Actividad 3: Simulación de la geometría de ciudad y estructura de datos para las personas

Ximena Cantón, Marissa Luna, Gustavo Aguilar, Nubia Garcidueñas

2024-11-13

En esta fase del reto, iniciarás un esquema en el que los individuos considerados son personas que se encuentran localizados en un espacio geográfico delimitado en una geometría conocida.

Para este fin, cada miembro del equipo deberá elegir una de las siguientes configuraciones de su problema:

Problema 1:

- Crear una ciudad cuadrada donde cada lado tiene tamaño D con distribución uniforme de personas.
- Crear un arreglo de posiciones x, y de posiciones y considerando N personas. Asignar una posición inicial para cada una de las personas de la población por medio de un número aleatorio uniforme. En `R` puede realizarse como: `runif(N,min=0,max=1)`. Note que debe elegir valores apropiados para el mínimo y el máximo para asegurar que todas las personas estén dentro de los límites de la ciudad.

```
library(dplyr)
```

```
Attaching package: 'dplyr'
```

```
The following objects are masked from 'package:stats':

    filter, lag
```

```
The following objects are masked from 'package:base':

    intersect, setdiff, setequal, union
```

```r
library(ggplot2)

N <- 1000  # Num de personas
D <- 300   # Tamaño ciudad
r_infeccion <- 5  # Radio de infección
razon_recuperacion <- 0.2  # Razón de recuperación
num_iteraciones <- 10  # Número de iteraciones

# Posiciones aleatorias
x_positions <- runif(N, 0, D)
y_positions <- runif(N, 0, D)

I <- 10  # Num inicial infectados
R <- 0  # Num inicial recuperados
S <- N - I   # Num inicial susceptibles

# Variable categórica para el estado inicial
estados <- c(rep("Infectado", I), rep("Susceptible", S), rep("Recuperado", R))

# Crear un data frame
population <- data.frame(
  ID = 1:N,
  Posicion_X = x_positions,
  Posicion_Y = y_positions,
  Estado = factor(estados, levels = c("Susceptible", "Infectado", "Recuperado")),
  Iteracion = 0
)

# Función para ver si la distancia entre dos puntos es menor que el r_infeccion
es_cercano <- function(x1, y1, x2, y2, r) {
  distancia <- sqrt((x2 - x1)^2 + (y2 - y1)^2)
  return(distancia < r)
}

# Simulación de propagación de la infección
for (i in 1:num_iteraciones) {
  new_state <- population$Estado
```

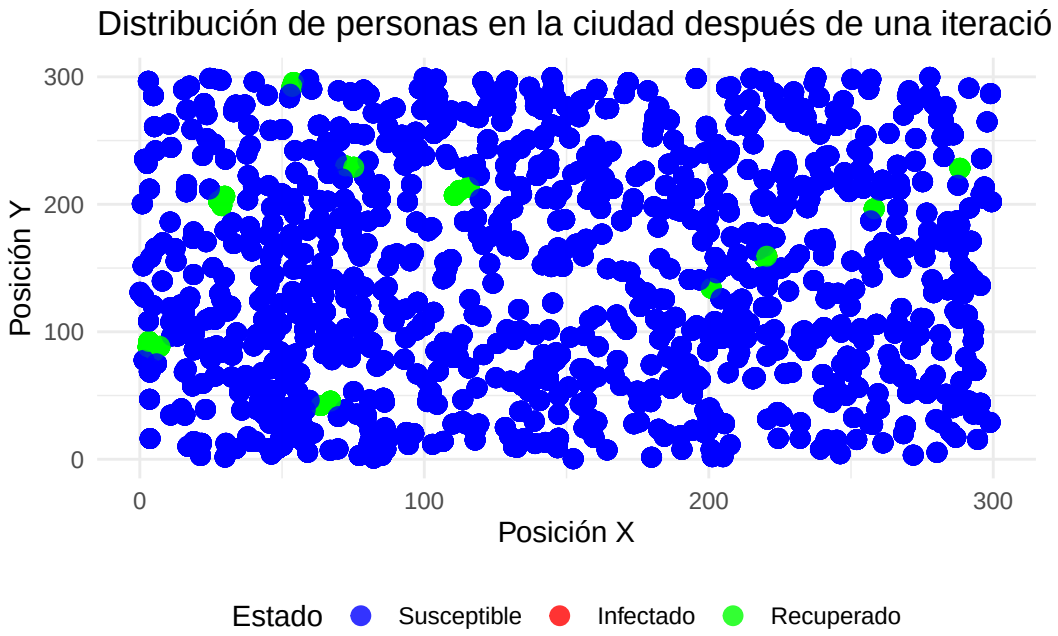
```
  infectados <- population %>% filter(Estado == "Infectado")
  susceptibles <- population %>% filter(Estado == "Susceptible")

  for (idx_infectado in infectados$ID) {
    for (idx_susceptible in susceptibles$ID) {
      if (es_cercano(
        population$Posicion_X[idx_infectado], population$Posicion_Y[idx_infectado],
        population$Posicion_X[idx_susceptible], population$Posicion_Y[idx_susceptible],
        r_infeccion)) {
        new_state[idx_susceptible] <- "Infectado"
      }
    }

    if (runif(1) < razon_recuperacion) {
      new_state[idx_infectado] <- "Recuperado"
    }
  }

  population$Estado <- new_state
  population$Iteracion <- i

  print(paste("Iteración:", i))
  print(table(population$Estado))
}
```

```
[1] "Iteración: 1"

Susceptible   Infectado  Recuperado
        980          19           1
[1] "Iteración: 2"

Susceptible   Infectado  Recuperado
        978          19           3
[1] "Iteración: 3"

Susceptible   Infectado  Recuperado
        978          13           9
[1] "Iteración: 4"

Susceptible   Infectado  Recuperado
        978          12          10
[1] "Iteración: 5"
```

```
Susceptible   Infectado  Recuperado
        978           9          13
[1] "Iteración: 6"

Susceptible   Infectado  Recuperado
        978           7          15
[1] "Iteración: 7"

Susceptible   Infectado  Recuperado
        978           5          17
[1] "Iteración: 8"

Susceptible   Infectado  Recuperado
        978           3          19
[1] "Iteración: 9"

Susceptible   Infectado  Recuperado
        978           3          19
[1] "Iteración: 10"

Susceptible   Infectado  Recuperado
        978           2          20
```

```r
# Inicializar un dataframe para almacenar la información de todas las iteraciones
population_history <- population

# Simulación de propagación de la infección
for (i in 1:num_iteraciones) {
  new_state <- population$Estado

  infectados <- population %>% filter(Estado == "Infectado")
  susceptibles <- population %>% filter(Estado == "Susceptible")

  for (idx_infectado in infectados$ID) {
    for (idx_susceptible in susceptibles$ID) {
      if (es_cercano(
        population$Posicion_X[idx_infectado], population$Posicion_Y[idx_infectado],
        population$Posicion_X[idx_susceptible], population$Posicion_Y[idx_susceptible],
        r_infeccion)) {
        new_state[idx_susceptible] <- "Infectado"
      }
```

```
    }

    if (runif(1) < razon_recuperacion) {
      new_state[idx_infectado] <- "Recuperado"
    }
  }

  population$Estado <- new_state
  population$Iteracion <- i

  # Guardar el estado de la población en la iteración actual
  population_history <- rbind(population_history, population)
}

# Graficar la propagación de la infección en cada iteración
ggplot(population_history, aes(x = Posicion_X, y = Posicion_Y, color = Estado)) +
  geom_point(alpha = 0.8, size = 3) +  # Ajustar tamaño y transparencia
  labs(title = "Distribución de personas en la ciudad después de una iteración de infección"
       x = "Posición X", y = "Posición Y") +
  theme_minimal() +
  scale_color_manual(values = c("blue", "red", "green")) +
  xlim(0, D) + ylim(0, D) +  # Limitar los ejes al tamaño de la ciudad
  theme(legend.position = "bottom")
```



