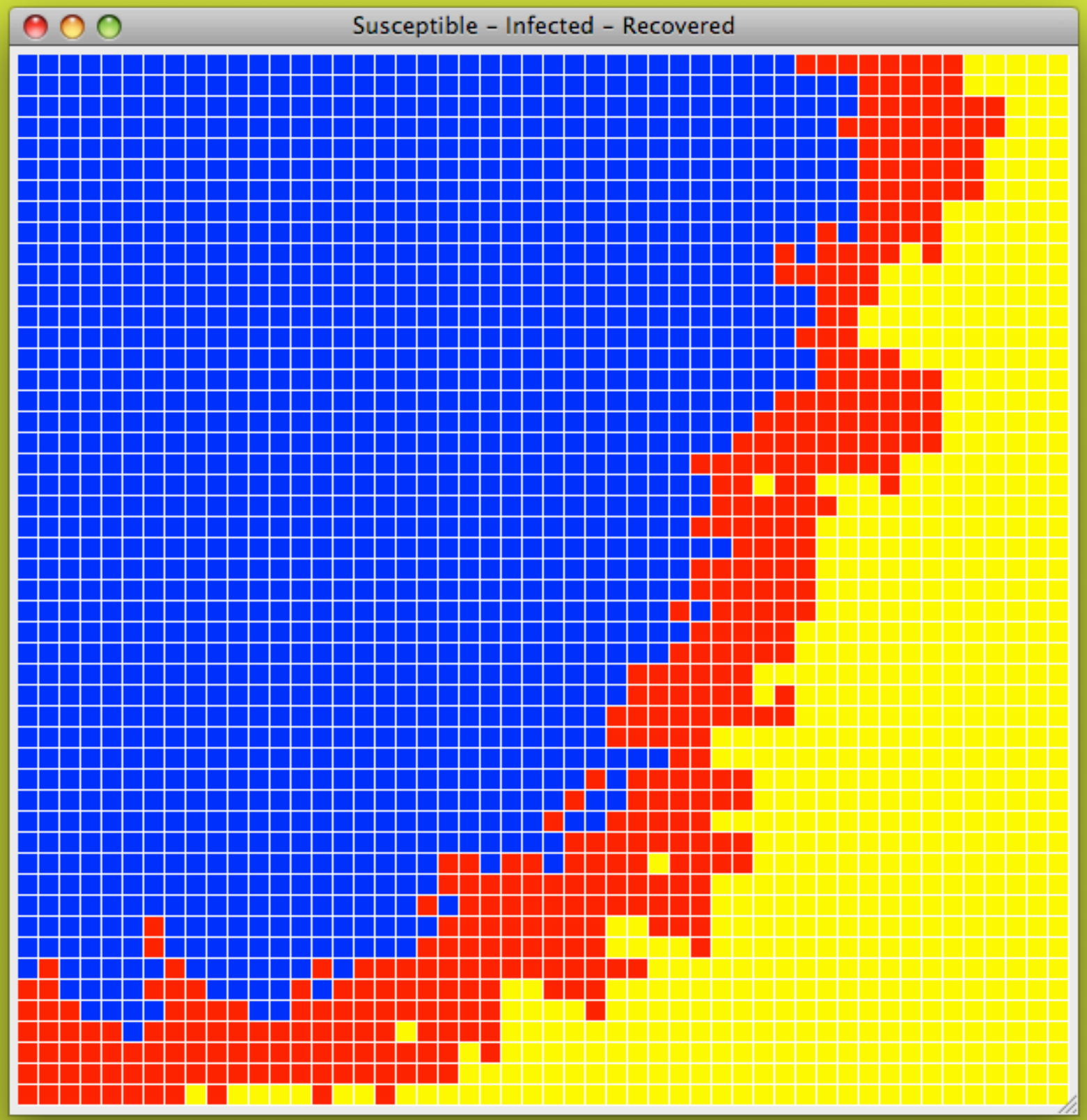
Towards Frabjous

Functionally Reactive Agent-Based Simulation

Oliver Schneider, Christopher Dutchyn,
Nathaniel Osgood

- Domain - Computational Health Models
- Approach - Functional Reactive Programming (FRP)
- Solution - Frabjous


