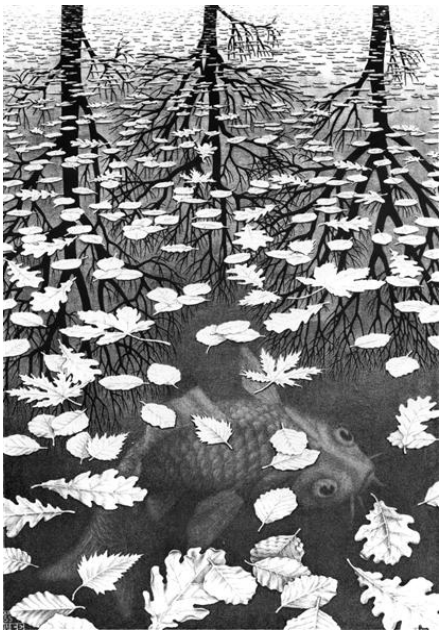
3Worlds, a simulation platform for ecosystem modelling

Jacques Gignoux, Ian D. Davies, Shayne R. Flint

Appendices



'Three Worlds', by M.C. Escher (1955).
reprinted from wikipedia.

Appendix 1

Aspect oriented-thinking

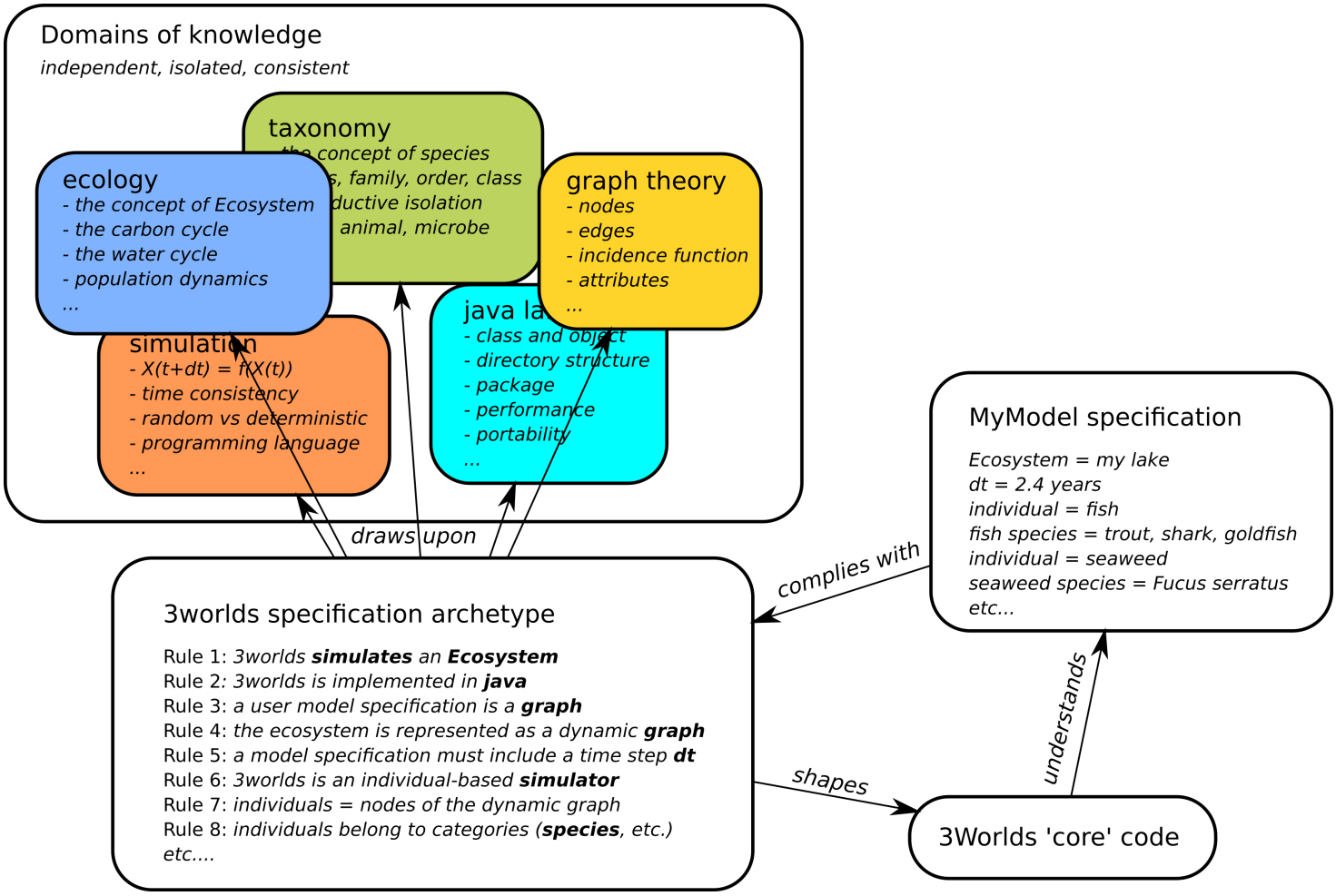


Illustration of how aspect-oriented thinking (AOT) is harnessed to produce the 3Worlds software. Independent domains of knowledge (concepts, rules, data) relevant to construct a simulator for ecosystem are identified through discussion with experts in those domains. Within those domains, the only knowledge relevant for our particular purpose (building an ecosystem simulator) are extracted to build assembly and interaction rules to reach this purpose (this is the specification archetype*). The 3Worlds application enable users to specify any particular model of an ecosystem complying with the rules captured in the archetype, including the generation of computer code and the associate requested parameter list.*

The 3Worlds specification archetype is too long for reproducing here. It can be found in the **tw-core** library, under the directory **au.edu.anu.twcore.archetype.tw** (***.ugt** files).

