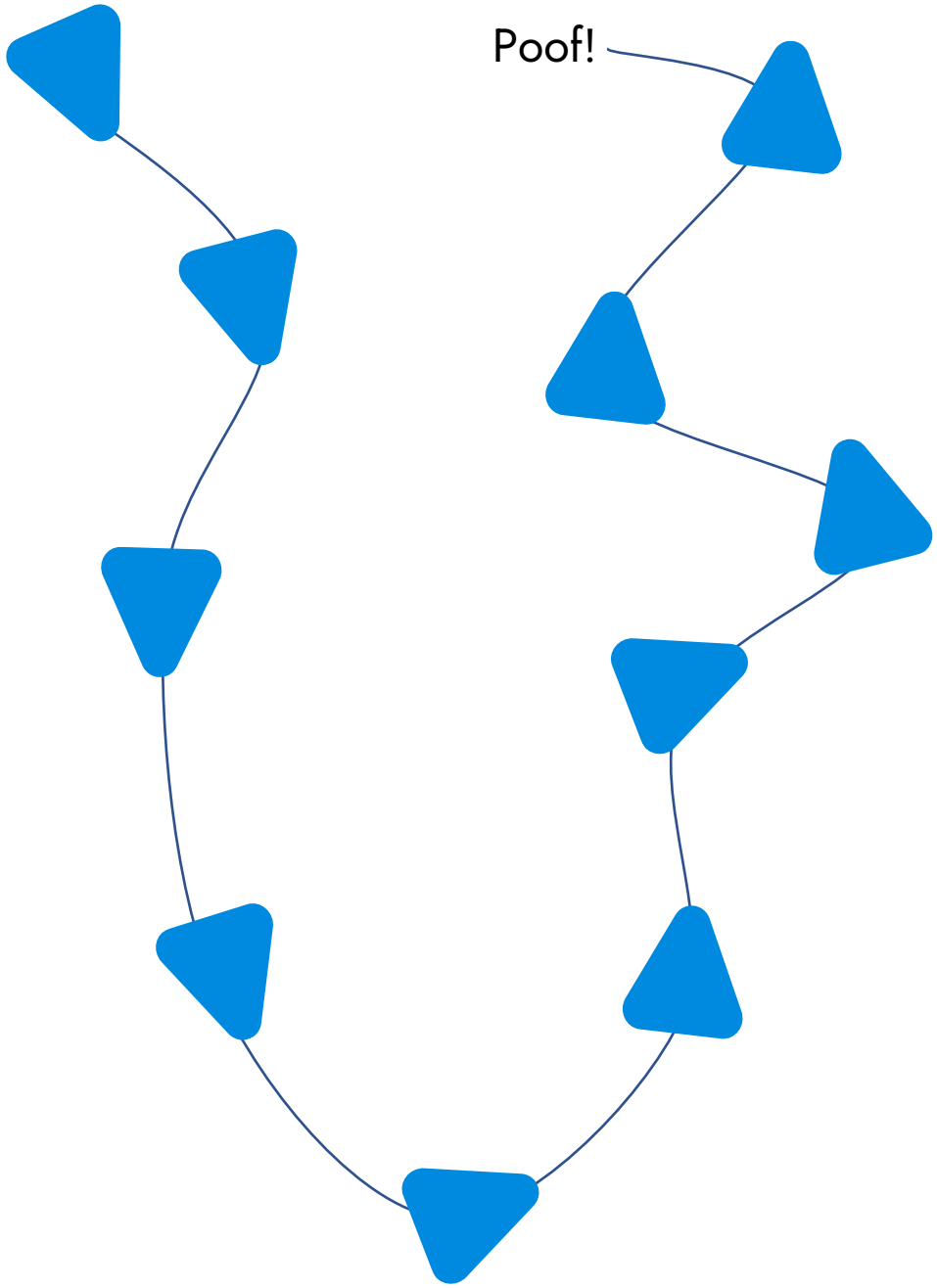
Here, malaria parasites tend to have very short life-spans.

It's more likely that these parasites will die before infecting a RBC

I'm going to assume that death is a time matter.

I can assume that the parasites will die in 10 time steps unless they infect a RBC

That is why one of the column is the time alive column in the data.frame



The trick is to have a little counter that adds one with each time step!

