

```
In [70]:  
    using Agents  
    using DataFrames  
    using Distributions  
    using GraphPlot  
    using LightGraphs  
    using Plots  
    using Clustering  
    using Colors
```

## Coupling The Campus Drinking Model

For this work, we use the simplified model of a social drinker.

### Healthy Drinker:

$$\frac{ds}{dt} = s_{\text{slope}}s + m(t)$$
$$\frac{dp}{dt} = \gamma[-p + \sigma(as) + b]$$
$$s(t_i) \leftarrow s(t_i) - s_{\text{drink}} \quad \text{if } t_i \in e$$

where  $c$  represents craving,  $p$  represents probability of drinking (if alcohol is encountered),  $m(t)$  is given and represents mood, and  $e$  is a given set of encounter times. Together these equations model an individual's drinking behavior.

### Coupling Multiple Individuals

To couple together the drinking of multiple people, we use a simple rule. Each time I drink, my neighbors in the social network are given an opportunity to drink at the next timestep. This rests on the idea that if my friends go out and drink, I am likely to get invited to go with them. We implement this as follows. If person  $i$  drinks at time  $t$ , and has neighbors  $j_1, \dots, j_m$ , then we update the set of event  $e_{j_k}$  times belonging to each neighbor to include the time  $t + \delta t$  where  $\delta t$  is the model timestep.

## Implementation

### ODE Solver (RK4)

```
In [3]:  
    """  
        rk4_step(f::F, y::T, t::T, dt::T; params::Array{T, 1}=[] ) -> dy  
  
        Returns the fourth order runge-kutta approximation of the change in `y` over the  
        timestep `dt`  
  
        ### Parameters  
        1. `f` (callable) accepts `y`, `t` and all of the parameters in `params`
```

```

2. `y` current system state
3. `t` time value
4. `dt` timestep size
5. `params` parameters for `f`

### Returns
`dy` approximation of the change in `y` over the timestep `dt`
"""

function rk4_step(f::F, y::T, t::T, dt::T; params=[]) where {F, T <: AbstractFloat}
    two = convert(T, 2)
    dy1 = f(y, t, params...) * dt
    dy2 = f(y + dy1 / two, t + dt / two, params...) * dt
    dy3 = f(y + dy2 / two, t + dt / two, params...) * dt
    dy4 = f(y + dy3, t + dt, params...) * dt
    dy = convert(T, 1/6) * (dy1 + dy2 * two + dy3 * two + dy4)
    return dy
end

function rk4_step(f::F, y::Array{T,1}, t::T, dt::T; params=[]) where {F, T <: AbstractFloat}
    two = convert(T, 2)
    dy1 = f(y, t, params...) .* dt
    dy2 = f(y .+ dy1 ./ two, t .+ dt ./ two, params...) * dt
    dy3 = f(y .+ dy2 ./ two, t .+ dt ./ two, params...) * dt
    dy4 = f(y .+ dy3, t .+ dt, params...) * dt
    dy = convert(T, 1/6) .* (dy1 .+ dy2 .* two .+ dy3 .* two .+ dy4)
    return dy
end

```

Out[3]: rk4\_step (generic function with 2 methods)

## ODE Definitions

```

In [4]:
"""
Sigmoid function """
σ(x::T) where T = 1. / (1. + exp(-x))
"""
Derivative of social drinking propensity """
dsdt(s::T, t::T, s_slope::T, s_drink::T, m::F) where {F, T <: AbstractFloat} = s
"""
Derivative of drinking probability """
dpdt(p::T, t::T, a::T, b::T, γ::T, s::T) where T = γ * (-p + σ(a * s) + b)
"""
Return true if current timestep contains an event time """
event(events::Array{T, 1}, t::T, dt::T) where T = any((events .< t + dt) .& (eve

```

Out[4]: event

## Mood and Event Time Generators

```

In [5]:
"""
random_event_times(end_time::T; start_time::T=0.0; dist::Distribution=Exponen
Returns an array of event times where t_i - t_(i-1) is distributed according to
`dist`.

**Parameters**

1. `end_time` maximum length of time to simulate. All returned event times
will be strictly less than this value
2. `start_time` Initial time. Defaults to zero. All events will be
strictly between start_time and max_time

```

```

3. `scale` Parameter for the exponential distribution
**Returns**

`event_times` An array of event times with values between start_time and end_time
The distance between events is exponentially distributed with parameter `scale`.
"""

function random_event_times(
    end_time::Float64;
    start_time::Float64=0.0,
    dist::Type=Exponential,
    distparams::Array{Float64, 1}=[1.0]
)
    event_times::Array{Float64, 1} = []
    current_event = start_time
    d = dist(distparams...)
    while current_event < end_time
        next_event = current_event + rand(d)
        push!(event_times, next_event)
        current_event = next_event
    end
    return event_times
end

"""

random_phase_sin(freq) -> func

Creates a sin function with frequency `freq` and random phase.
"""

function random_phase_sin(freq::Float64)
    a = rand()
    func = t::Float64 -> sin(2π*freq*(t - a))
    return func
end