Susceptible - Infected - Recovered

Status Quo - AnyLogic

- Java-based program
- Hybrid specification of models
  - Both GUI and code
- Generates Java code which is then compiled and executed

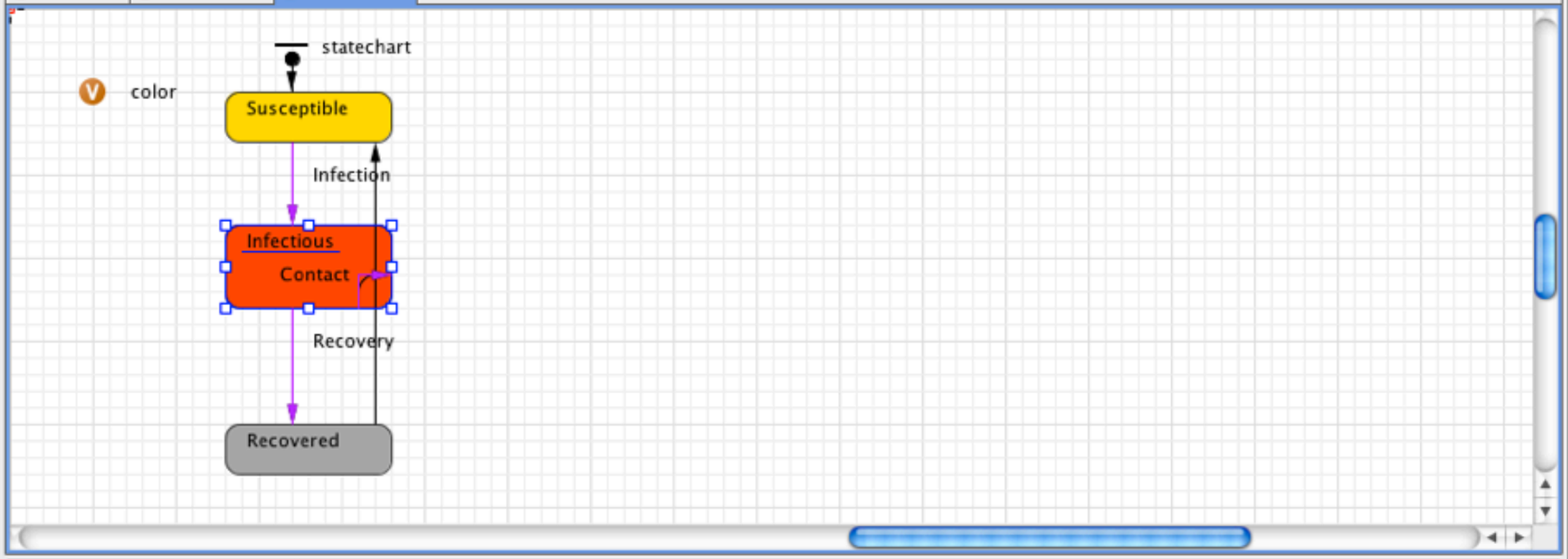
AnyLogic Advanced [EDUCATIONAL USE ONLY]

Get Support

Project

- BassDiffusic
  - Main
  - Person
  - Simulat
- TestModel
- SIR Agent Ba
  - Main
  - Person
  - Simulat
- ESRD_TB
  - Main
  - Person
    - Par
    - Pla
    - Dy
    - Sta
    - Fu
    - Pre
  - Simulat
- Flocks of Bo

Main | Person | Person



Palette

- General
  - Parameter
  - Event
  - Dynamic Event
  - Plain Variable
  - Collection Variable
  - Function
  - Table Function
  - Port
  - Connector
  - Environment
- System Dynamics
- Statechart
- Actionchart
- Analysis
- Presentation
- Connectivity
- Pictures
- Enterprise Library

Palettes...

Probl...

Description	...