I create three data.frames within the function to divide it up into the red blood cells, the parasites, and the infected red blood cells

```
RBC <-      subset(Pop, Pop$type=='R')
###Parasite
Parasite <- subset(Pop,Pop$type=='P')
###Infected
Infected_RBC <- subset(Pop,Pop$type=='I')

#Bye Bye
Dead_Parasite<- subset(Parasite, Parasite$time_alive > 10 & Parasite$infect == "F")
```

Using the subset function, I subset the parasite data.frame (because these are the only agents that naturally die)

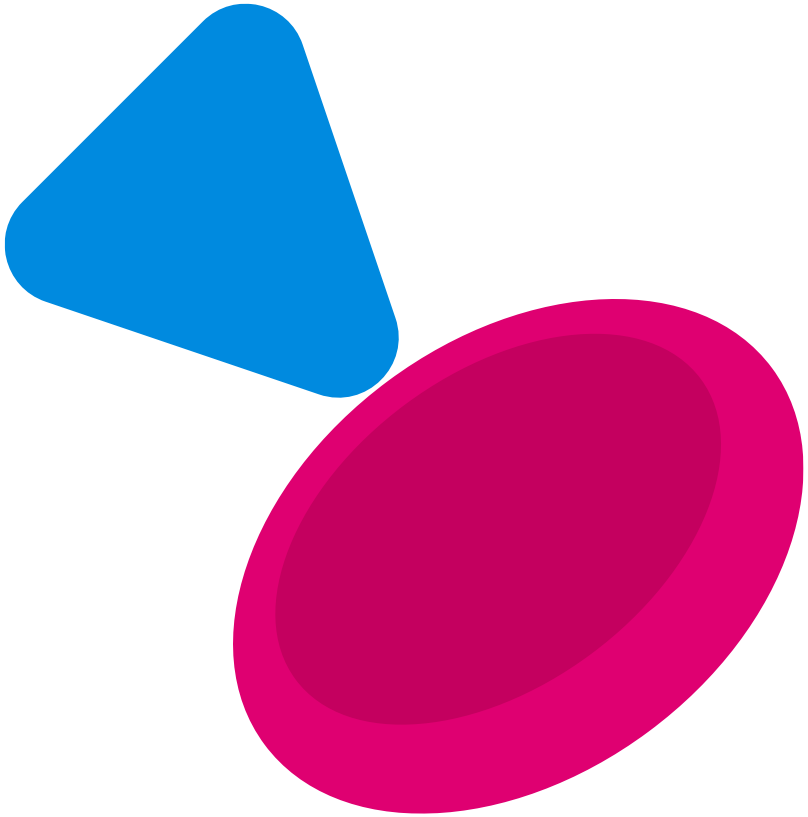
Let's say that all parasites die at time step 10 if they have not infected anyone

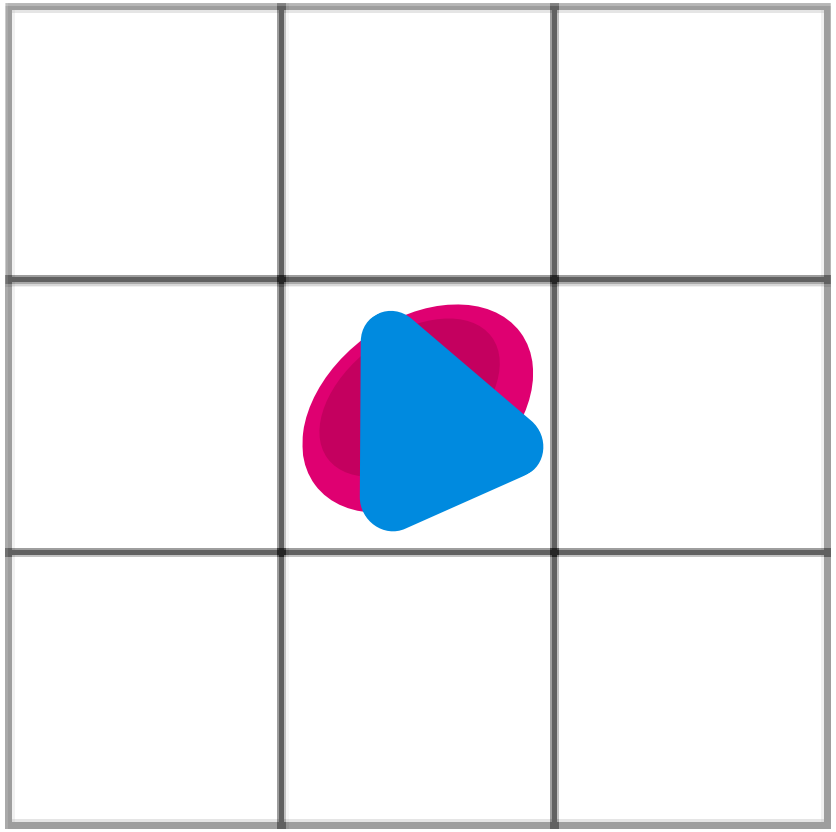
```
Parasite[Parasite$individuals %in% Dead_Parasite_Inds,]$dead = 'T'
Parasite[Parasite$individuals%in% Dead_Parasite_Inds,]$x = NA
Parasite[Parasite$individuals%in% Dead_Parasite_Inds,]$y = NA
```