```

Out[5]: random\_phase\_sin

## Social Drinker Definition and State Update

```

In [6]: mutable struct SocialDrinker{F, T} <: AbstractAgent
    id::Int
    pos::Int
    s::T
    p::T
    d::T
    e::T
    m::F
    s_slope::T
    s_drink::T
    γ::T
    a::T
    b::T
    events::Array{T, 1}
end

function Base.show(io::IO, sd::SocialDrinker)
    descr = "SocialDrinker"
    print(io, descr)
end

```

```

function Base.show(io::IO, a::Array{SocialDrinker, N}) where N
    descr = "Array{SocialDrinker, $N}"
    print(io, descr)
end

"""
    agent_step!(sd::SocialDrinker, model::AgentBasedModel)

The social drinker acts during a timestep. First, social drinking propensity
and drinking probability are evolved according to their respective ODE, then
if the agent has a drinking opportunity during this timestep, they drink with
probability `sd.p`.

"""

function agent_step!(sd::SocialDrinker, model::AgentBasedModel)
    # Agent did not drink yet during this timestep
    sd.d = 0.0
    sd.e = 0.0
    # Update internal state
    sd.s += rk4_step(dsdt, sd.s, model.t, model.dt, params=[sd.s_slope, sd.s_dri
    sd.p += rk4_step(dpdt, sd.p, model.t, model.dt, params=[sd.a, sd.b, sd.γ, sd
    event(sd.events, model.t, model.dt) && event_step!(sd)
end

""" Social drinker response to drinking event """
function event_step!(sd::SocialDrinker)
    sd.e = 1.0
    if rand() < sd.p
        sd.d = 1.0
        sd.s += sd.s_drink
    end
end

```

Out[6]: event\_step!

In [7]:

```

newdrinker(id, maxdays; freq=1) =
    SocialDrinker(id, 0, 0.0, 0.0, 0.0, 0.0, random_phase_sin(1.0), -1., -.5, 3.

```

Out[7]: newdrinker (generic function with 1 method)

## Model Initializer

In [9]:

```

function nightlife(; npeople=2, adj=[0. 1.; 1. 0.], starttime=0.0, maxtime=30.0,
    space = GraphSpace(Graph(1)) # Dummy space to place agents
    properties = Dict(:t => starttime, :dt => dt, :adj => adj, :npeople => npeop
    atype = SocialDrinker{typeof(random_phase_sin(1.0)), Float64}
    model = AgentBasedModel(atype, space; properties)
    for i in 1:npeople
        add_agent!(newdrinker(i, maxtime), model)
    end
    return model
end

```

Out[9]: nightlife (generic function with 1 method)

## Coupling Rule

People are coupled in the model in the following way: If I drink, each of my neighbors in the supplied social network is given an **opportunity** to drink at the next timestep. That is, if I drink at time  $t$ , we add  $t + dt$  to each neighbor's set of drinking opportunities  $e$ .

In [10]:

```
function model_step!(model)
    drinks = [model[i].d for i in 1:model.npeople]
    drink_influence = model.adj * drinks .> 0.0
    for id in findall(drink_influence)
        push!(model[id].events, model.t + model.dt)
    end
    model.t += model.dt
end
```

Out[10]: model\_step! (generic function with 1 method)

## Plotting and Data Measurement

In [135...]

```
function plot_agent(i::Int, adf::DataFrame, mdf::DataFrame)
    ai = filter(:id => ==(i), adf)
    sp = plot(mdf.t, ai.s, color=:pink, linewidth=2, label="Social Propensity",
    maxtime = round(mdf.t[end]))
    title!("$maxtime Days of Drinking Behavior")
    pp = plot(mdf.t, ai.p, color=:darkblue, alpha=.7, linewidth=2, label="Probability")
    ylims!(-.1, 1.1)
    yticks!([0, 1])
    dp = plot(mdf.t, ai.e, color=:orange, alpha=.7, linewidth=2, label="Drinking Probability")
    plot!(mdf.t, ai.d, color=:green, linewidth=2, label="Drinking Behavior", legend=false)
    ylims!(-.2, 1.2)
    yticks!([0, 1], ["Abstain", "Drink"])
    p = plot(sp, pp, dp, layout=(3, 1), size=(1000,375))
    return p
end

function drinking_stats(i::Int, adf::DataFrame, mdf::DataFrame)
    aidf = filter(:id => ==(i), adf)
    totdrinks = sum(aidf.d)
    totenc = sum(aidf.e)
    meanprob = mean(aidf.p)
    return totdrinks, totenc, meanprob
end

function raster_plot(adf::DataFrame; starttime=0.0, dt=0.001, opportunites=false)
    drinks = filter(:d => ==(1.0), adf)
    dtimes = drinks.step .* dt
    p = plot(dtimes, drinks.id, seriestype=:scatter, alpha=0.7, color=:green, label="Opportunities")
    if opportunites
        opps = filter(:e => ==(1.0), adf)
        optimes = opps.step .* dt
        plot!(optimes, opps.id)
    end
    ylabel!("Person Id")
    xlabel!("Time")
    return p
end
```

```
Out[135... raster_plot (generic function with 1 method)
```