Properties | Console

Infectious – State

General

Description

Name: Infectious

Show name | Ignore | Public | Show at runtime

Fill color: orangeRed

Entry action:

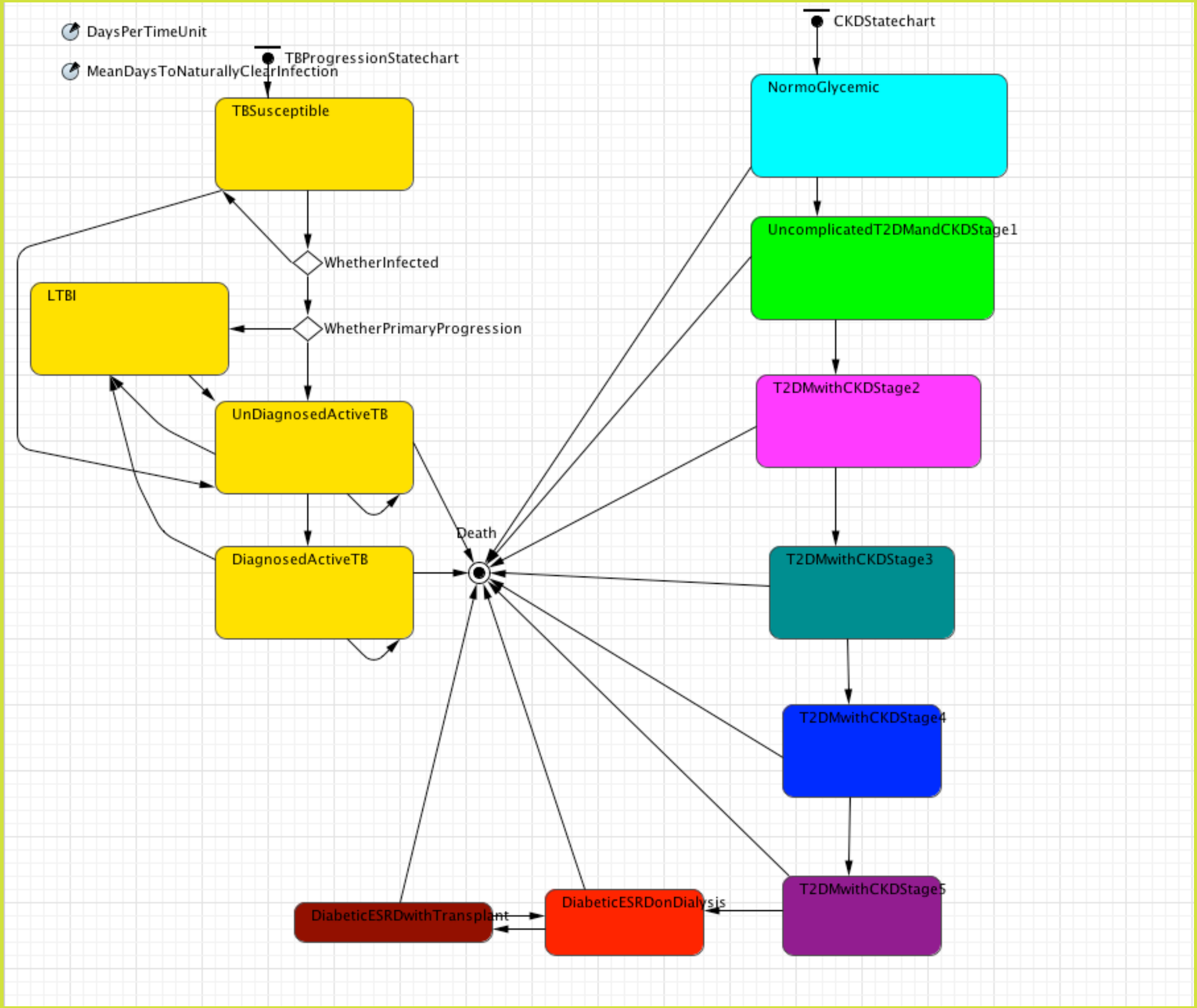
```
get_Main().nInfectious++;
color = RED;
```