We then update the full parasite data.frame, by making sure they are dead.

This might not be the best way but if the parasite die, I make the coordinates NA to prevent them from floating around in the grid.

Getting some to infect or be infected





Infection occur when a red blood cell and the parasite is on the same square

I create three data.frames within the function to divide it up into the red blood cells, the parasites, and the infected red blood cells

```
###RBC
RBC ←  subset(Pop, Pop$type=='R')
###Parasite
Parasite ← subset(Pop,Pop$type=='P')
###Infected
Infected_RBC ← subset(Pop,Pop$type=='I')

Same_spots ← merge(RBC, Parasite, by =c('x','y'))
```

I use merge to figure out which x and y coordinates are shared by the RBC and parasite

```
Same_spots <- merge(RBC, Parasite, by =c('x','y'))
```

This is one way to figure out which parasite and RBCS are occupying the same space!

The merge function will combine the columns

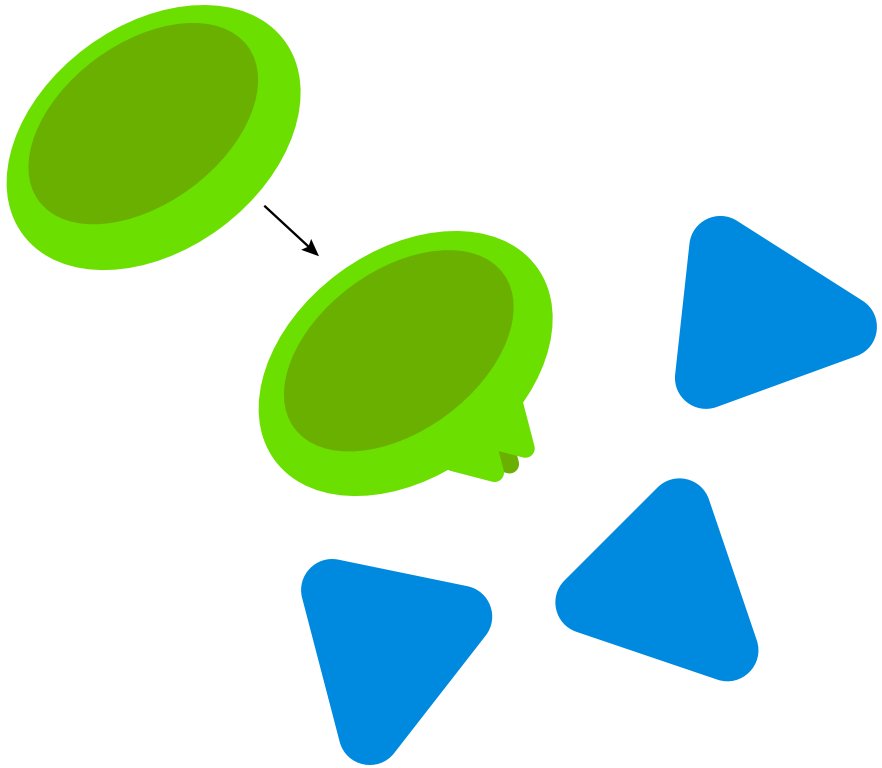
- individuals.x give the identification number of the RBC
- individual.y give the identification number of the parasite