# Results

## Single Individual

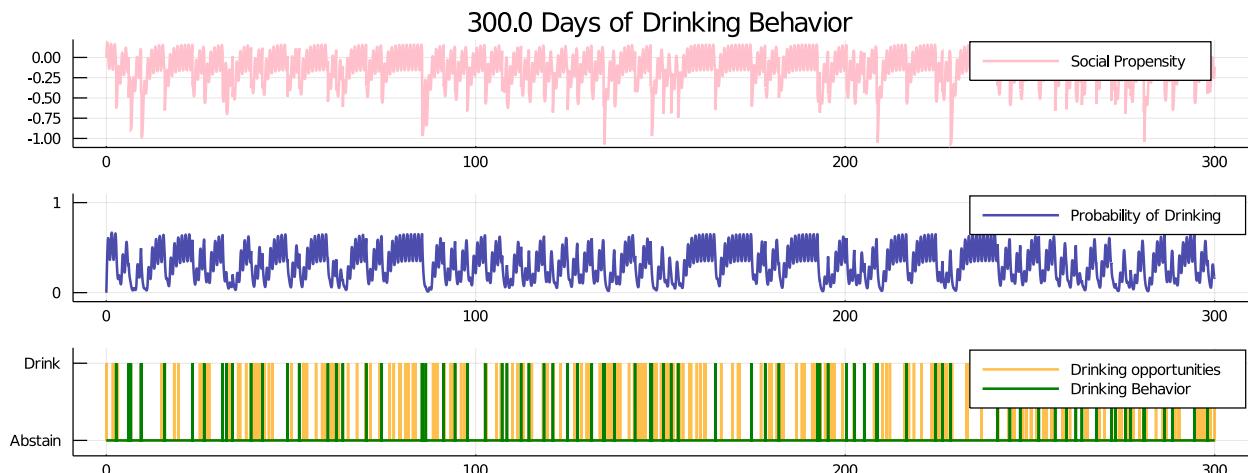
```
In [136...]
```

```
adata = [:s, :p, :d, :e]
mdata = [:t]
maxdays = 300.0
dt = .001
n = floor(Int, maxdays / dt)
single_adj = reshape([0.0], 1, 1)
single_nl = nightlife(maxtime=maxdays, npeople=1, adj=single_adj)
single_adf, single_mdf = run!(single_nl, agent_step!, model_step!, n; adata, mda
```

## Results

```
In [137...]
```

```
display(plot_agent(1, single_adf, single_mdf))
td, te, mprob = drinking_stats(1, single_adf, single_mdf)
println("Total Drinks: $td (Per day = $(td/maxdays))")
println("Total Encounters: $te (Per day = $(te/maxdays))")
println("Mean of Probability of Drinking Variable: $mprob")
```



## Two individuals coupled together

```
In [138...]
```

```
nl = nightlife(maxtime=maxdays)
adf, mdf = run!(nl, agent_step!, model_step!, n; adata, mdata);
```

## Individual 1

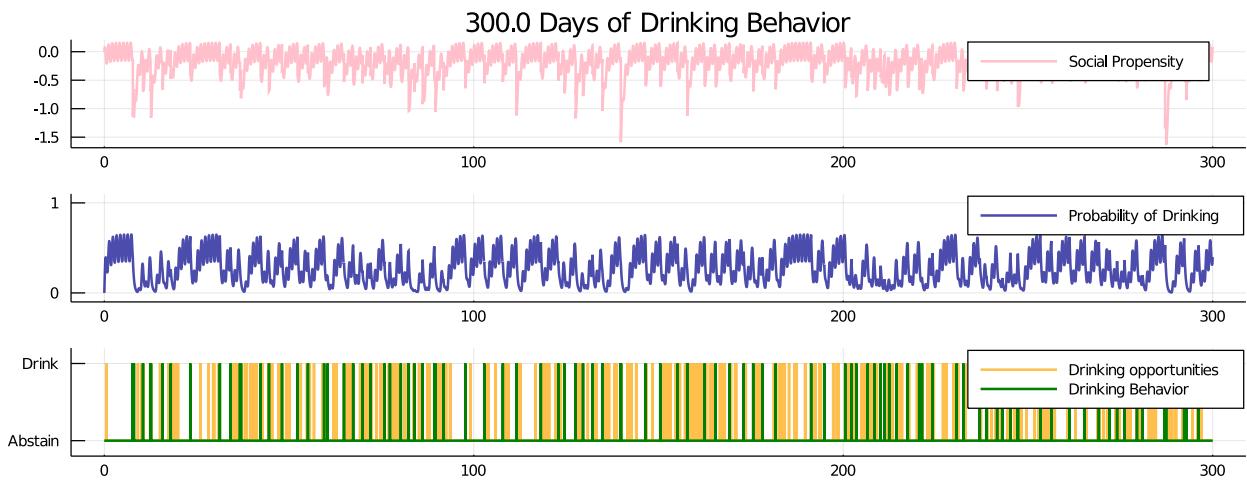
```
In [139...]
```

```
display(plot_agent(1, adf, mdf))
td, te, mprob = drinking_stats(1, adf, mdf)
println("Total Drinks: $td (Per day = $(td/maxdays))")
```

```

println("Total Encounters: $te (Per day = $(te/maxdays))")
println("Mean of Probability of Drinking Variable: $mprob")