Exit action:

```
get_Main().nInfectious--;
```

Infectious – State | Selection | X=459, Y=209

```
87    // States of all statecharts
88
89    public static final short Susceptible = 0;
90    public static final short Infectious = 1;
91    public static final short Recovered = 2;
92
93
94    @Override
95    public String getNameOfState( short _state ) {
96      switch( _state ) {
97        case Susceptible: return "Susceptible";
98        case Infectious: return "Infectious";
99        case Recovered: return "Recovered";
100       default: return super.getNameOfState( _state );
101     }
102   }
103
104   @Override
105   public void enterState( short _state, boolean _destination ) {
106     switch( _state ) {
107       case Susceptible: // (Simple state (not composite))
108         statechart.setActiveState( Susceptible );
109         {
110 get_Main().nSusceptible++;
111 color = GOLD;
112 ;}
113         Infection.start();
114         return;
115       case Infectious: // (Simple state (not composite))
116         statechart.setActiveState( Infectious );
117         {
118 get_Main().nInfectious++;
119 color = RED;
120 ;}
121         Recovery.start();
122         Contact.start();
123         return;
124       case Recovered: // (Simple state (not composite))
125         statechart.setActiveState( Recovered );
126         {
127 get_Main().nRecovered++;
128 color = GRAY;
129 ;}
130         transition.start();
131         return;
132       default:
133         super.enterState( _state, _destination );
134         return;
135     }
136   }
137
138   @Override
139   public void exitState( short _state, Transition _t, boolean _source, Statechart _statechart) {
140     switch( _state ) {
```


DaysPerTimeUnit
MeanDaysToNaturallyClearInfection
TBProgressionStatechart
TBSusceptible
WhetherInfected
LTBI
WhetherPrimaryProgression
UnDiagnosedActiveTB
DiagnosedActiveTB
Death
CKDStatechart
NormoGlycemic
UncomplicatedT2DMandCKDStage1
T2DMwithCKDStage2
T2DMwithCKDStage3
T2DMwithCKDStage4
T2DMwithCKDStage5
DiabeticESRDonDialysis
DiabeticESRDwithTransplant

Computational Models

- Are often long and opaque
- Require expertise in programming

Functional Programming Languages

- Are concise and transparent
- Require expertise in programming

Haskell

```
qsort []     = []
qsort (x:xs) = qsort (filter (< x) xs) ++ [x] ++ qsort (filter (>= x) xs)
```

http://www.haskell.org/

FRP

- Adds "reactivity", which is a sense of time
- Very natural for time-based simulations
- Functions → "Signal Functions"

sir.hs (~/Documents/Research/2009summer/haskell/yampa) – VIM

```
38
39 --init
40 board :: CellGrid
41 board =  [cell (x, y) defRecTime (get_neighbours x y) (startSF x y) | x <- [0..boardSize-1], y <- [0..boardSize-1]] --values
42     where
43         --starting board for testing
44         startSF 0 0 = infected
45         startSF _ _      = susceptible
46         --no wraparound
47         get_neighbours x y = [(x', y') | x' <- [x-1,x,x+1], y' <- [y-1,y,y+1], (x',y') /= (x,y), inBounds x' y']
48         inBounds a b = a >= 0 && a < boardSize && b >= 0 && b < boardSize
49
50
51 -- main logic
52 cell :: Key -> Time -> [Key] -> StateSF -> Cell
53 cell (x,y) recTime neighbours sf = proc input -> do
54         let id = (x,y)
55         let inp = event [] (filter (==id)) input
56         state <- sf -< inp
57         index <- noiseR (0, (length neighbours)-1) (mkStdGen (boardSize * x + y)) -< ()
58         let target = if state == Inf then Event (neighbours!!index) else noEvent
59         returnA -< ( (id, state), target) --placeholder
60
61
62 --states of agents
63 susceptible :: StateSF
64 susceptible = dSwitch (constant Sus &&& (arr (/=[]) >>> edge)) (\_ -> infected)
65
66 infected :: StateSF
67 infected = dSwitch (constant Inf &&& (now () >>> delayEvent defRecTime)) (\_ -> recovered)
68
69 recovered :: StateSF
70 recovered = dSwitch (constant Rec &&& (never >>> delayEvent defRecTime)) (\_ -> susceptible)
71
72
```

