Network Models in individual

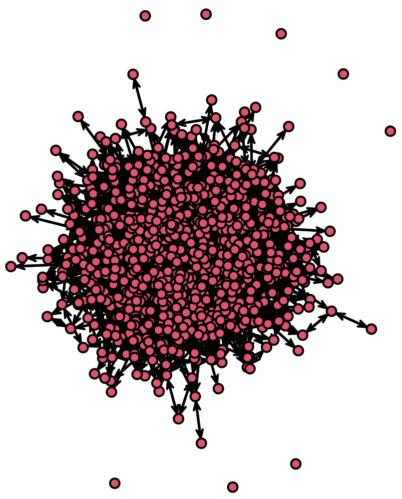
Sean L. Wu (slwood89@gmail.com)

Let's look at modifying the SIR example to be simulated on a network structure.

```r
library(network)
library(sna)
library(individual)
```

We'll use a simple Bernoulli random graph as the contact structure. We store the contact network as a `network` class object.

```r
N <- 1e3
g <- as.network(rgraph(n = N, tprob=0.005, mode = "graph"))
plot(g)
```



Now we set epidemic parameters, and sample the initial conditions.

```r
I0 <- 5
S0 <- N - I0
dt <- 0.1
tmax <- 100
steps <- tmax/dt
gamma <- 1/10
beta <- 0.05

health_states <- c("S","I","R")
health_states_t0 <- rep("S",N)
health_states_t0[sample.int(n = N,size = I0)] <- "I"
```

This `CategoricalVariable` stores the health status of each individual.

```r
health <- CategoricalVariable$new(categories = health_states,initial_values = health_states_t0)
```

Now, we'll assume that for a susceptible person, infection is transmitted according to rate β along each link to an infectious person. This means we'll need an `IntegerVariable` that counts how many infectious contacts each susceptible person has. We'll use this variable, `contacts` to sample new infections each time step.

```r
contacts <- IntegerVariable$new(initial_values = rep(0, N))
```

Next we'll need 2 `Bitset` objects. The `empty_bitset` is just a bitset of zeros, used to clear out the other one. The `contact_bitset` will contain those individuals an infectious individuals has a contact (outbound edges) with. It is usually more efficient to focus on contacts made by infectious individuals rather than contacts made by susceptible individuals, as we'll see later in the infectious process.

```r
contact_bitset <- Bitset$new(N)
empty_bitset <- Bitset$new(N)
```

The infection process is below. The rough logic is this:

1. Get the infected individuals, and convert to integer vector. This is because later we will need it to interact with the `network` class object we use to store the contact network, which does not know how to interact with bitsets.
2. If there could be infection events, get the susceptible individuals. We also set the variable `contacts` to zero.
3. For each infectious individual, do the following:
    1. Clear out `contact_bitset`
    2. `S_i` will be those susceptible individuals who are contacted by this infected.
    3. `S_i` is the intersection of all susceptibles and the individuals on the outbound edges of this infected, where we use `get.neighborhood` from the `network` package to get those vertices being contacted, which are put into `contact_bitset`.
    4. Update the variable `contacts` by incrementing the contacted susceptible's number of infectious contacts by 1.
4. Set `S_vulnerable` to be the susceptible individuals with ≥ 1 infectious contact.
5. The force of infection is `S_vulnerable[i] * beta`, use that to sample those who will become infected on this time step.
6. Queue the health update.

```r
# infection process
infection_process <- function(t) {

  # infected individuals
  I <- health$get_index_of("I")$to_vector()

  # only do this if there could be infection events
  if (length(I) > 0 & health$get_size_of("S") > 0) {

    # susceptible individuals
    S <- health$get_index_of("S")

    # clear out inbound infectious contacts
    contacts$queue_update(values = 0,index = NULL)
    contacts$.update()

    # I's send out contacts
    for (i in seq_along(I)) {

      # clear out the extra bitset
      contact_bitset$and(empty_bitset)

      # S_i are susceptibles being contacted by i-th I person
      S_i <- S$copy()
      S_i$and(contact_bitset$insert(v = get.neighborhood(x = g,v = I[i],type = "out")))

      # add to their contacts
      contacts$queue_update(values = contacts$get_values(S_i) + 1,index = S_i)
      contacts$.update()
    }

    # S's with I contacts (people with >= 1 infectious contact)
    S_vulnerable <- contacts$get_index_of(a = 1,b = N)

    # the foi on each of them is num contacts * beta
    foi <- contacts$get_values(S_vulnerable) * beta

    # sample those who get sick
    S_vulnerable$sample(rate = pexp(q = foi * dt))

    # queue update
    health$queue_update(value = "I",index = S_vulnerable)

  }
}
```