```



Total Drinks: 105.0 (Per day = 0.35)

Total Encounters: 415.0 (Per day = 1.383333333333333)

Mean of Probability of Drinking Variable: 0.27637436574562363

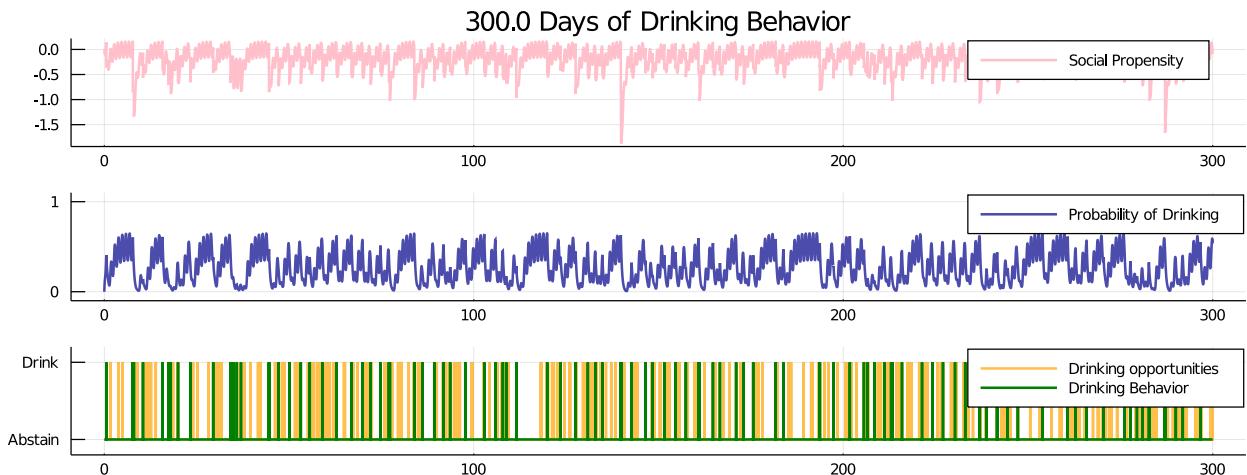
## Individual 2

In [140]:

```

display(plot_agent(2, adf, mdf))
td, te, mprob = drinking_stats(2, adf, mdf)
println("Total Drinks: $td (Per day = $(td/maxdays))")
println("Total Encounters: $te (Per day = $(te/maxdays))")
println("Mean of Probability of Drinking Variable: $mprob")

```



Total Drinks: 108.0 (Per day = 0.36)

Total Encounters: 390.0 (Per day = 1.3)

Mean of Probability of Drinking Variable: 0.27239964979266684

## Simulation of 50 People.

**Identical parameters, random phase sine waves, random encounter times**

In [31]:

```

n = 50
p = .02
A1 = Float64.(rand(n, n) .< p)

```

```

maxtime = 60.0
big_nl = nightlife(; npeople=n, adj=A1, maxtime=maxtime)
dt = .001
steps = floor(Int, maxdays / dt)
adf1, mdf1 = run!(big_nl, agent_step!, model_step!, steps; adata, mdata);