Problema 2:

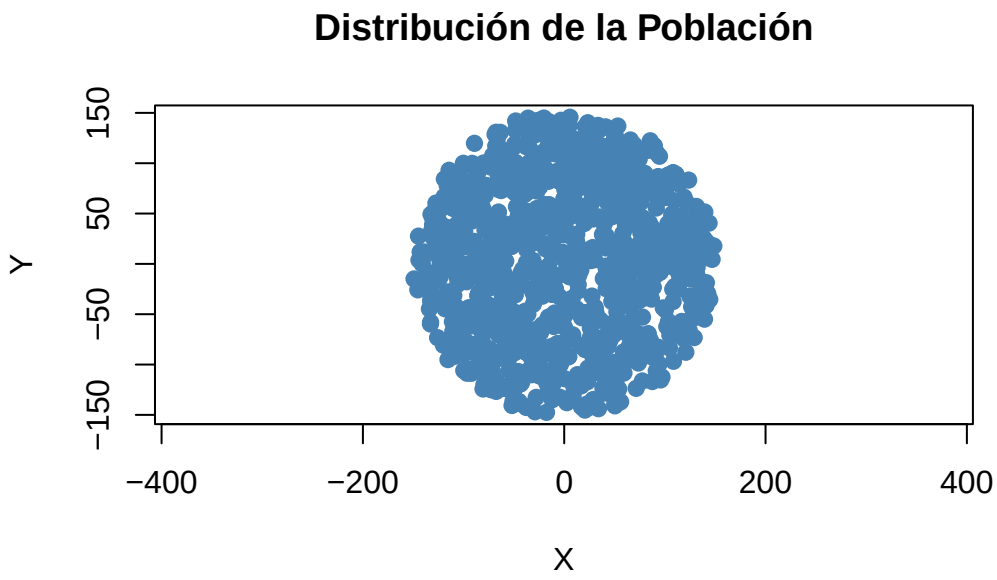
- Crear una ciudad circular de radio $D/2$ con distribución uniforme de personas, misma que no puede generarse de manera estándar sino que debe considerarse una distribución modificada tal como se ve en https://programming.guide/random-point-within-circle.html.
- Crear un arreglo de posiciones x, y de posiciones y considerando N personas. Asignar una posición inicial aleatoria para cada una de las personas de la población.

```
N <- 1000  #número de personas
D <- 300   #diametro de la ciudad
r <- 5      #radio de infección
gamma <- 0.2  #razón de recuperación
num_iter <- 10

a <- runif(N,min=0,max=2*pi)                #ángulo
radius <- D/2 * sqrt(runif(N,min=0,max=1))
x <- radius * cos(a)
y <- radius * sin(a)

#estados iniciales
I <- 1
```

```r
S <- N - I
R <- 0

state <- c(rep("Infected", I), rep("Suceptible", S), rep("Recovered", R))

#dataframe de la población
population <- data.frame(
  id = 1:N,
  x = x,
  y = y,
  state = factor(state, levels = c("Suceptible", "Infected", "Recovered")),
  iter = 0
)

dist_euc <- function(x1,x2,y1,y2){
  sqrt((x2 - x1)^2 + (y2 - y1)^2)
}

for (i in 1:num_iter){

  a <- runif(N,min=0,max=2*pi)
  radius <- D/2 * sqrt(runif(N,min=0,max=1))
  population$x <- radius * cos(a)
  population$y <- radius * sin(a)

  new_state <- population$state

  for (j in 1:N){
    if (population$state[j] == "Infected") {
      for (k in 1:N){
        if (population$state[k] == "Suceptible") {
          dist_measure <- dist_euc(population$x[j], population$x[k], population$y[j], popula
          if (dist_measure < r) {
            new_state[k] <- "Infected"
          }
        }
      }
      if (runif(1) < gamma) {
        new_state[j] <- "Recovered"
      }
    }
  }
```

```r
  population$state <- new_state
  population$iter <- i

  cat("Iteración:", i, "\n")
  print(table(population$state))
  plot(
    x, y,
    main = "Distribución de la Población",
    xlab = "X",
    ylab = "Y",
    pch = 16,             # Puntos sólidos
    col = "steelblue",
    cex = 1.2,            # Tamaño de los puntos
    asp = 1               # Proporción igual en los ejes
  )

  Sys.sleep(0.1)
}
```

```
Iteración: 1

Suceptible   Infected  Recovered
       999          1          0
```