The rules to write an archetype are themselves captured in a « meta » archetype, i.e. an archetype for archetypes (this is part of the **aot** library). This is how this file looks like:

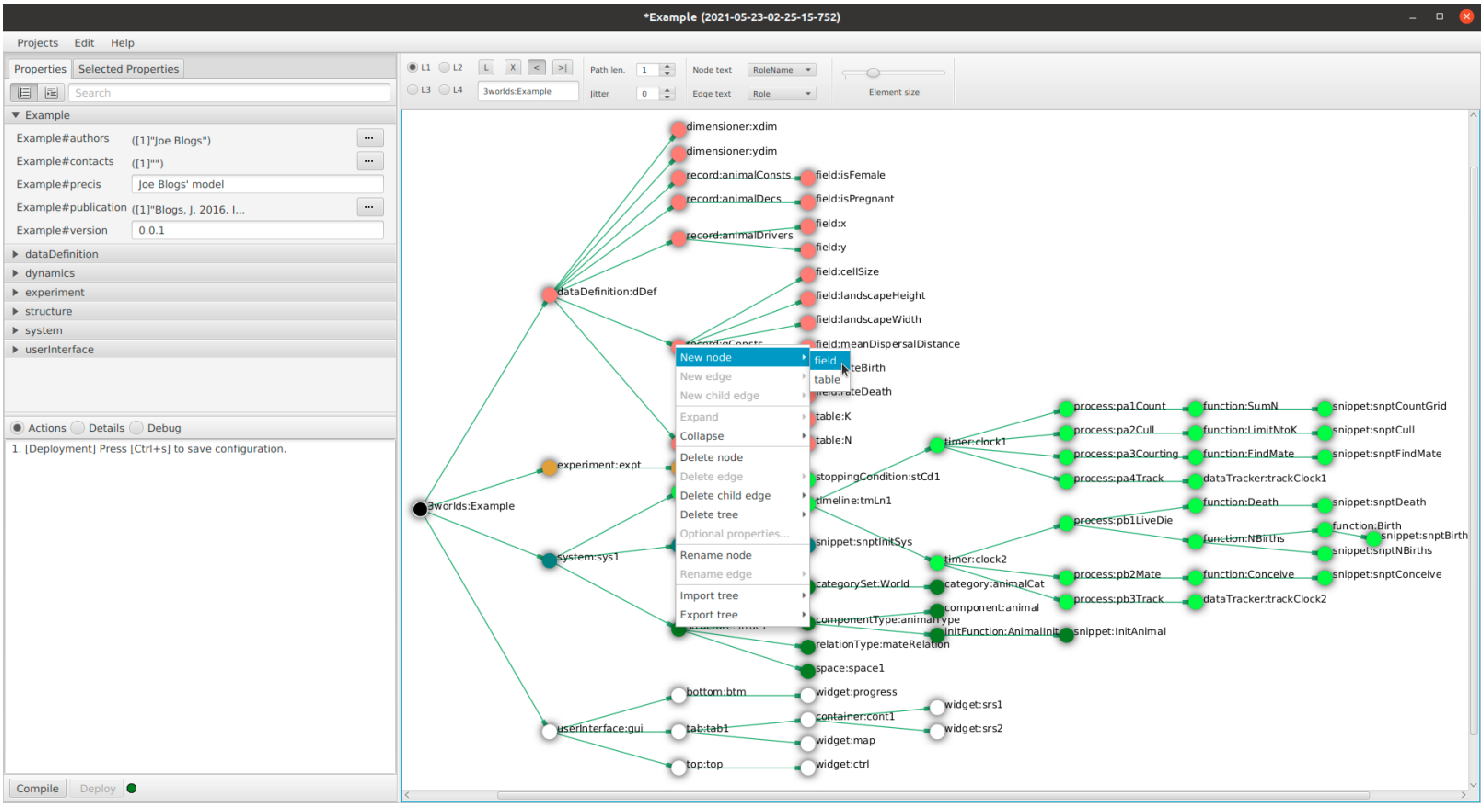
```
tree // Archetype for all archetypes - an archetype is a tree!
//=========================================================//
// CAUTION: DO NOT EDIT THIS FILE !                        //
// it is the root of Aspect-Oriented Thinking              //
// disastrous consequences may follow any change           //
//=========================================================//

  // the scope for node ids coming from this file
  scope =            String("AOT-archetype")
  // labels used in this file
  hasNode =          String("au.edu.anu.rscs.aot.archetype.NodeSpec")
  hasProperty =      String("au.edu.anu.rscs.aot.archetype.PropertySpec")
  hasEdge =          String("au.edu.anu.rscs.aot.archetype.EdgeSpec")
  mustSatisfyQuery = String("au.edu.anu.rscs.aot.archetype.ConstraintSpec")
  archetype =        String("au.edu.anu.rscs.aot.archetype.ArchetypeRootSpec")

//---------------------------------------------------------
// 1 this is an archetype
//---------------------------------------------------------
archetype ArchetypeForArchetypes

  //---------------------------------------------------------
  // 2 this archetype does not accept any other specification
  //---------------------------------------------------------
  exclusive = Boolean(true)

  //---------------------------------------------------------
  // 3 an archetype starts with a root node of class *archetype*
  //---------------------------------------------------------
  hasNode ArchetypeRootSpec
    isOfClass = String("archetype")
    multiplicity = IntegerRange("1..1")
    hasParent = StringTable(([1]""))

    //---------------------------------------------------------
    // 4 the "archetype" node has an *exclusive* property
    // which tells if other definitions than those contained in the tree
    // are acceptable or not
    // absence of the property means non-exclusive archetype
    //---------------------------------------------------------
    hasProperty exclusivePropertySpec
      hasName = String("exclusive")
      multiplicity = IntegerRange("0..1")
      type = String("Boolean")

  //---------------------------------------------------------
  // 5 specification for node specifications:
  // a node with class *hasNode*, child of *archetype*
  //---------------------------------------------------------
  hasNode NodeSpec
    isOfClass = String("hasNode")
    hasParent = StringTable(([1]"archetype:"))
    multiplicity = IntegerRange("0..*")

    //---------------------------------------------------------
    // 6 a node spec may specify the node class (formerly label)
    // if not present then any label is suitable and will be
    // be lost during the tree import process, a default
    // class (eg TreeNode) will be used to instantiate the node
    //---------------------------------------------------------
    hasProperty NodeClassSpec
      hasName = String("isOfClass")
      type = String("String")
      multiplicity = IntegerRange("0..1")

    //---------------------------------------------------------
    // 7 a node spec must specify a list of valid parents for the node
    // for the root node, write hasParent=StringTable(([1]""))
    // single-level Node references can be used, e.g. NodeClass:NodeId
    // or NodeClass: or :NodeId
    // NOTE: This means if a sub archetype parent table differs from the class
    // archetype we must use a ParentClassQuery to constrain the options
    //---------------------------------------------------------
```

```
  hasProperty NodeParentSpecJe mets e
    hasName = String("hasParent")
    type = String("StringTable")
    multiplicity = IntegerRange("1..1")

  //--------------------------------------------------------
  // 8 a node spec may specify the node id (formerly name)
  //--------------------------------------------------------
  hasProperty NodeIdSpec
    hasName = String("hasId")
    type = String("String")
    multiplicity = IntegerRange("0..1")

  //--------------------------------------------------------