```

Mean degree: 1.02

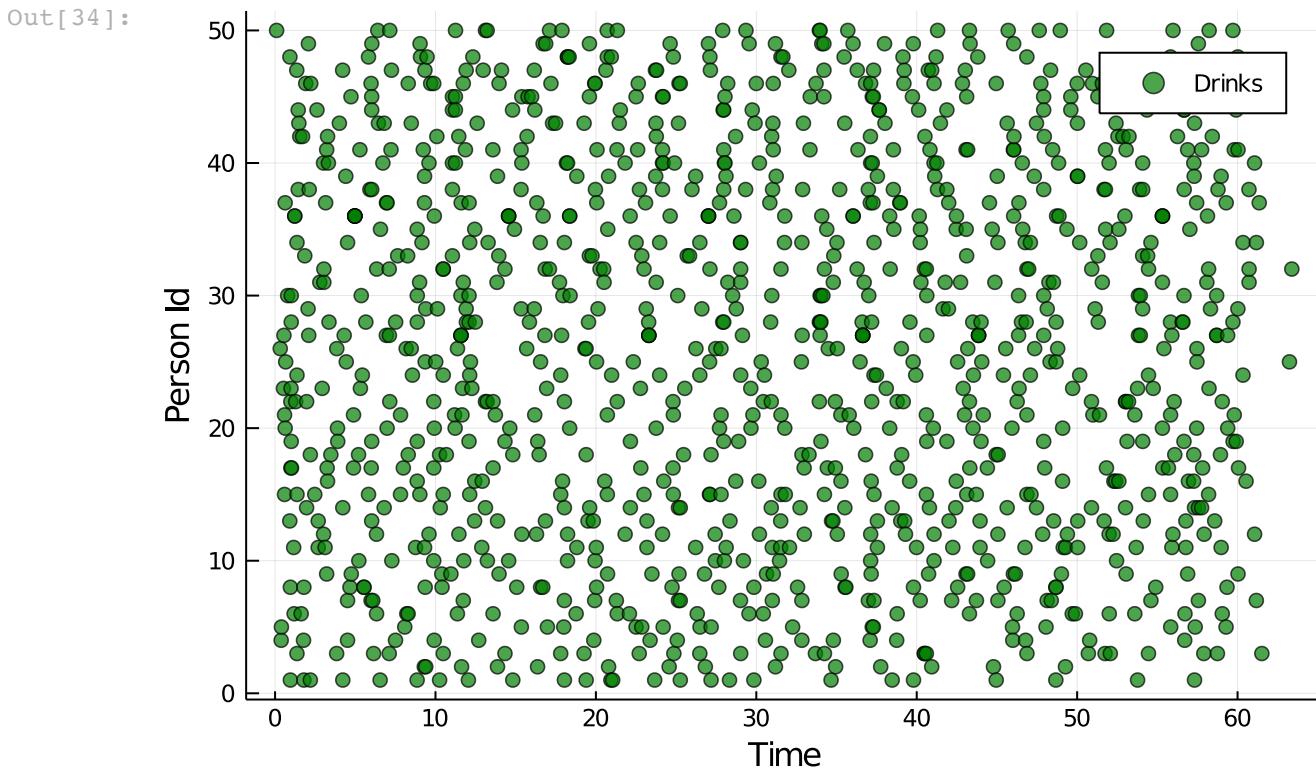
## Random Graph Mean Degree Appox. 1

In [32]: `g = DiGraph(A1)`

Out[32]: {50, 51} directed simple Int64 graph

In [ ]: `gplot(DiGraph(Int.(A1)), nodefillc=colorant "orange")`

In [34]: `raster_plot(adf1)`



Each point  $(t, i)$  in this plot is a drinking instance of the person with id number  $i$  at time  $t$ . Thus  $i$  is always an integer. Horizontal lines at each integer pass through the drinking events of a single individual. Vertical lines at  $t = t_0$  pass through all drinking events that occurred at time  $t_0$ .

We are interested in seeing if certain people drink together or in quick succession. Are there groups of people who frequently drink together? Do any patterns stand out from this plot? We might assume that the model will cause people in the same network communities to drink together. We can investigate this by reordering node indexes by community.