Then you update the necessary columns

```
RBC_ind ← Same_spots$individuals.x
Parasite_ind ← Same_spots$individuals.y

RBC[RBC$individuals== RBC_ind ,]$type← "I"
RBC[RBC$individuals== RBC_ind,]$infected← "T"
RBC[RBC$individuals== RBC_ind,]$time_infected ← 1

Parasite[Parasite$individuals==Parasite_ind ,]$infect ←"T"
Parasite[Parasite$individuals==Parasite_ind ,]$x ← NA
Parasite[Parasite$individuals== Parasite_ind ,]$y ← NA
```

Getting them to burst

Bursting is like the death function and it’s driven by a timer

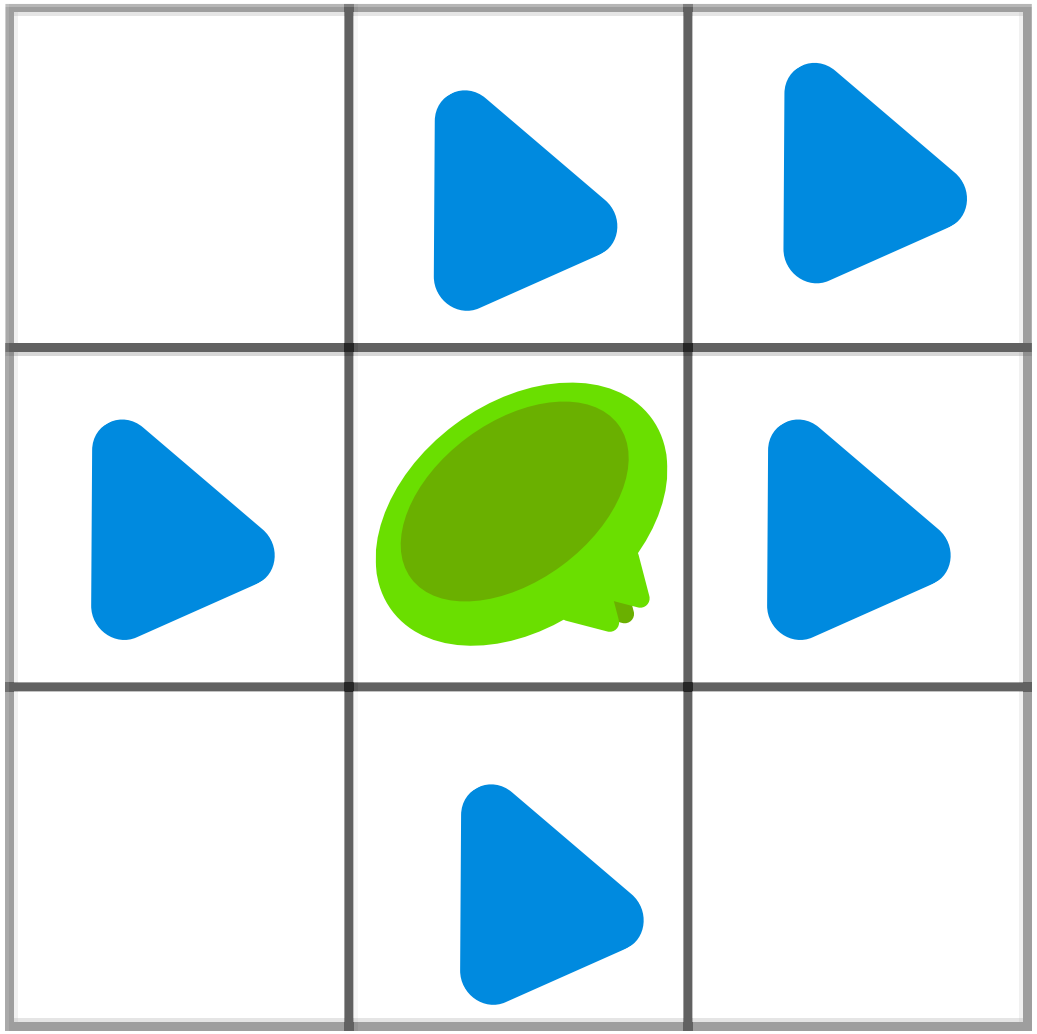


Similar to the death function

```
  #Bursting parasites
Burst ← subset(Infected_RBC,  Infected_RBC$time_infected > 5 & Infected_RBC$dead == "F")
```

Similar to what we have already done,
we are interested in the infected RBC that have not died
(they die from bursting) and so subset this information out!