  // 9 a node spec optionally has a *multiplicity*  property
  // defaults to 0..* if not set
  //--------------------------------------------------------
  hasProperty NodeMultiplicitySpec
    hasName = String("multiplicity")
    type = String("IntegerRange")
    multiplicity = IntegerRange("0..1")

//--------------------------------------------------------
// 10 specification for constraint specifications
// constraints can apply to node, edge or property specifications
//--------------------------------------------------------
hasNode ConstraintSpec
  isOfClass = String("mustSatisfyQuery")
  hasParent = StringTable(([3]"hasNode:","hasProperty:","hasEdge:"))
  multiplicity = IntegerRange("0..*")

  //--------------------------------------------------------
  // 11 a constraint spec uses a Query class descendant
  // which fully qualified name must be provided here
  //--------------------------------------------------------
  hasProperty ConstraintClassSpec
    hasName = String("className")
    type = String("String")
    multiplicity = IntegerRange("1..1")

//--------------------------------------------------------
// 12 specification for property specifications
// properties apply to node or edge specifications
//--------------------------------------------------------
hasNode PropertySpec
  isOfClass = String("hasProperty")
  hasParent = StringTable(([2]"hasNode:","hasEdge:"))
  multiplicity = IntegerRange("0..*")

  //--------------------------------------------------------
  // 13 a property spec must specify a name for the property
  //--------------------------------------------------------
  hasProperty PropertyNameSpec
    hasName = String("hasName")
    type = String("String")
    multiplicity = IntegerRange("1..1")

  //--------------------------------------------------------
  // 14 a property spec must specify a type for the property
  //--------------------------------------------------------
  hasProperty PropertyTypeSpec
    hasName = String("type")
    type = String("String")
    multiplicity = IntegerRange("1..1")

  //--------------------------------------------------------
  // 15 a property spec must specify a multiplicity for the property
  // NB valid values are 0..1 and 1..1
  //--------------------------------------------------------
  hasProperty PropertyMultiplicitySpec
    hasName = String("multiplicity")
    type = String("IntegerRange")
    multiplicity = IntegerRange("1..1")

//--------------------------------------------------------
// 16 specification for edge specifications
//--------------------------------------------------------
```

```
hasNode EdgeSpec
  isOfClass = String("hasEdge")
  hasParent = StringTable(([1]"hasNode:"))
  multiplicity = IntegerRange("0..*")

  //-------------------------------------------------------
  // 17 an edge spec may specify the edge id (formerly name)
  //-------------------------------------------------------
  hasProperty EdgeIdSpec
    hasName = String("hasId")
    type = String("String")
    multiplicity = IntegerRange("0..1")

  //-------------------------------------------------------
  // 18 an edge spec may specify the edge class (formerly label)
  // if not present then any label is suitable and will be
  // be lost during the tree import process, a default
  // class (eg Edge) will be used to instantiate the edge
  //-------------------------------------------------------
  hasProperty EdgeClassSpec
    hasName = String("isOfClass")
    type = String("String")
    multiplicity = IntegerRange("0..1")

  //-------------------------------------------------------
  // 19 an edge spec must specify an end node (the start node is
  // its parent node specification)
  //-------------------------------------------------------
  hasProperty EdgeEndNodeSpec
    hasName = String("toNode")
    type = String("String")
    multiplicity = IntegerRange("1..1")

  //-------------------------------------------------------
  // 20 an edge spec optionally has a *multiplicity*  property
  // defaults to 0..* if not set
  //-------------------------------------------------------
  hasProperty EdgeMultiplicitySpec
    hasName = String("multiplicity")
    type = String("IntegerRange")
    multiplicity = IntegerRange("0..1")
```