## Reorder by Community in Social Network

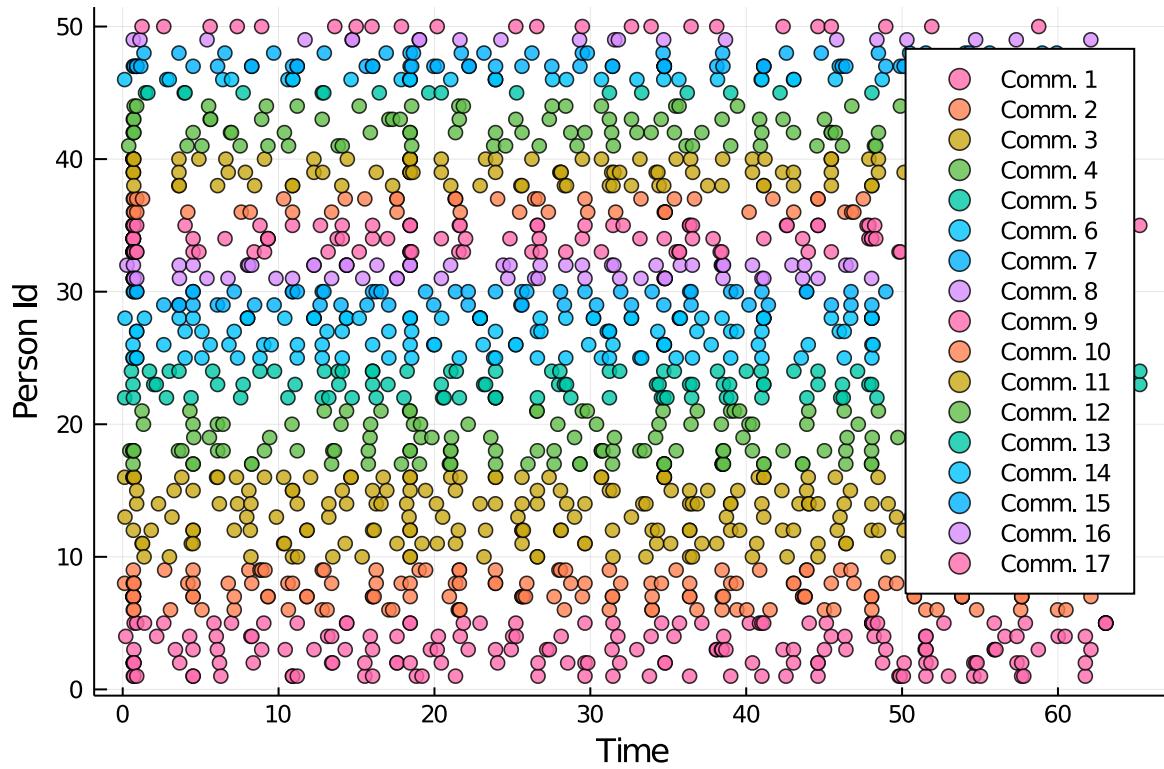
```
In [106]: clusters = mcl(A1, expansion=2, inflation=2)
nclust = maximum(clusters.assignments)
print(nclust)
```

17

```
In [114]: perm = sortperm(clusters.assignments)
idmapping = zeros(n)
for (i, pi) in enumerate(perm)
    idmapping[pi] = i
end
colors = range(LChab(70,70,0), stop=LChab(70,70,720), length=nclust)
#colors = distinguishable_colors(nclust)
p = plot()
for c in 1:nclust
    ids = findall(clusters.assignments .== c)
    clustdf = filter(:id => in(ids), adf3)
    cdrinks = filter(:d => ==(1), clustdf)
    dtimes = cdrinks.step .* dt

    plot!(dtimes, [idmapping[id] for id in cdrinks.id], seriestype=:scatter, al
end
ylabel!("Person Id")
xlabel!("Time")
```

Out[114]:



## Random Graph: Mean Degree Approx. 3

```
In [35]: n = 50
p = .06
A3 = Float64.(rand(n, n) .< p)
println("Mean degree: $(sum(A3) / n)")

maxtime = 60.0
```

```

big_nl = nightlife(; npeople=n, adj=A3, maxtime=maxtime)
dt = .001
steps = floor(Int, maxdays / dt)
adf3, mdf3 = run!(big_nl, agent_step!, model_step!, steps; adata, mdata);

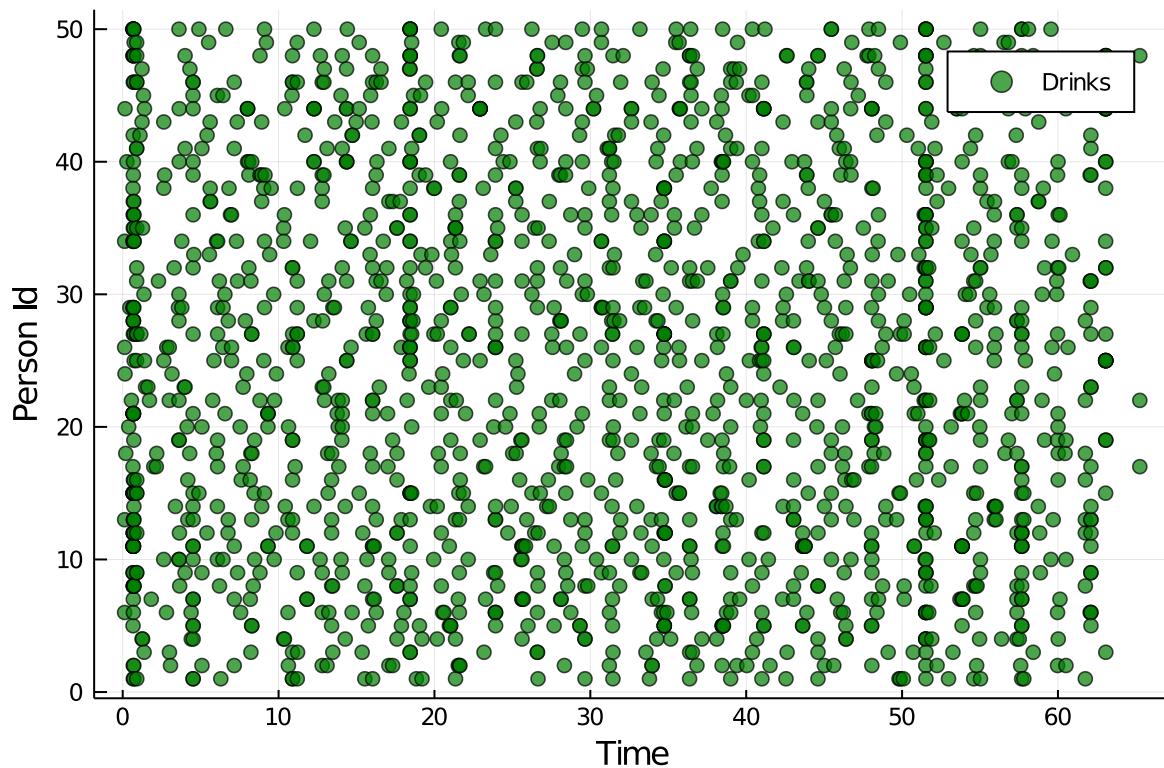
```