We must figure out how many parasites will emerge from each parasite! And where!



I have it so that when the infected red blood cell releases parasites into the system, it is randomly dispersed around it

(Also the infected red blood cells die and disappear)

Assume each infected red blood cell releases 5 parasites

Retooling the movement model

```
New_Parasites ← NULL
for (ind in seq(1,length(Burst_Ind))){
  ind_no ← Burst_Ind[ind]
  x_pos ←  Infected_RBC[Infected_RBC$individual ==ind_no,]$x
  y_pos ←  Infected_RBC[Infected_RBC$individual ==ind_no,]$y


  move_x ← sample(x = c(-1, 0, 1), size = 5,replace=TRUE)
  move_y ← sample(x = c(-1, 0, 1), size = 5,replace=TRUE)

  x_pos_new← x_pos + move_x
  y_pos_new ← y_pos + move_y
```

For each infected red blood cell, I give them 5 parasites and put these in a list

Each new parasite needs to be tagged with an individual ID and such

```
New_Parasites[[ind]]←

 data.frame(
  type = rep("P",5),
   x =  x_pos_new,
   y =   y_pos_new ,
   time_alive = rep(0,5),
   time_infected = rep(0,5),
   infected = rep("F",5),
   dead = rep("F",5),
   infect = rep("F",5),
   time = unique(Pop$time)
 )
```

Type: So everyone is a parasite,
X and Y: They get their x and y coordinate,
Time alive: All of them are 'new born' so they have 0 as their time alive,
Infected: they can't be infected (because they're not RBC)
Dead: They're not dead (Yet)
Infect: They haven't infected anyone (Yet)
Time: Their time step is dependent on what time the entire system is.

We want to give new individual id number

So if the last individual ID is 100 then the 5 new parasites should be 101,102,103,104, and 105!

These ID should then increase!

one simple way is to use the tail of the last ID number and then add the numbers to it.

```
previous_last_entry← tail(Pop,n=1)$individuals


New_Parasites_F$individuals ←New_Parasites_F$individuals+ previous_last_entry
```

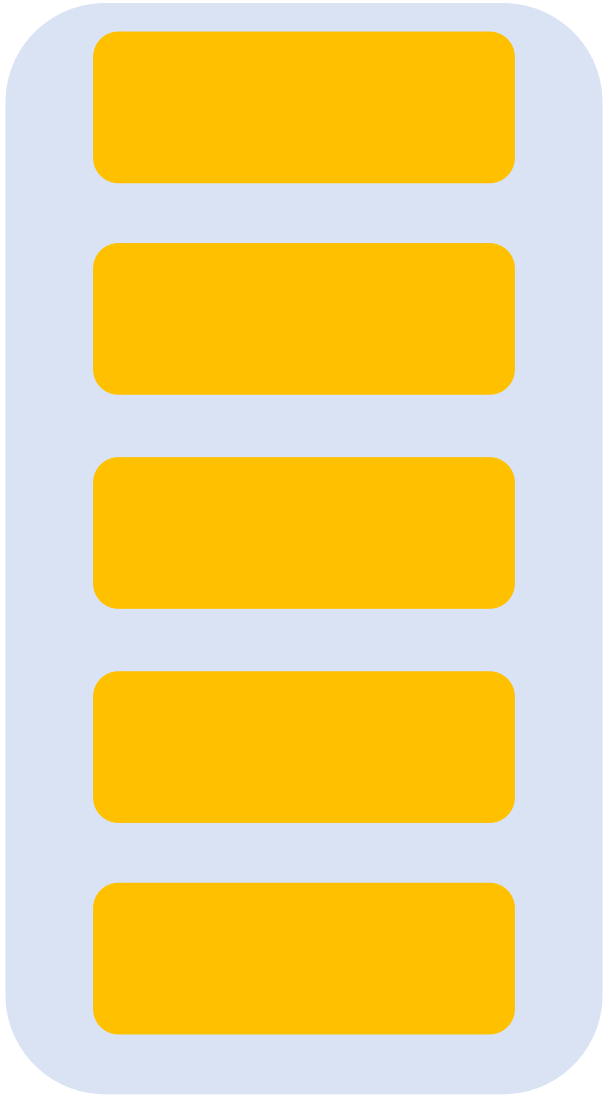
Processes are all done; how to run?