38,0-1 41%

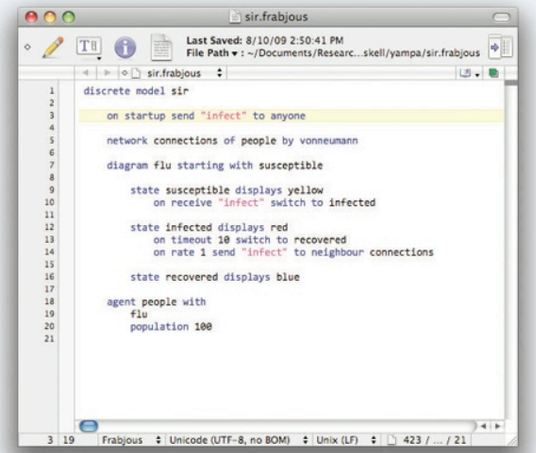
Functional Programming Languages

- Are concise and transparent
- Require expertise in programming

Domain Specific Languages (DSLs)

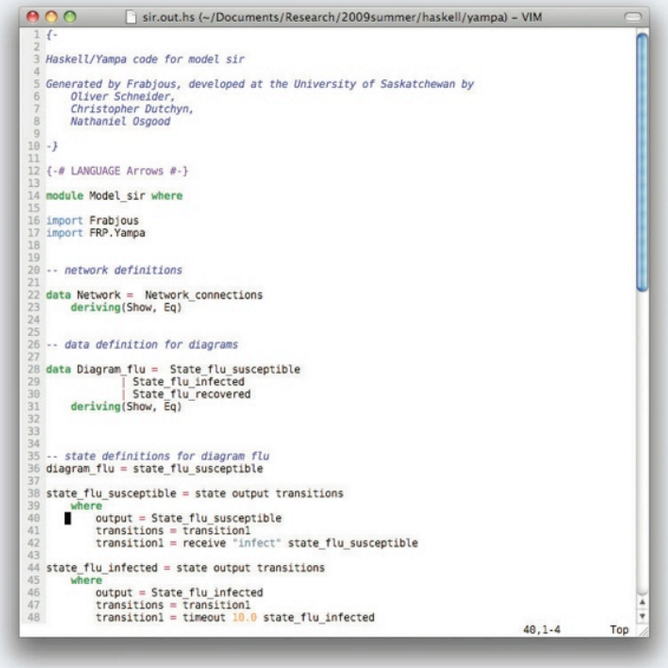
- Are concise and transparent
- Do not require expertise in programming
- Are easily developed from functional languages