Mean degree: 2.96

In [ ]: gplot(SimpleDiGraph(A3))

In [37]: raster\_plot(adf3)

Out[37]:



## Reorder by Community in Social Network

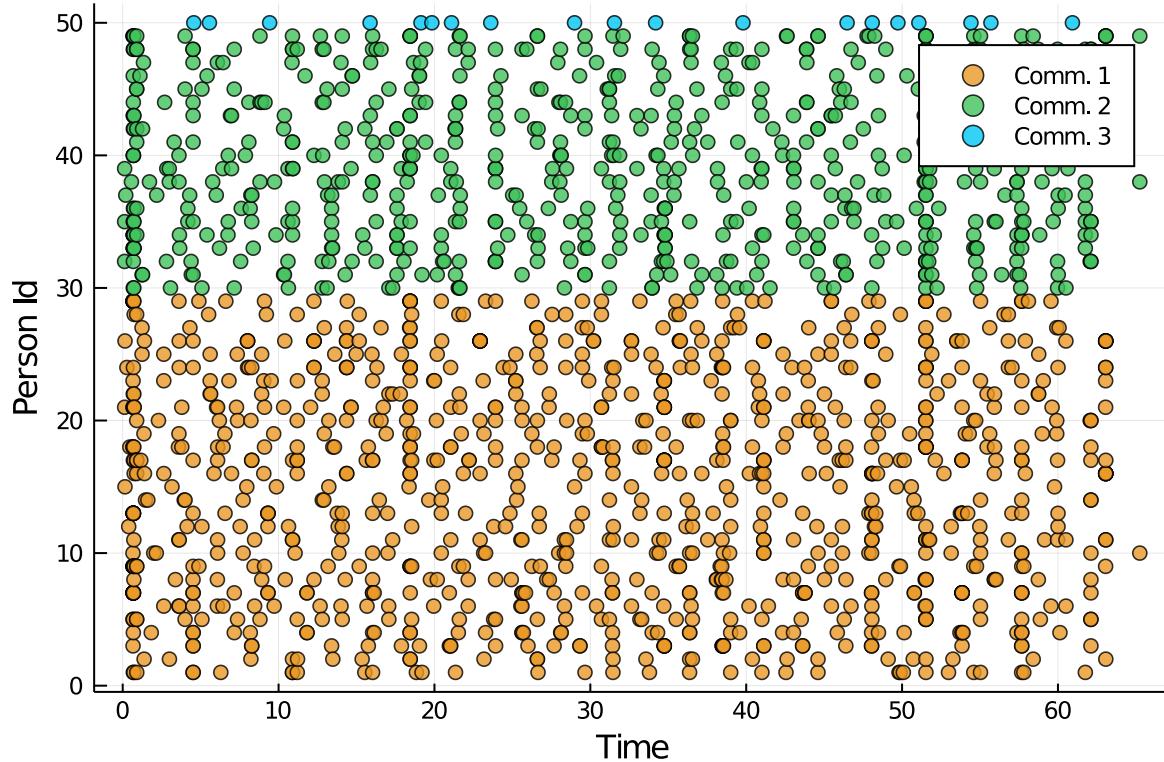
```

In [93]: perm = sortperm(clusters.assignments)
idmapping = zeros(n)
for (i, pi) in enumerate(perm)
    idmapping[pi] = i
end
colors = range(LChab(70, 70, 0), stop=LChab(70, 70, 720), length=11)
p = plot()
for c in 1:nclust
    ids = findall(clusters.assignments .== c)
    clustdf = filter(:id => in(ids), adf3)
    cdrinks = filter(:d => ==(1), clustdf)
    dtimes = cdrinks.step .* dt

    plot!(dtimes, [idmapping[id] for id in cdrinks.id], seriestype=:scatter, al
end
ylabel!("Person Id")
xlabel!("Time")

```

Out[93]:



There appears to be a community effect as we increase mean degree. You can see how at t=35, community 2 (green) all drinks together, but community 1 (orange) does not. There are better ways of measuring synchronization (phase locking value) and we can do a more in depth analysis of this another time.