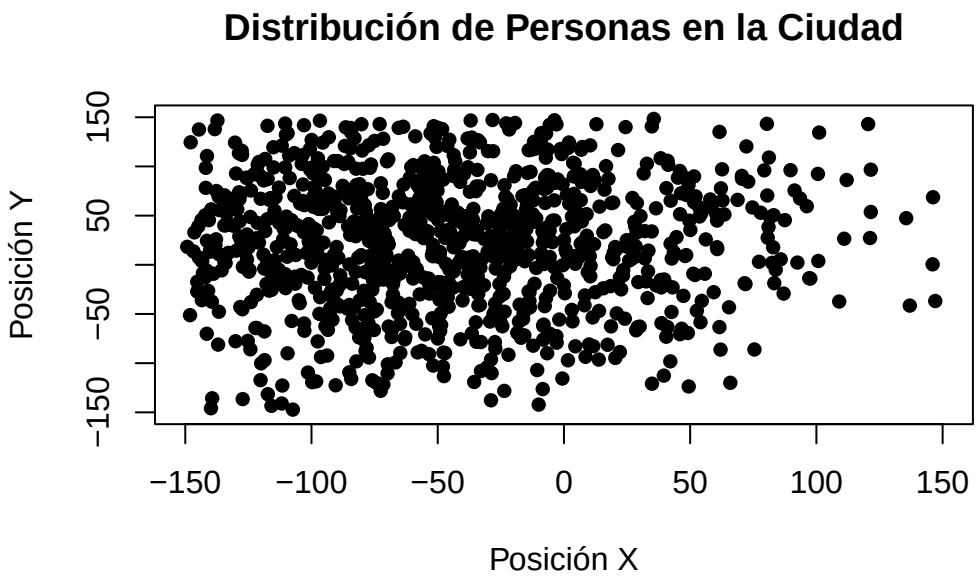

```
Iteración: 2

Suceptible   Infected  Recovered
       999          1          0


Iteración: 3

Suceptible   Infected  Recovered
       997          3          0


Iteración: 4

Suceptible   Infected  Recovered
       990          9          1


Iteración: 5

Suceptible   Infected  Recovered
       977         21          2


Iteración: 6
```

```
Suceptible   Infected  Recovered
       950         46          4

Iteración: 7

Suceptible   Infected  Recovered
       907         75         18

Iteración: 8

Suceptible   Infected  Recovered
       841        125         34

Iteración: 9

Suceptible   Infected  Recovered
       736        201         63

Iteración: 10

Suceptible   Infected  Recovered
       581        318        101
```

Problema 3:

- Crear una ciudad cuadrada de lado D en la que las personas están distribuidas en forma de "cluster" en donde hay una preferencia de las personas para estar ubicadas en cierta zona dentro de la ciudad. Para esto, defina un lugar de preferencia en forma aleatoria (x_0, y_0) y determine la posición aleatoria de la posición de N personas distribuidas de acuerdo a una distribución normal.
- Considere la función de la distribución normal (en `R` la función para generar números normalmente distribuidos con media 0 y desviación estándar 1 es `rnorm(N,mean=0,sd=1)`). Con esto, puede elegir coordenadas en x que se concentran alrededor de x_0 y coordenadas y que se concentran alrededor de y_0. Considere una desviación estándar de tamaño $D/20$.

```r
#parámetros iniciales
N <- 1000# número de personas
D <- 300   # lado de la ciudad (ahora un cuadrado)
r <- 5   # radio de infección
gamma <- 0.2  # razón de recuperación
num_iter <- 10  # número de iteraciones

#punto de preferencia dentro de la ciudad cuadrada
x0 <- runif(1, min = -D/2, max = D/2)
y0 <- runif(1, min = -D/2, max = D/2)

#distribución normal
sd_pos <- D / 4# desviación estándar para la distribución normal
x <- numeric(N)
y <- numeric(N)

for (i in 1:N) {
  repeat {
    #posición aleatoria con una distribución normal
    xi <- rnorm(1, mean = x0, sd = sd_pos)
    yi <- rnorm(1, mean = y0, sd = sd_pos)

    #verificar si el punto está dentro del cuadrado
    if (xi >= -D/2 && xi <= D/2 && yi >= -D/2 && yi <= D/2) {
      x[i] <- xi
      y[i] <- yi
      break  #si el punto está dentro, salimos del ciclo
    }
  }
}

# Graficar los puntos
plot(x, y, pch = 16, xlim = c(-D/2, D/2), ylim = c(-D/2, D/2),
     xlab = "Posición X", ylab = "Posición Y", main = "Distribución de Personas en la Ciudad"
```




```r
# Estados iniciales
I <- 1
S <- N - I
R <- 0

state <- c(rep("Infected", I), rep("Suceptible", S), rep("Recovered", R))

# DataFrame de la población
population <- data.frame(
  id = 1:N,
  x = x,
  y = y,
  state = factor(state, levels = c("Suceptible", "Infected", "Recovered")),
  iter = 0
)

# Función de distancia euclidiana
dist_euc <- function(x1, x2, y1, y2) {
  sqrt((x2 - x1)^2 + (y2 - y1)^2)
}

# Bucle de simulación
for (i in 1:num_iter) {
```

```
  # Actualizar posiciones alrededor de (x0, y0) en cada iteración
  x <- rnorm(N, mean = x0, sd = sd_pos)
  y <- rnorm(N, mean = y0, sd = sd_pos)

  population$x <- x
  population$y <- y

  # Copiar el estado actual para realizar cambios
  new_state <- population$state

  # Simular contagio y recuperación
  for (j in 1:N) {
    if (population$state[j] == "Infected") {
      for (k in 1:N) {
        if (population$state[k] == "Suceptible") {
          dist_measure <- dist_euc(population$x[j], population$x[k], population$y[j], popula
          if (dist_measure < r) {
            new_state[k] <- "Infected"
          }
        }
      }
      # Recuperación
      if (runif(1) < gamma) {
        new_state[j] <- "Recovered"
      }
    }
  }

  # Actualizar estado y mostrar estadísticas
  population$state <- new_state
  population$iter <- i

  Sys.sleep(0.1)
}
```

Problema 4:

- Crear una ciudad circular de radio $D/2$ con distribución de personas en "cluster". Considere la generación de números aleatorios en un círculo de acuerdo a https://programming.guide/random-point-within-circle.html.

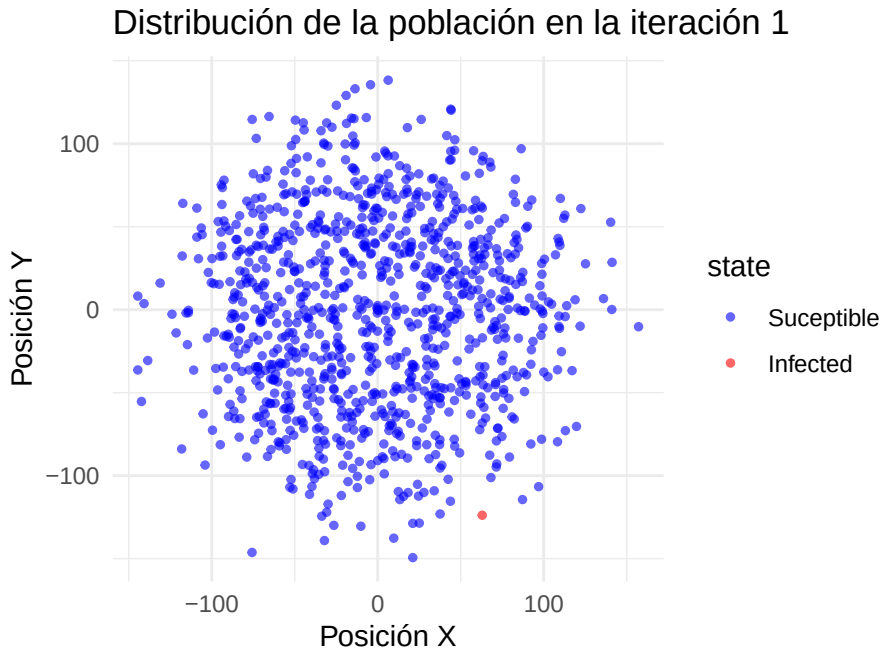
- Considere un arreglo para posiciones x, y otro para posiciones y correspondientes a N personas que se concentran de acuerdo a una distribución normal en un ángulo y distancia al centro seleccionados de manera aleatoria.

```
# Librería para gráficos
library(ggplot2)

# Parámetros
N <- 1000  #número de personas
D <- 300   #diametro de la ciudad
r <- 5      #radio de infección
gamma <- 0.2  #razón de recuperación
num_iter <- 10  # Número de iteraciones

# Generación de posiciones en "clusters" en un círculo
a <- runif(N, min = 0, max = 2 * pi)               # Ángulo aleatorio
radius <- abs(rnorm(N, mean = D / 4, sd = D / 10)) # Radio con distribución normal

# Coordenadas x e y en función del ángulo y radio
x <- radius * cos(a)
y <- radius * sin(a)

# Estados iniciales
I <- 1
S <- N - I
R <- 0

state <- c(rep("Infected", I), rep("Suceptible", S), rep("Recovered", R))

# DataFrame de la población
population <- data.frame(
  id = 1:N,
  x = x,
  y = y,
  state = factor(state, levels = c("Suceptible", "Infected", "Recovered")),
  iter = 0
)

# Función de distancia euclidiana
dist_euc <- function(x1, x2, y1, y2) {
  sqrt((x2 - x1)^2 + (y2 - y1)^2)
}
```