Appendix 2

Screen copy of **ModelMaker**

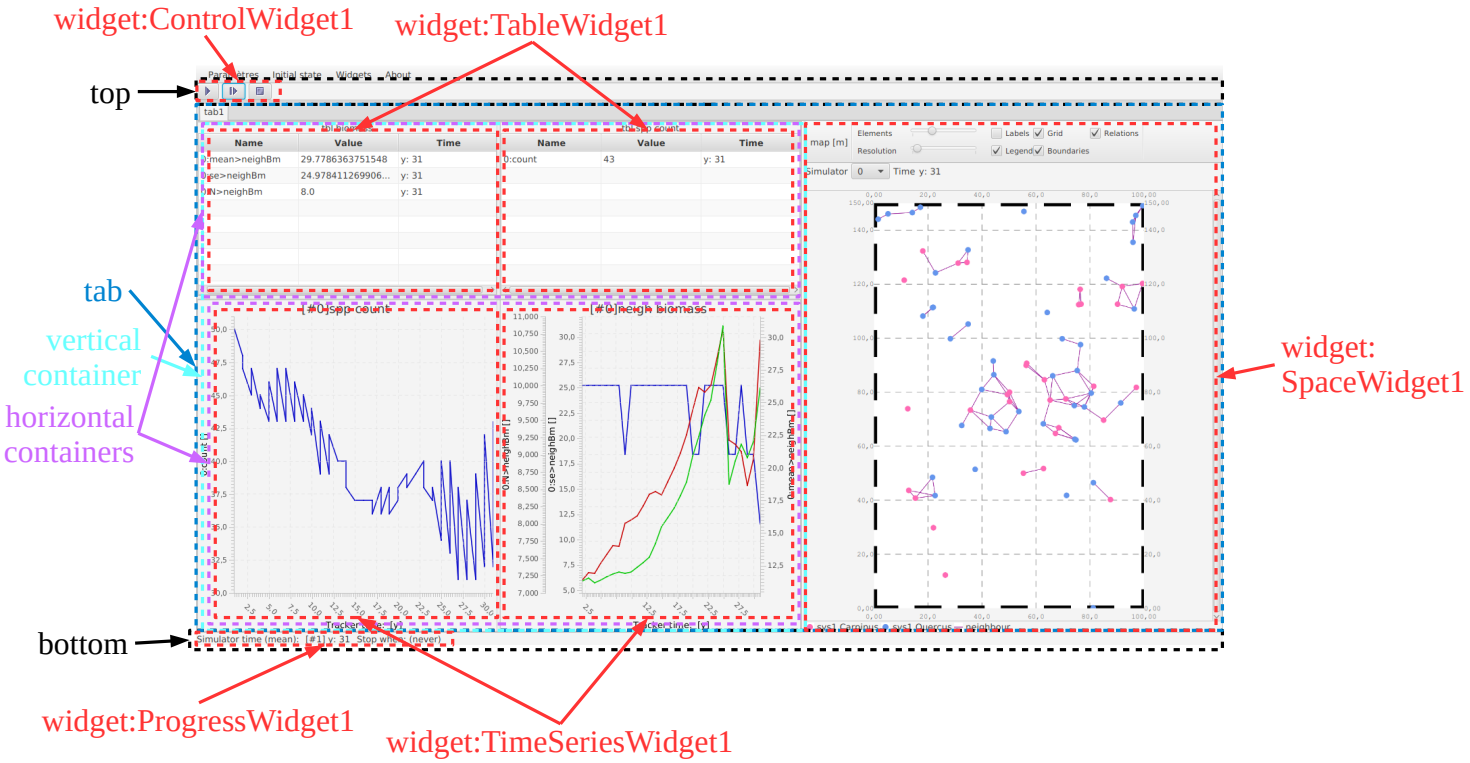


The right panel is a **graph editor** showing the **configuration graph** representing the model specification. Every node or edge is defined in compliance with the **3Worlds specification archetype** (Appendix 1): when modellers select a node, the editor filters which types of nodes and edges can be linked to it by checking the archetype rules. The different sub-tree colors match those shown in Fig.1. Green lines show parent-child relations (i.e., the hierarchical tree structure of a 3Worlds configuration), while red lines (not shown here for clarity) show cross-links between nodes.

The top left panel shows all the **properties** associated to a node of the configuration graph, as permitted by the archetype.

The bottom left panel display error messages that occur during graph editing, as a result of checking the configuration with the archetype. At the bottom, a little light must become green for the ‘deploy’ button to be enabled and allow to launch **ModelRunner**.