The Frabjous System



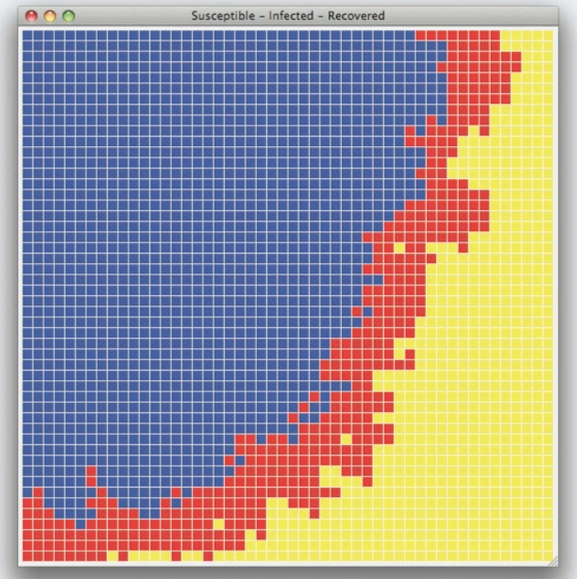
Frabjous Code

Frabjous



Generated Haskell/Yampa Code

GHC



Executable Simulation

sir.frabjous

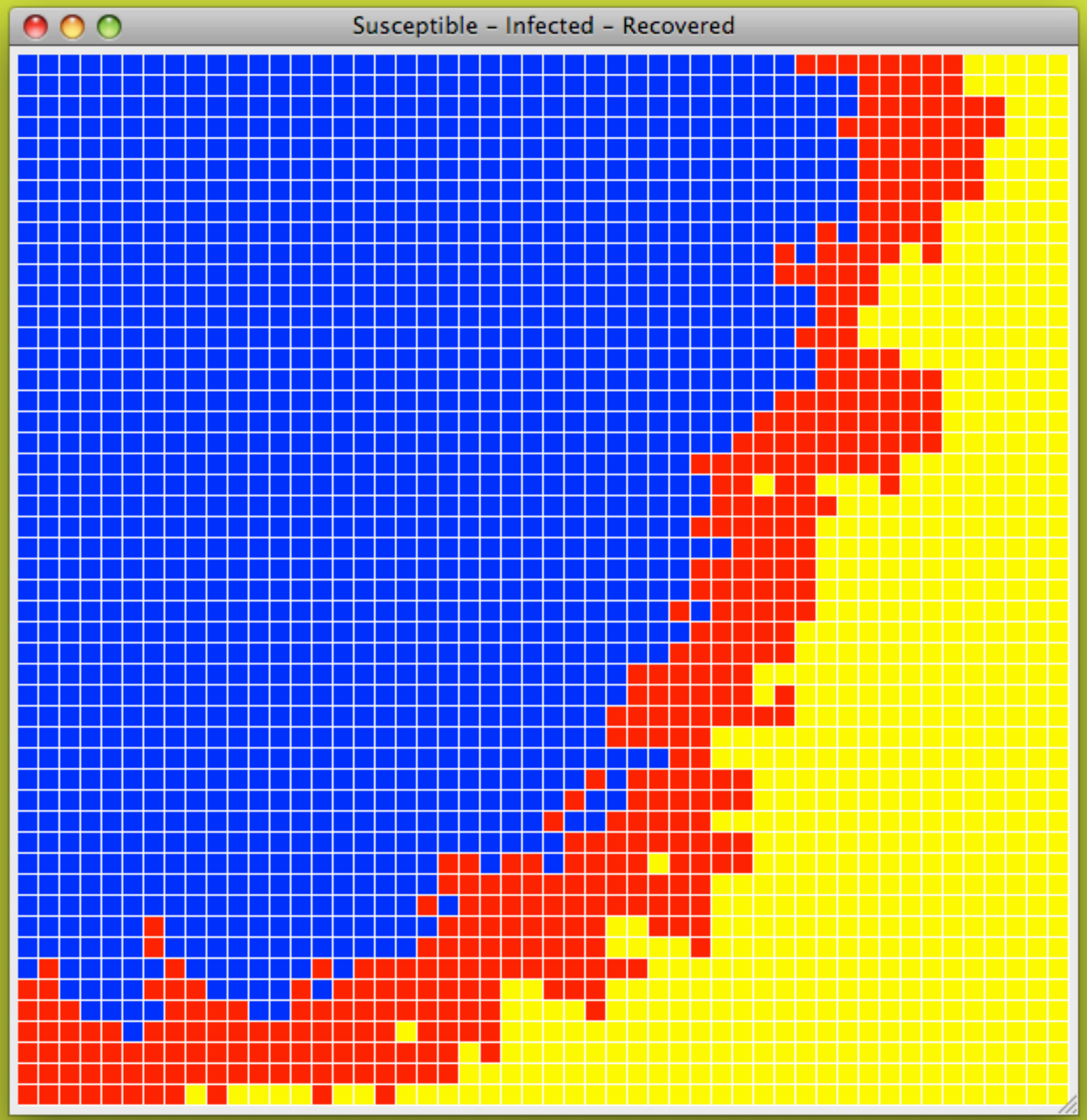
Last Saved: 8/10/09 2:50:41 PM
File Path ▾ : ~/Documents/Researc...skell/yampa/sir.frabjous