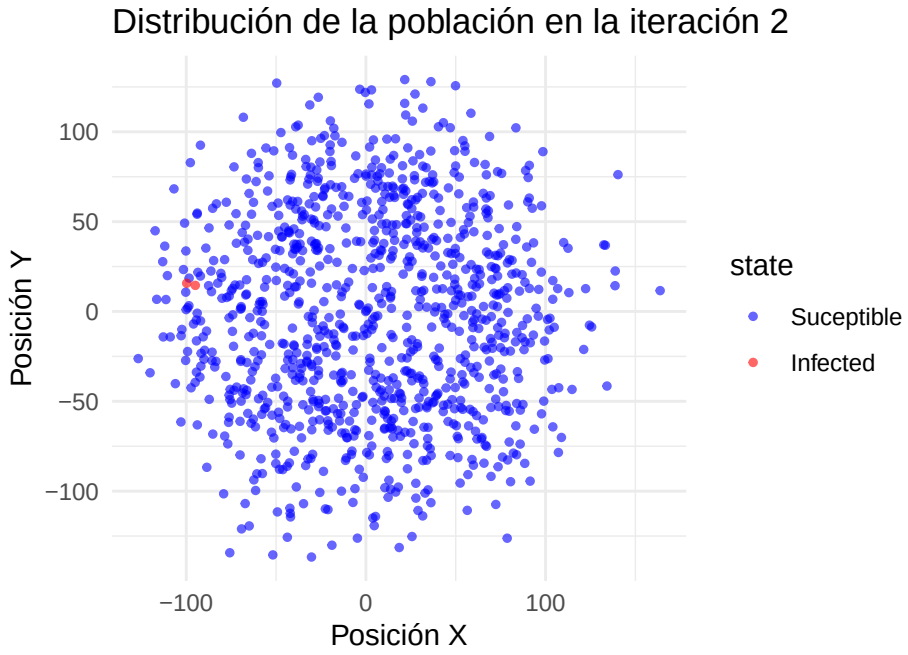
```r
# Simulación de la propagación de la infección
for (i in 1:num_iter) {

  # Movimiento aleatorio en "clusters" para cada iteración
  a <- runif(N, min = 0, max = 2 * pi)
  radius <- abs(rnorm(N, mean = D / 4, sd = D / 10)) # Distribución normal para la posición i
  population$x <- radius * cos(a)
  population$y <- radius * sin(a)

  new_state <- population$state

  for (j in 1:N) {
    if (population$state[j] == "Infected") {
      for (k in 1:N) {
        if (population$state[k] == "Suceptible") {
          dist_measure <- dist_euc(population$x[j], population$x[k], population$y[j], popula
          if (dist_measure < r) {
            new_state[k] <- "Infected"
          }
        }
      }
      # Probabilidad de recuperación
      if (runif(1) < gamma) {
        new_state[j] <- "Recovered"
      }
    }
  }

  population$state <- new_state
  population$iter <- i

  # Imprimir el estado actual
  cat("Iteración:", i, "\n")
  print(table(population$state))

  # Scatter plot de la población
  p <- ggplot(population, aes(x = x, y = y, color = state)) +
    geom_point(size = 1, alpha = 0.6) +
    coord_fixed() +
    labs(title = paste("Distribución de la población en la iteración", i),
         x = "Posición X", y = "Posición Y") +
    scale_color_manual(values = c("Suceptible" = "blue", "Infected" = "red", "Recovered" = "g
```

```
    theme_minimal()

  print(p)

  Sys.sleep(0.5)  # Pausa para observar el gráfico
}
```