Screen copy of **ModelRunner**

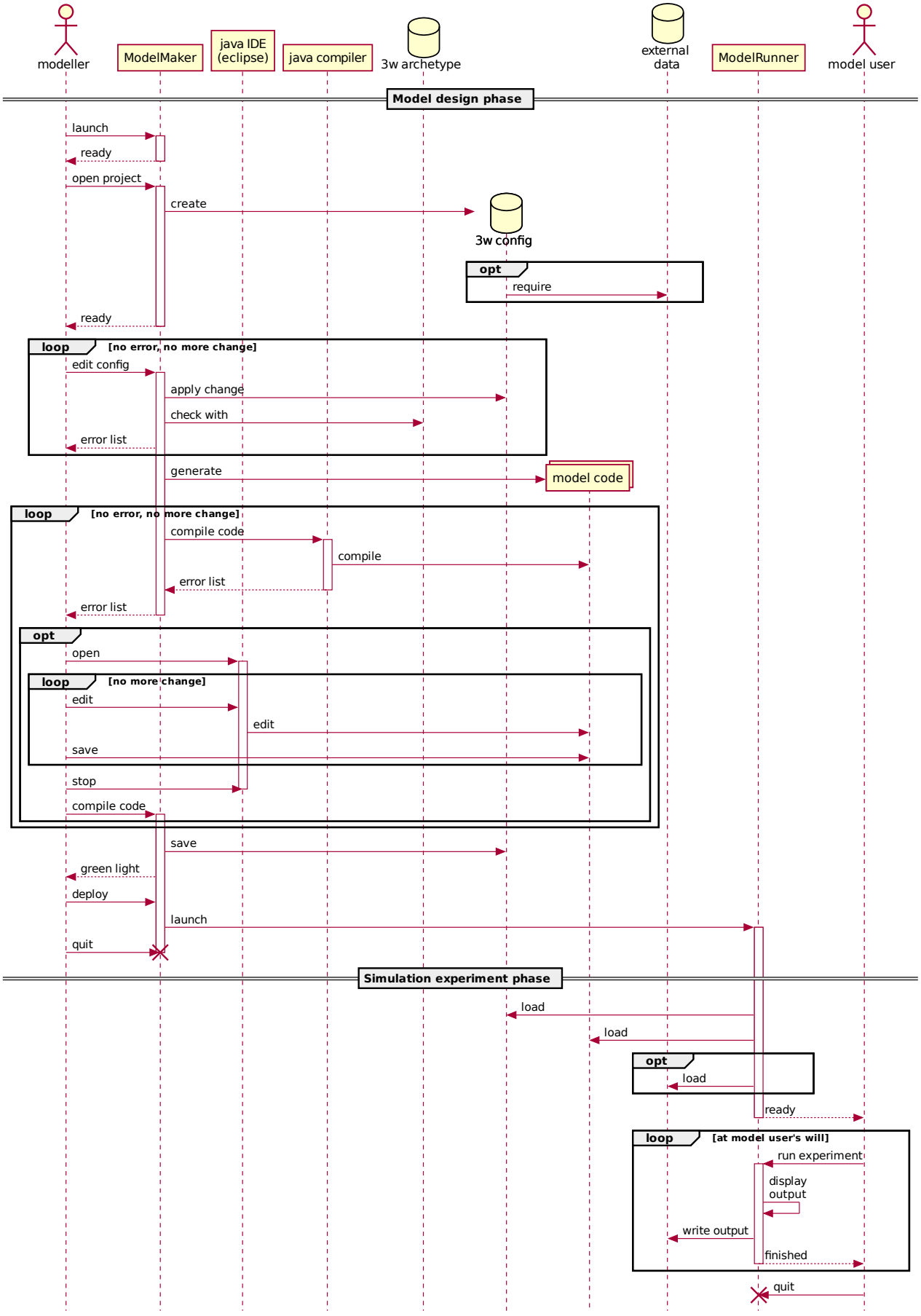


The user interface of **ModelRunner** is highly configurable from **ModelMaker** (this the white sub-tree). Some nodes in the **configuration graph** represent visual elements of the **ModelRunner** window: **tabs**, **containers**, and **widgets**. A library of various widget is proposed to perform various tasks: control the simulation experiment (‘ControlWidget’), display tables of numbers (‘TableWidget’), show time series of output data (‘TimeSeriesWidget’), show a map of the simulated ecosystem if relevant (‘SpaceWidget’), display experiment progress (‘ProgressWidget’), and many more. The code architecture allows infinite expansion of the widget library.

The code of the **ModelMaker** and **ModelRunner** applications is centralized in the **tw-apps** library, while the code for their graphical interfaces is in the **tw-uifx** library.

Appendix 3

UML sequence diagram of a 3worlds modelling session.

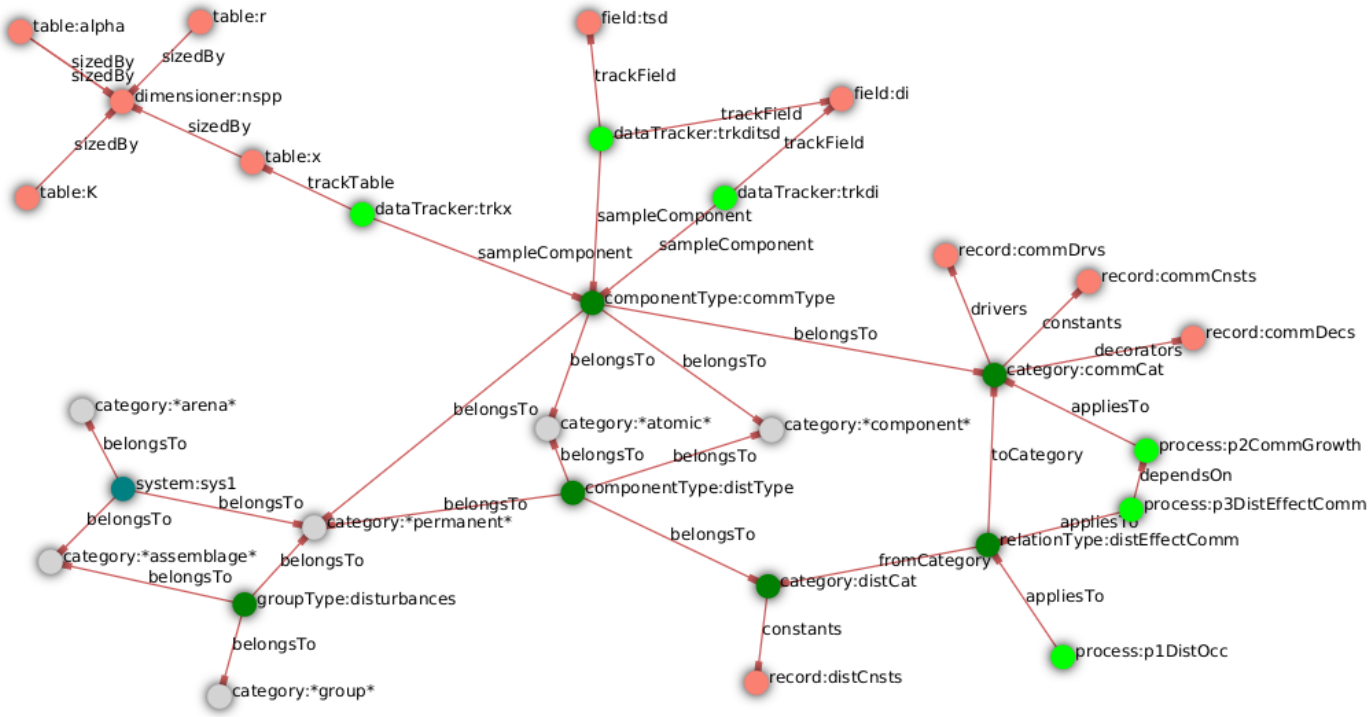


Appendix 4

Example of a simple model: I.D.H.