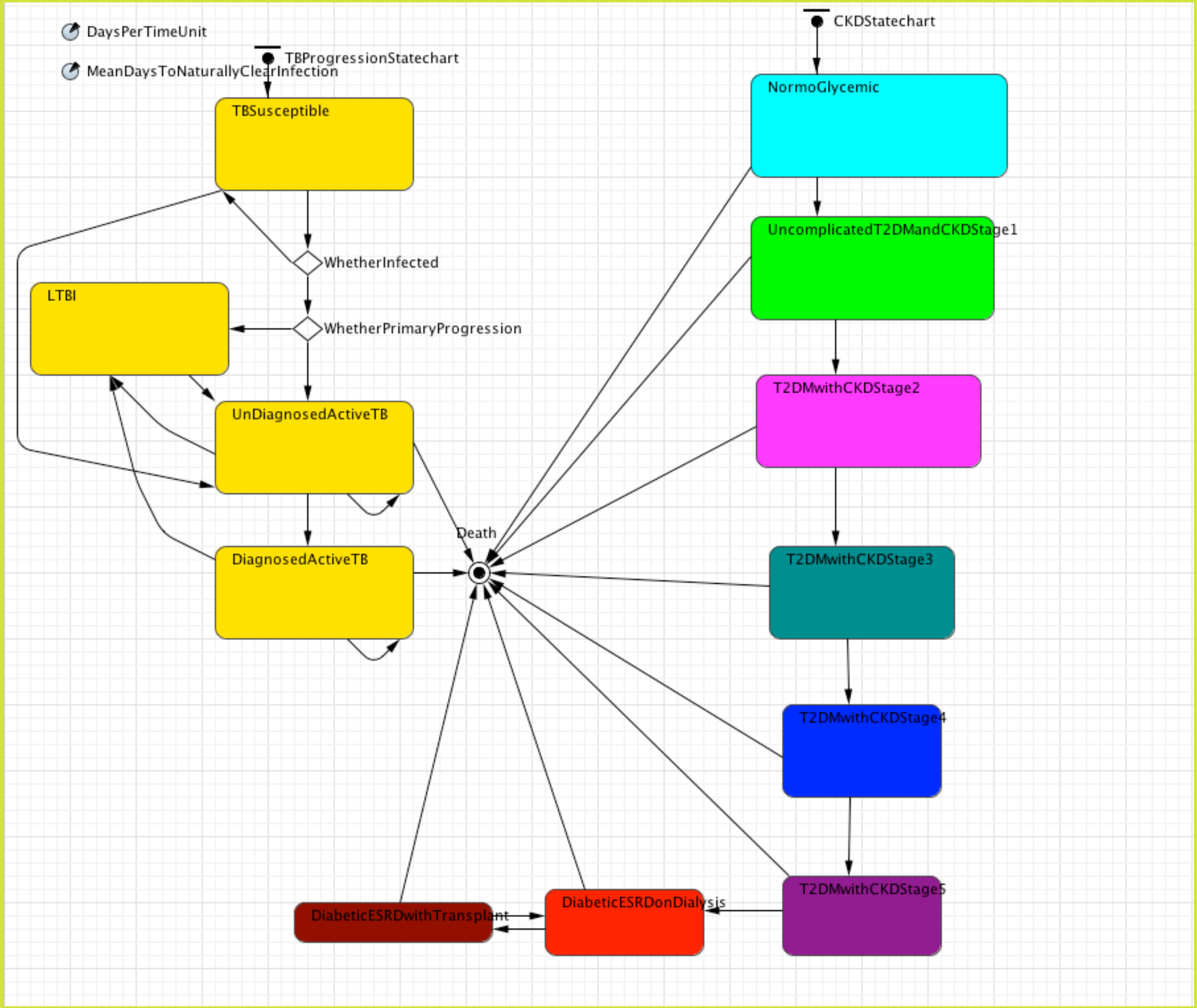
sir.frabjous

```
1  discrete model sir
2
3      on startup send "infect" to anyone
4
5      network connections of people by vonneumann
6
7      diagram flu starting with susceptible
8
9          state susceptible displays yellow
10             on receive "infect" switch to infected
11
12         state infected displays red
13             on timeout 10 switch to recovered
14             on rate 1 send "infect" to neighbour connections
15
16         state recovered displays blue
17
18     agent people with
19         flu
20         population 100
21
```