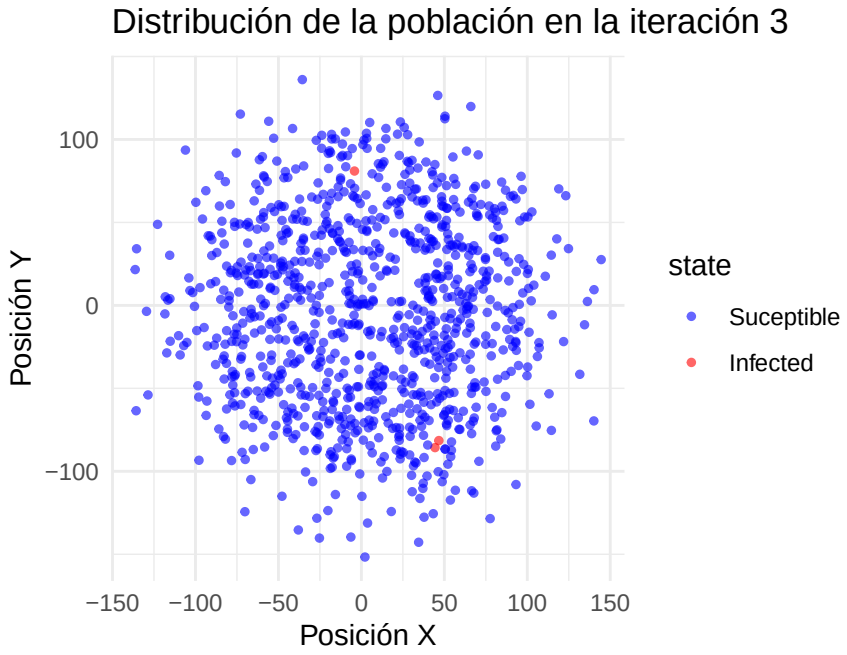
```
Iteración: 1

Suceptible   Infected  Recovered
       999          1          0
```



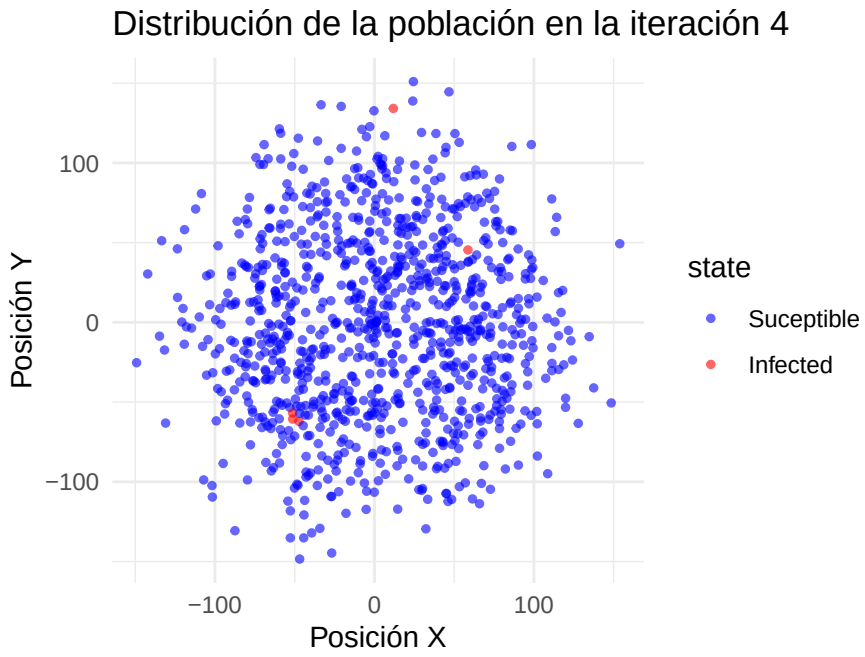

```
Iteración: 2

Suceptible   Infected  Recovered
       998          2          0
```



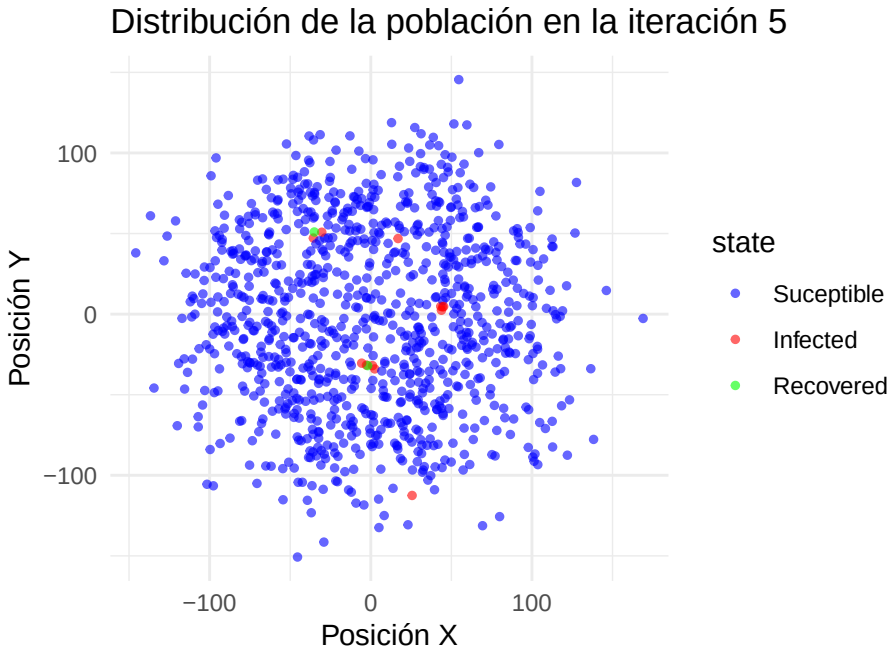

```
Iteración: 3

Suceptible   Infected  Recovered
       997          3          0
```