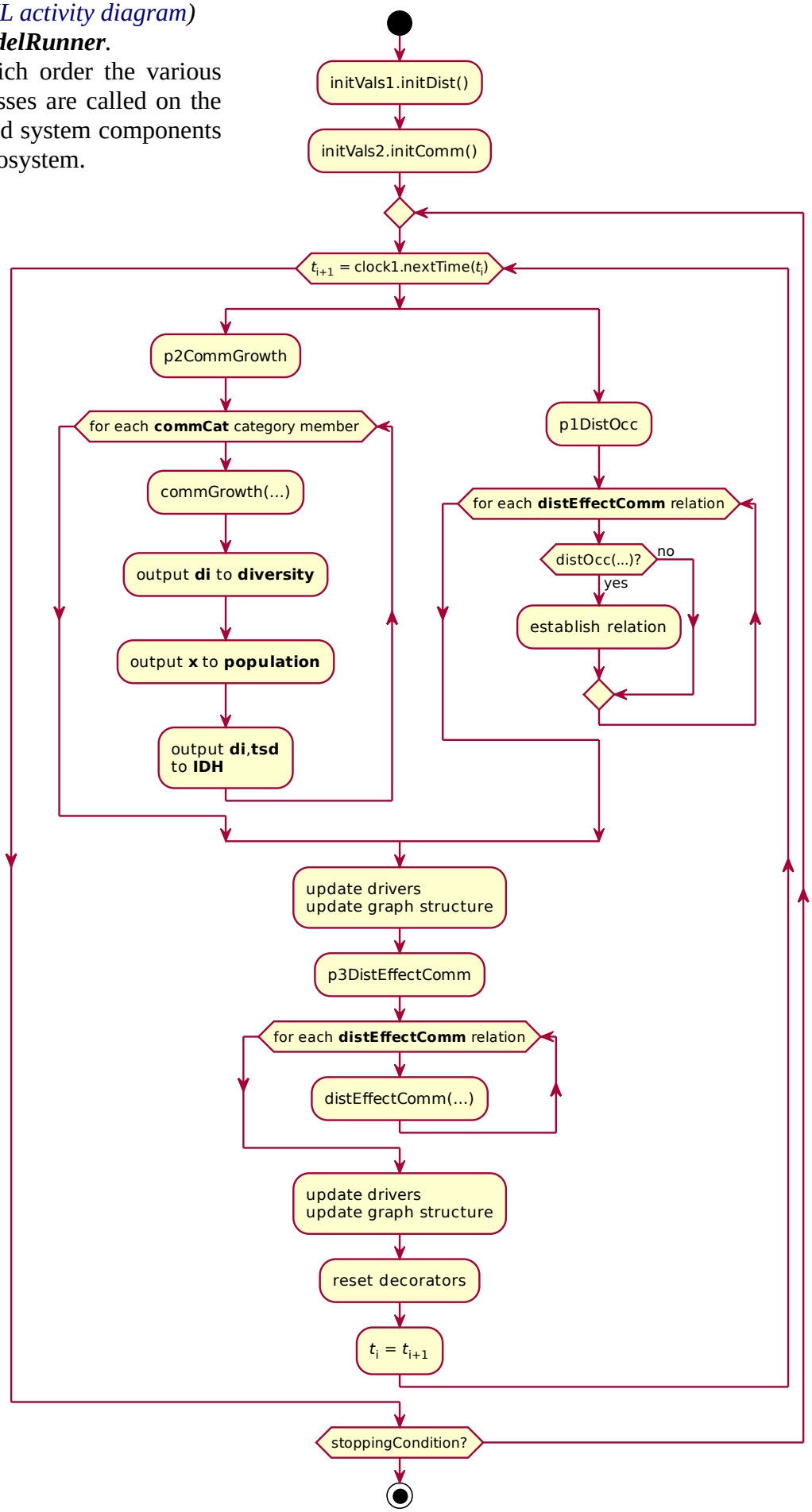
This model is part of the 3Worlds *tutorial* library (**tw-models**). cf. text for general description.

Screen copy of the configuration graph showing the tree structure



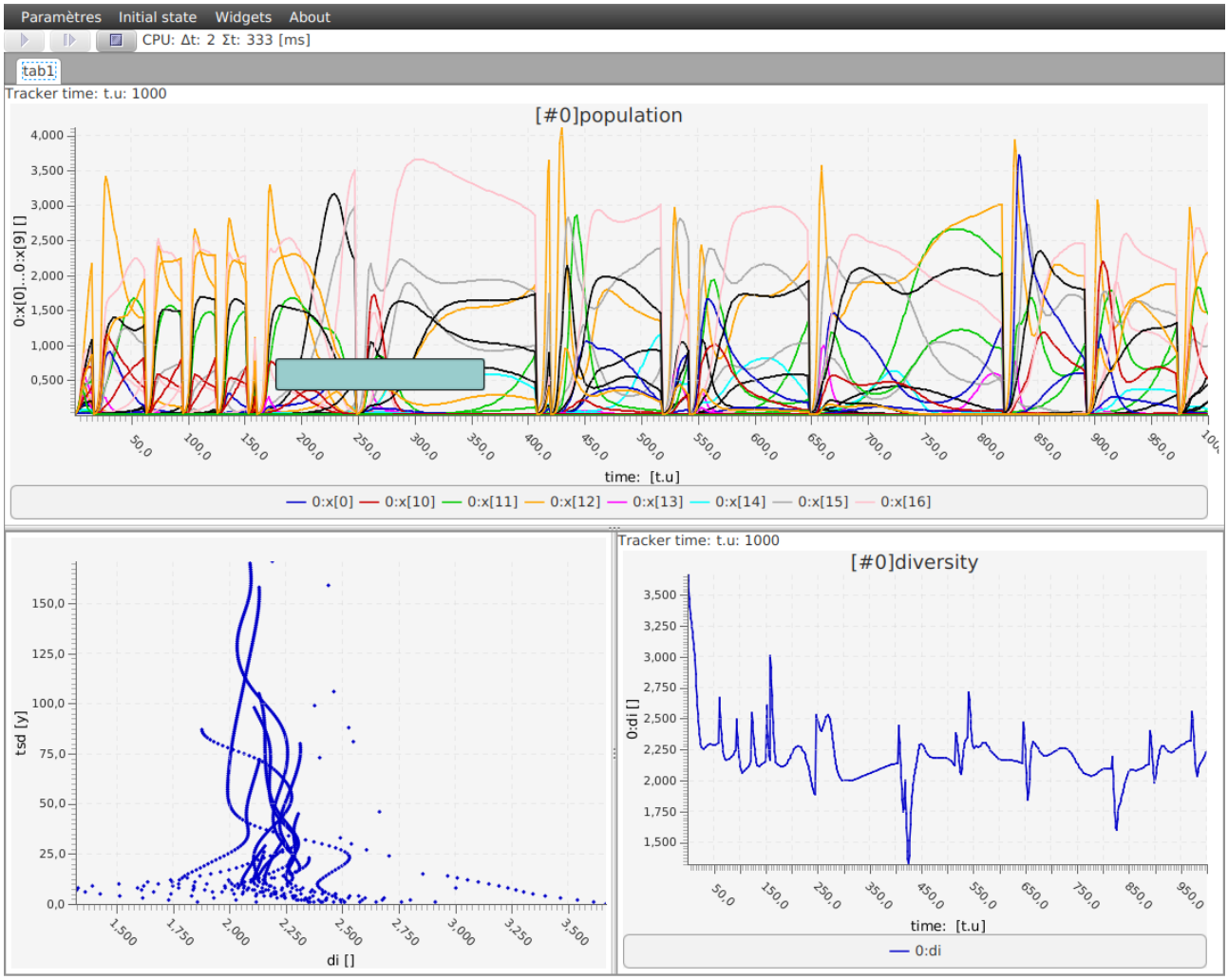
Screen copy of the configuration graph showing the cross-links between the tree nodes

*Flow chart (as UML activity diagram) as exported by **ModelRunner**.*
This shows in which order the various user-defined processes are called on the various user-defined system components representing the ecosystem.