This is not the most computational efficient and will get into trouble as you run it longer with more individuals/timesteps- but this should give you some idea of how to run this.

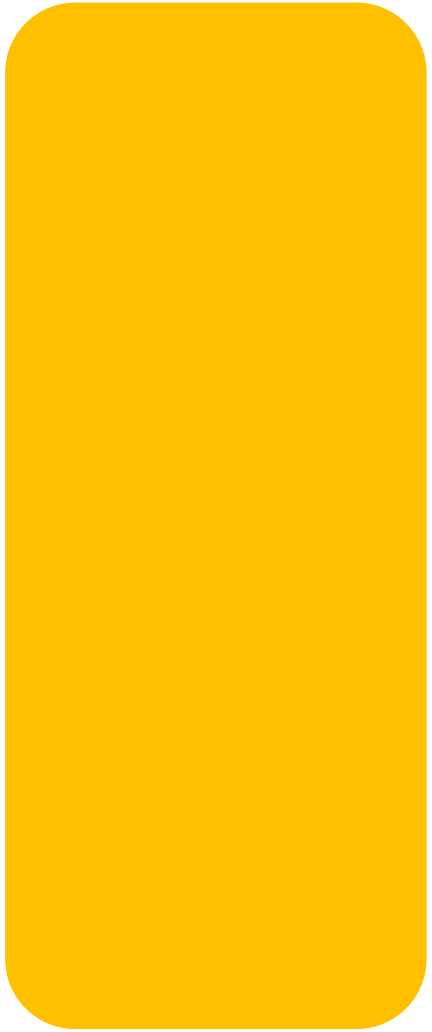
- Disclaimer: I'm thinking of a better way to do this, if you can offer any insight please email me!