The scheduled recovery event and rendering are the same as the SIR tutorial.

```r
# recovery event
recovery_event <- TargetedEvent$new(population_size = N)
recovery_event$add_listener(function(t, target) {
  health$queue_update("R", target)
})
```

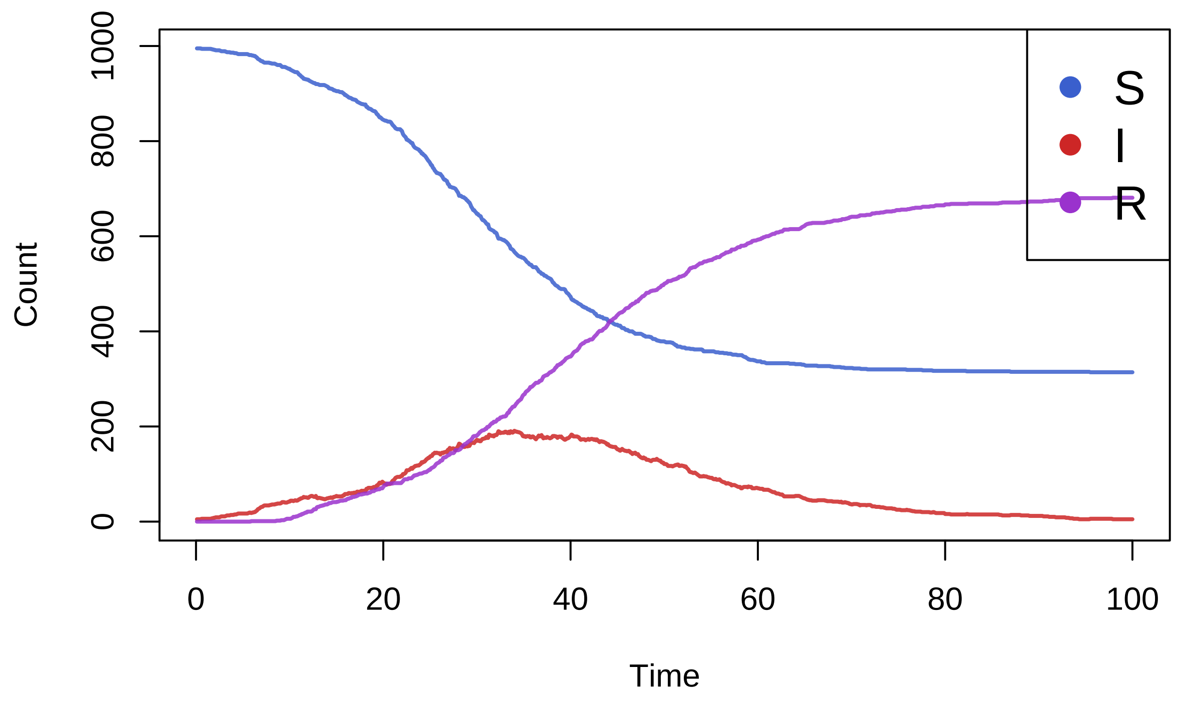
```r
# recovery process
recovery_process <- function(t){
  I <- health$get_index_of("I")
  already_scheduled <- recovery_event$get_scheduled()
  I$and(already_scheduled$not())
  rec_times <- rgeom(n = I$size(),prob = pexp(q = gamma * dt)) + 1
  recovery_event$schedule(target = I,delay = rec_times)
}

# rendering
health_render <- Render$new(timesteps = steps)
health_render_process <- categorical_count_renderer_process(
  renderer = health_render,
  variable = health,
  categories = health_states
)
```

Run the simulation and plot the output.

```r
# simulation
system.time(
  simulation_loop(
    variables = list(health),
    events = list(recovery_event),
    processes = list(infection_process,recovery_process,health_render_process),
    timesteps = steps
  )
)
```

```
##    user  system elapsed
##   7.179   0.000   7.180
```

```r
states <- health_render$to_dataframe()
health_cols <-  c("royalblue3","firebrick3","darkorchid3")
matplot(
  x = states[[1]]*dt, y = states[-1],
  type="l",lwd=2,lty = 1,col = adjustcolor(col = health_cols, alpha.f = 0.85),
  xlab = "Time",ylab = "Count"
)
legend(
  x = "topright",pch = rep(16,3),
  col = health_cols,bg = "transparent",
  legend = health_states, cex = 1.5
)
```



Count
0
200
400
600
800
1000
0
20
40
60
80
100
Time
S
I
R