Configuration graph metrics:

Metric	Value
1 #Nodes	27
2 #Edges	25
3 #Properties	84
4 configuration size (1+2+3)	136
5 #Drivers	41
6 #Constants	122
7 #Decorators	1
8 #ComponentTypes	2
9 #RelationTypes	1
10 #GroupTypes	0
11 #Lines of code	47
12 #Complexity (kB)	14,060



Screen copy of ModelRunner. Top panel, biomasses of the 40 species (1 sp. per color) as a function of time. Bottom left panel, relationship between diversity index and time interval between two disturbance events. Bottom right panel, Shannon diversity index as a function of time.

The java code specific to the model. Generated parts are in black, user-defined code is in red:

```
/*
 *                    *** 3Worlds - A software for the simulation of ecosystems ***
 *                    *                                                          *
 *                    *        by:  Jacques Gignoux - jacques.gignoux@upmc.fr    *
 *                    *             Ian D. Davies   - ian.davies@anu.edu.au      *
 *                    *             Shayne R. Flint - shayne.flint@anu.edu.au    *
 *                    *                                                          *
 *                    *         http:// ???                                      *
 *                    *                                                          *
 *                    ************************************************************
 * This program is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program.  If not, see <http://www.gnu.org/licenses/>.
 * ****************************************************************************
*/

package code.sys1;

import au.edu.anu.rscs.aot.collections.tables.DoubleTable;
import au.edu.anu.twcore.ecosystem.runtime.biology.DecisionFunction;
import java.util.Random;
import code.sys1.generated.*;
// ---- Import insert Begin-->
import static java.lang.Math.*;
// ---- Import insert End----<

/**
 * <h2>Model-specific code for model <em>IdhClock1</em></h2>
 * <p>version demo - Tue Nov 16 15:31:20 CET 2021</p>
 *
 * <dl><dt>Authors: </dt><dd>
 * Jacques Gignoux<br/>
 * Ian D. Davies<br/>
 *          </dd></dl>
 *
 *
 * <dl><dt>Contacts: </dt><dd>
 * jacques.gignoux@upmc.fr<br/>
 * Ian.Davies@anu.edu.au<br/>
 *          </dd></dl>
 *
 *
 * <dl><dt>Reference publication: </dt><dd>
 * https:/en.wikipedia.org/wiki/Competitive_Lotka%E2%80%93Volterra_equations</dd></dl>
 *
 * <h3>Instructions to model developers:</h3>
 * <ol><li>Non 3worlds-generated extra methods should be placed in other files linked
 *  to the present file through imports.</li>
 * <li><strong>Do not</strong> alter the code insertion markers. They are used to avoid
 *  losing your code when managing this file.</li>
 * <li>For convenience, all the static methods of the {@link Math} and
 *  {@link Distance} classes are directly accessible here</li>
 * <li>The particular random number stream attached to each {@link TwFunction} is
 * passed as the <em>random</em> argument.</li>
 * <li>For all <em>Decision-</em> functions, a <em>decider</em> argument is provided to help make decisions out of probabilities.
<strong>decider.decide(double)</strong> returns true with probability equal to the argument.</li>
 * <li>For <em>ChangeCategoryDecision</em> functions, a <em>selector</em> argument is provided to select among different possible
outcomes. <strong>selector.select(double...)</strong> returns an integer between 0 and <em>n</em> (the number of arguments) using the
arguments as weights for probabilities (ie the argument do not need to sum to 1).</li>
 * <li>For <em>ChangeCategoryDecision</em> functions, a <em>recruit</em> argument is provided that must be used to return the proper
category name as a String. <strong>recruit.transition(boolean)</strong> will return the category to recruit to if the argument is
true, triggering the change in category of the focal SystemComponent. <strong>recruit.transition(int)</strong> will return the
category name matching the index using alphabetical order, 0 index meaning no change in category. For example, if the decision may
result in category "young" or "juvenile", 0 will map to no change, 1 to change to juvenile and 2 to change to young.</li> </ol>
*/
public interface IdhClock1 {

  /**
   * <p><strong>InitDist</strong> method of type <em>SetInitialState</em>: sets the initial state of a newly created
                    SystemComponent</p>
   * <p>- applies to categories {<em> *atomic* *component* *permanent* distCat </em>}</p>
   *
   * <p>- called once for every component, at creation time</p>
   *
   * @param freq focal component constants disturbance return interval ± 0.0 ∈]5.0,49.0[
   * @param inten focal component constants disturbance intensity ± 0.0 ∈]0.0,100.0[
   * @param focalCnt new constants for focal component
   * @param random random number generator
  */
  public static void initDist(
    double freq,                        // focal component constants disturbance return interval ± 0.0 ∈]5.0,49.0[
    double inten,                       // focal component constants disturbance intensity ± 0.0 ∈]0.0,100.0[
    InitDist.FocalCnt focalCnt,         // new constants for focal component
    Random random) {                    // random number generator
  // initDist ---- Code insert Begin-->
  focalCnt.freq = 5 + random.nextInt(50);
  focalCnt.inten = random.nextDouble() * 100;
  // initDist ---- Code insert End----<
  }

  /**
      *  <p><strong>InitComm</strong>  method  of  type  <em>SetInitialState</em>:  sets  the  initial  state  of  a  newly  created
SystemComponent</p>
   * <p>- applies to categories {<em> *atomic* *component* *permanent* commCat </em>}</p>
```