3 | 19 | Frabjous | Unicode (UTF-8, no BOM) | Unix (LF) | 423 / ... / 21

```haskell
{-# LANGUAGE Arrows #-}

module Model_sir where

import Frabjous
import FRP.Yampa


-- network definitions

data Network =  Network_connections
    deriving(Show, Eq)


-- data definition for diagrams

data Diagram_flu =  State_flu_susceptible
                  | State_flu_infected
                  | State_flu_recovered
    deriving(Show, Eq)



-- state definitions for diagram flu
diagram_flu = state_flu_susceptible

state_flu_susceptible = state output transitions
    where
        output = State_flu_susceptible
        transitions = transition1
        transition1 = receive "infect" state_flu_susceptible

state_flu_infected = state output transitions
    where
        output = State_flu_infected
        transitions = transition1
        transition1 = timeout 10.0 state_flu_infected
```


Susceptible - Infected - Recovered


DaysPerTimeUnit
MeanDaysToNaturallyClearInfection
TBProgressionStatechart
TBSusceptible
WhetherInfected
LTBI
WhetherPrimaryProgression
UnDiagnosedActiveTB
DiagnosedActiveTB
Death
CKDStatechart
NormoGlycemic
UncomplicatedT2DMandCKDStage1
T2DMwithCKDStage2
T2DMwithCKDStage3
T2DMwithCKDStage4
T2DMwithCKDStage5
DiabeticESRDonDialysis
DiabeticESRDwithTransplant

esrd.frabjous

esrd.frabjous

```
continuous model esrd_tb

    on startup send "tbforce" to anyone

    agent people with
            tb_prog
            ckd_prog
            population 500

    network connections of people by random 0.3

    diagram tb_prog starting with susceptible
        var reactivation = 1
        state susceptible
            on receive "tbforce" switch to active
            on receive "tbcontact" switch to latent
        state active
            on timeout 30 switch to diagnosed
            on rate 40 switch to latent
            on rate 2 send "tbcontact" to neighbour connections
        state diagnosed
            on timeout 20 switch to latent
        state latent
            on rate reactivation switch to active

    diagram ckd_prog
        state stage1
            on rate 0.0074 switch to stage2
        state stage2
            on rate 0.2023 switch to stage3
        state stage3
            on rate 0.1245 switch to stage4
        state stage4
            on rate 0.0629 switch to stage5
        state stage5
            on rate 0.1015 switch to stage6
        state stage6
            on rate 0.1648 switch to stage7
        state stage7
            on rate 0.0500 switch to stage8
        state stage8
            on rate 0.0970 switch to stage7


    confound tb_prog ckd_prog
        on enter ckd_prog:stage4 set tb_prog:reactivation to 7
        on enter ckd_prog:stage8 set tb_prog:reactivation to 50
        on leave ckd_prog:stage8 set tb_prog:reactivation to 7
```

50 | 1 | Frabjous | Unicode (UTF-8, no BOM) | Unix (LF) | 1,219 / … / 50

In Conclusion

- Used FRP to rapidly developed a DSL (Frabjous)
- Frabjous is transparent and concise
- Frabjous does not require programming expertise
- Compiles into legible and easily editable Haskell code

Contact

Oliver Schneider.....o.schneider@usask.ca

Christopher Dutchyn.....dutchyn@cs.usask.ca

Nathaniel Osgood.....osgood@cs.usask.ca