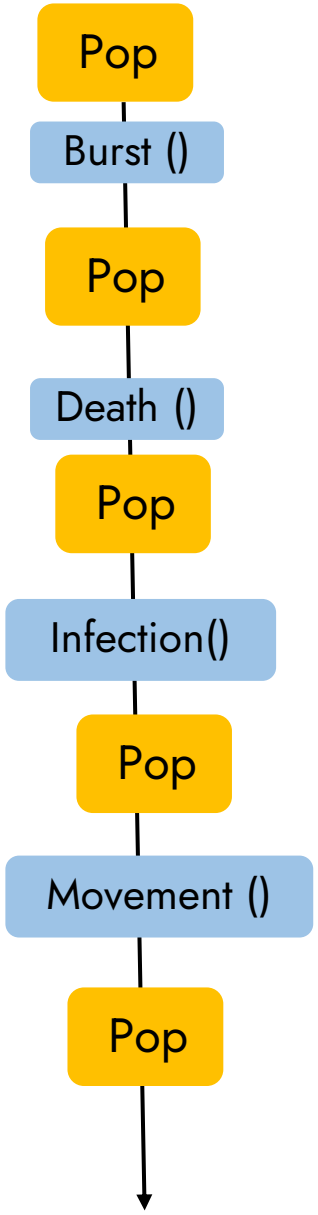
1) Create an empty list

2) Save the data.frame at each time step



3) Bind the elements together as one data.frame





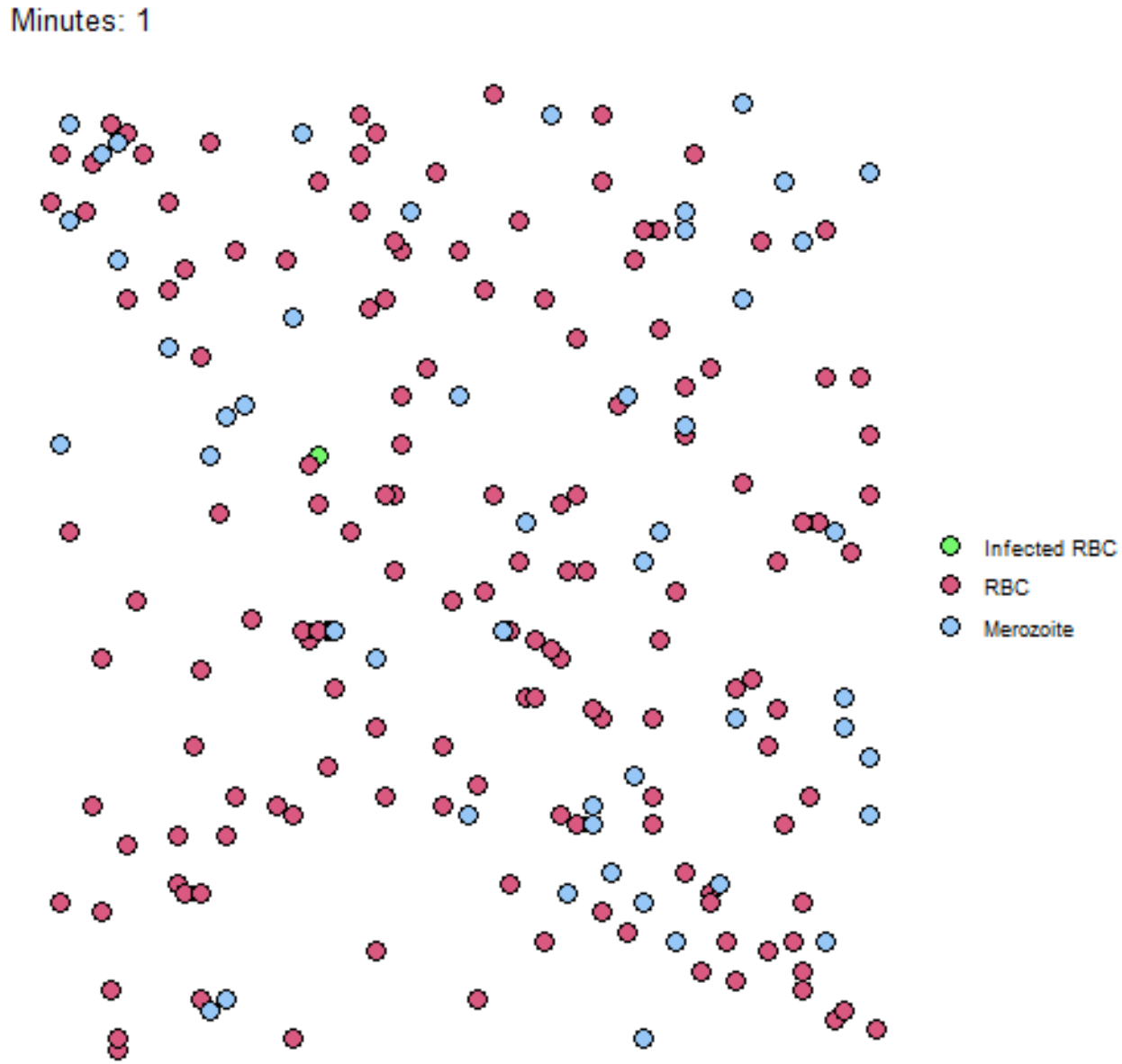

```
full_list ← NULL
for (k in seq(1,100)){

  Pop$time ← k
  Pop ← Burst(Pop)
  Pop ← Death(Pop)
  Pop ← Infection(Pop)
  Pop ← Movement(Pop)
  Pop[Pop$dead = 'F',]$time_alive ← Pop$time_alive +1
  Pop[Pop$time_infected≠0,]$time_infected ←  Pop[Pop$time_infected≠0,]$time_infected+1



  full_list[[k]] ← Pop

}
```

With each loop, remember to add +1 for the counter

Using gganimate

gganimate is a great package to see the code in action. Because we're interested in each frame being time, we use transition_time()

```
ggplot(Pop_All, aes(x= x, y= y,fill=type))+
  geom_point(size=4,shape=21)+
  scale_fill_manual(name="",
                    label=c("I"='Infected RBC', "R"='RBC', "P"= 'Merozoite'),
                     values=c("I"='#74f76d','R'='#d95981', 'P'='#97c7f7'))+
                transition_time(time)+ theme_void()+ labs(title = 'Minutes: {frame_time}')
```


Minutes: 1
Infected RBC
RBC
Merozoite

How to make better (Damie's TODO)

1) Make it a lot easier to change initial conditions
2) Make it a lot easier to change how many parasites can burst out of the parasite; time until death etc.
3) Maybe add some type of stochasticity (more random distribution of death and number of parasites)

etc