## Random Graph: Mean Degree Approx. 5

In [38]:

```
n = 50
p = .1
A5 = Float64.(rand(n, n) .< p)
println("Mean degree: $(sum(A5) / n)")

maxtime = 60.0
big_nl = nightlife(; npeople=n, adj=A5, maxtime=maxtime)
dt = .001
steps = floor(Int, maxdays / dt)
adf5, mdf5 = run!(big_nl, agent_step!, model_step!, steps; adata, mdata);
```

Mean degree: 4.6

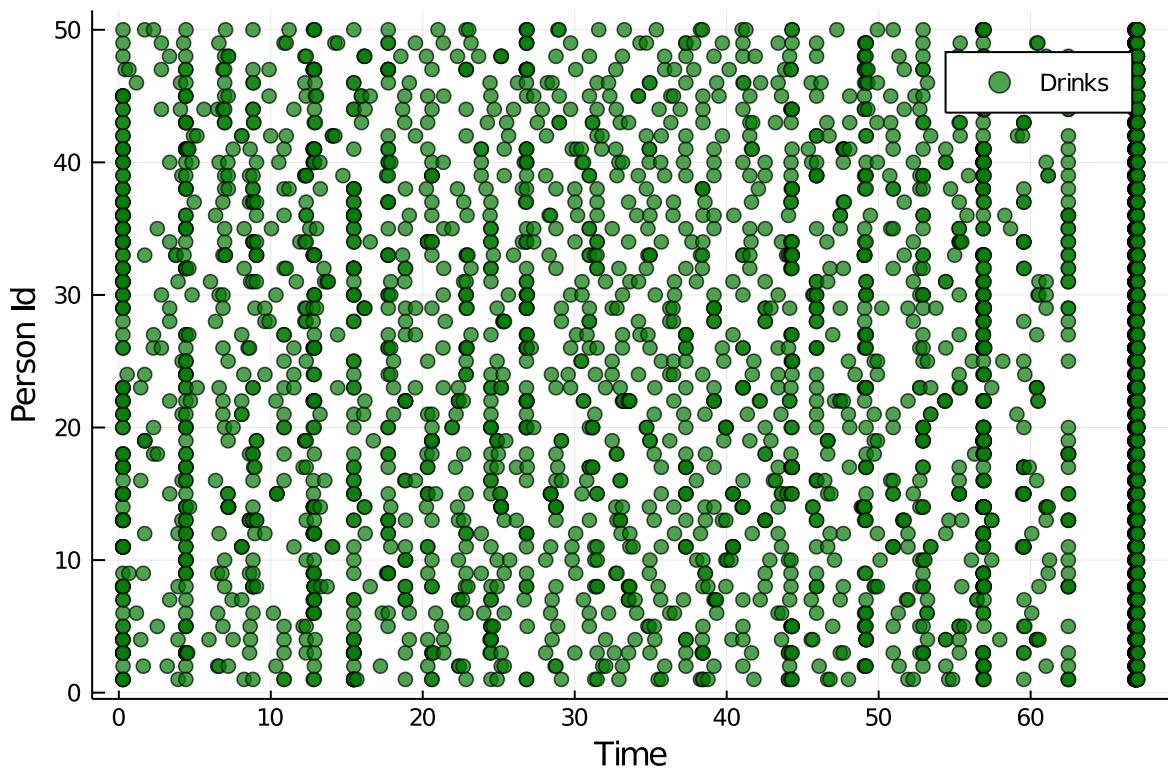
In [ ]:

```
gplot(DiGraph(A5), nodefillc=colorant"pink")
```

In [40]:

```
raster_plot(adf5)
```

Out[40]:



No point in redordering because all nodes are in the same community. However, this plot shows a much higher degree of synchronization than the previous plots, implying that the network structure is really making a difference. Don't ask me about that giant party at time 70. I don't know what happened there.

## Misc Code:

```
In [ ]: """
    eventspikes(event_times::Array{T, 1}, t::Array{T, 1}) where T -> espikes

    Take an array of event times and transform it to an array of event spikes.

    **Parameters**
    1. `event_times`: A list of times when an event occurred
    2. `t`: An equally spaced array of discrete time points

    **Returns**

    `espikes`: An array of zeros and ones corresponding to `t`. Ones appear at event
"""

function eventspikes(event_times::Array{T, 1}, t::Array{T, 1}) where T
    espikes = zeros(size(t)... )
    dt = t[2] - t[1]
    for (i, ti) in enumerate(t)
        if event(event_times, ti, dt)
            espikes[i] = 1
        end
    end
    return espikes
end
```