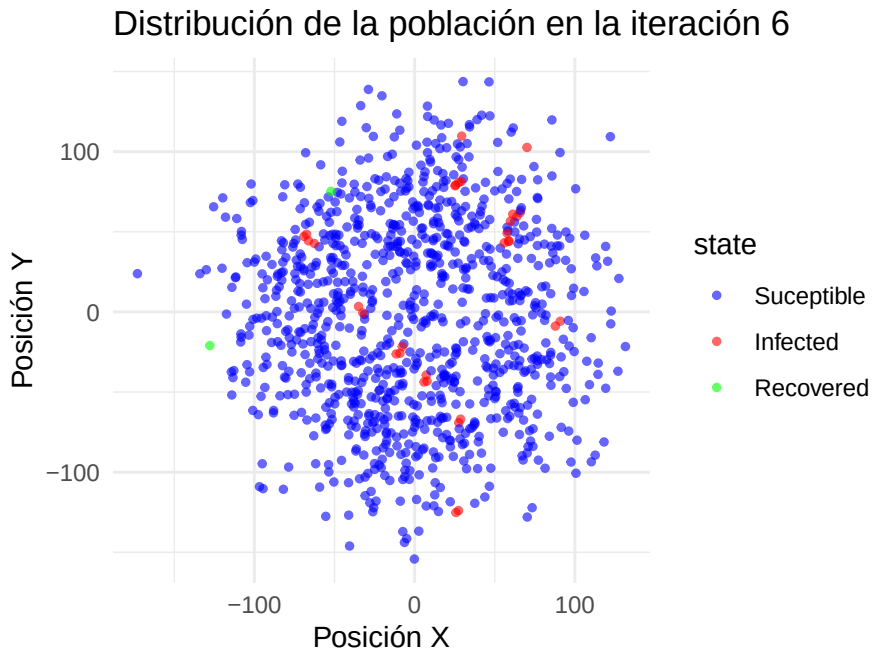

```
Iteración: 4

Suceptible   Infected  Recovered
       995          5          0
```



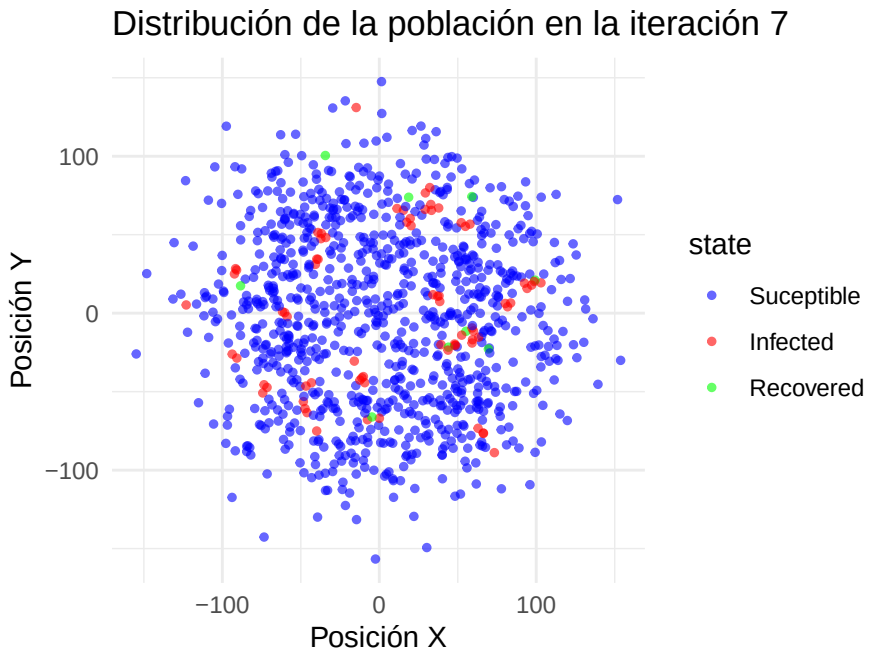

```
Iteración: 5

Suceptible   Infected  Recovered
       986         12          2
```



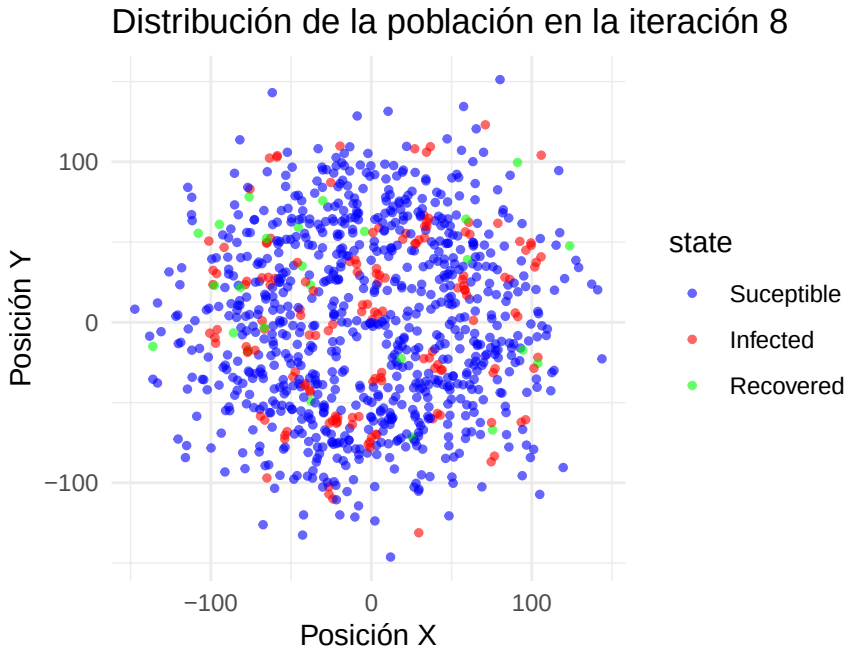

```
Iteración: 6

Suceptible   Infected  Recovered
       966         32          2
```



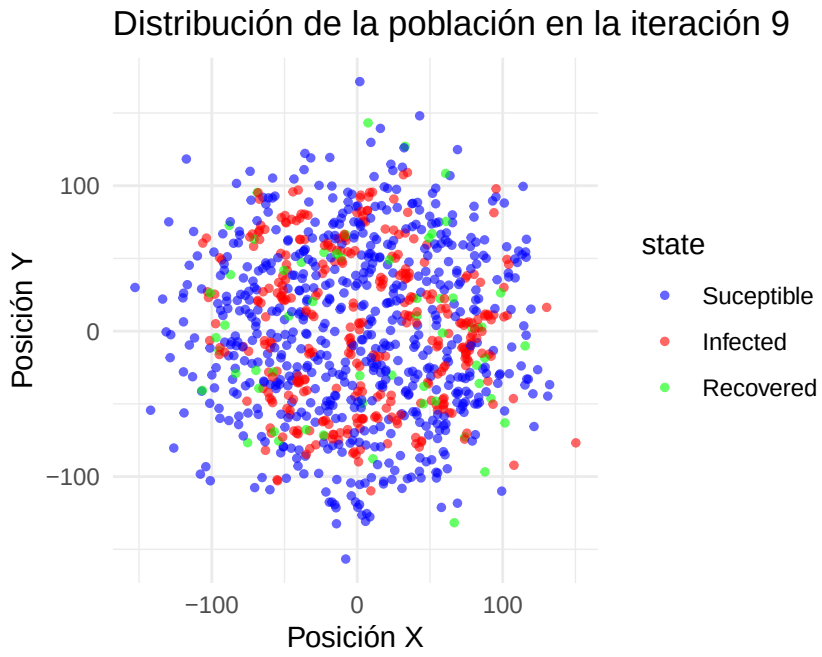

```
Iteración: 7

Suceptible   Infected  Recovered
       917         74          9
```



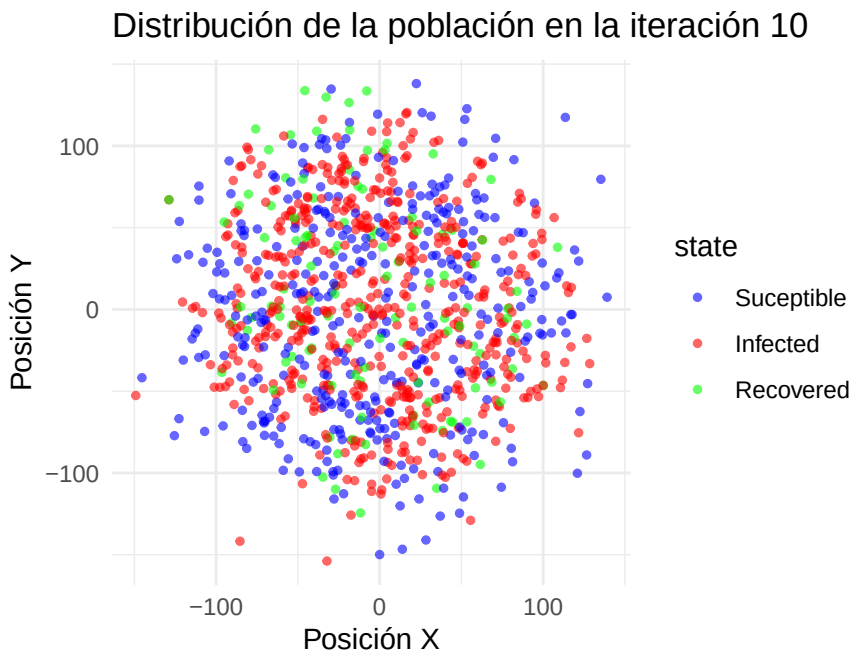

```
Iteración: 8

Suceptible   Infected  Recovered
       823        152         25
```




```
Iteración: 9

Suceptible   Infected  Recovered
       611        332         57
```




```
Iteración: 10

Suceptible   Infected  Recovered
       357        530        113
```



Para cada problema de esta actividad:

- Dado un valor inicial de N, crear 3 variables para contar el número inicial de infectados, de susceptibles y de recuperados. Las variables deben ser tales que $N = I + S + R$ y que haya por lo menos una persona infectada. Inicialmente no hay recuperados ($R = 0$).
- Crear 2 variables para definir el "radio" de infección r, que representará la distancia para poderse infectar (por ejemplo $r = 0.6$), y para la razón de recuperación.
- Crear una variable categórica (factor variable) que represente el estado de la persona (suceptible, infectada o recuperada).
- Crear una estructura de datos (dataframe o varios arreglos) para representar las variables `posición x`, `posición y`, `estado`, `id único de la persona` y `número de iteración` de las N personas de manera que haya I personas infectadas, S susceptibles y R recuperados.
- Escribir una función que revise la distancia euclidiana entre dos puntos y regrese `TRUE` (o `1`) si la distancia es menor que r y regrese `FALSE` (o `0`) si la distancia es mayor o igual que r.