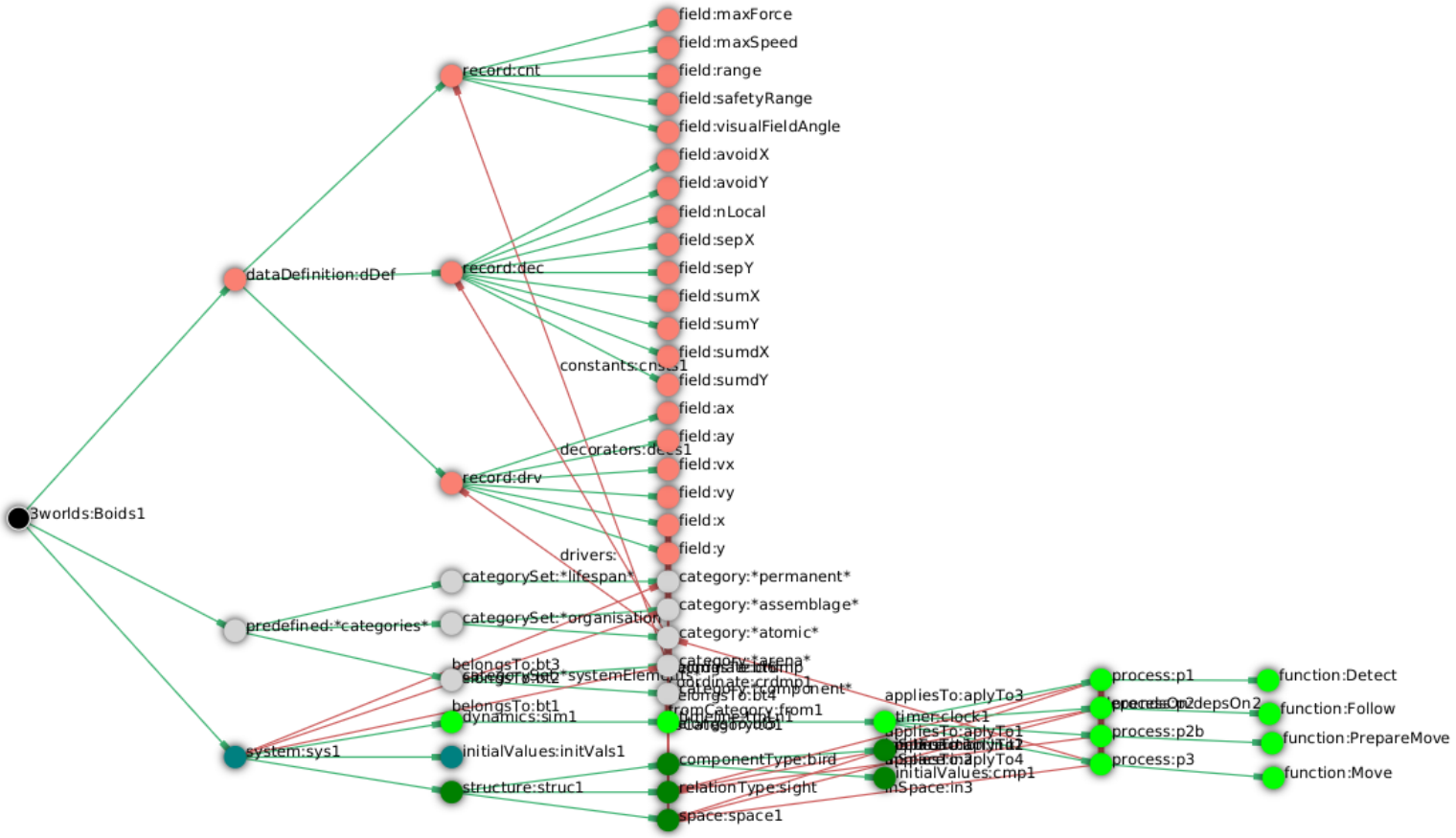
```
   *
   * <p>- called once for every component, at creation time</p>
   *
   * @param tsd focal component drivers time since disturbance (y) ∈ 0.0 ∈]1.0,+∞[
   * @param x focal component drivers population size dim = [40] ∈ 0.0 ∈]0.0,+∞[
   * @param focalDrv next drivers for focal component
   * @param di focal component decorators Shannon's diveristy index ∈ 0.0 ∈]0.0,+∞[
   * @param K focal component constants carrying capacity dim = [40] ∈ 0.0 ∈]5.0,+∞[
   * @param alpha focal component constants interspecific competition coefficient dim = [40,40] ∈ 0.0 ∈]1.0E-4,1.0[
   * @param r focal component constants growth rate dim = [40] ∈ 0.0 ∈]0.0,1.0[
   * @param focalCnt new constants for focal component
   * @param random random number generator
   */
  public static void initComm(
    double tsd,                          // focal component drivers time since disturbance (y) ∈ 0.0 ∈]1.0,+∞[
    DoubleTable x,                       // focal component drivers population size dim = [40] ∈ 0.0 ∈]0.0,+∞[
    InitComm.FocalDrv focalDrv,          // next drivers for focal component
    double di,                           // focal component decorators Shannon's diveristy index ∈ 0.0 ∈]0.0,+∞[
    DoubleTable K,                       // focal component constants carrying capacity dim = [40] ∈ 0.0 ∈]5.0,+∞[
      DoubleTable alpha,                                 // focal component constants interspecific competition coefficient dim =
[40,40] ∈ 0.0 ∈]1.0E-4,1.0[
    DoubleTable r,                       // focal component constants growth rate dim = [40] ∈ 0.0 ∈]0.0,1.0[
    InitComm.FocalCnt focalCnt,          // new constants for focal component
    Random random) {                     // random number generator
  // initComm ---- Code insert Begin-->
  double initFreq = 1.0 / x.size();
  focalDrv.x.fillWith(initFreq);
  for (int i = 0; i < r.size(0); i++) {
    focalCnt.r.setByInt(random.nextDouble(), i);
    focalCnt.K.setByInt(5.0 + initFreq + random.nextDouble(), i);
    for (int j = 0; j < alpha.size(1); j++) {
      if (i == j)
        focalCnt.alpha.setByInt(1.0, i, j);
      else
        focalCnt.alpha.setByInt(max(0.0001, random.nextDouble()), i, j);
    }
  }
  // initComm ---- Code insert End----<
  }

  /**
     * <p><strong>CommGrowth</strong> method of type <em>ChangeState</em>: change the state, i.e. the values of the descriptors
of a system component</p>
   * <p>- applies to categories {<em> commCat </em>}</p>
   *
   * <p>- follows timer <em>clock1</em> of type {@link ClockTimer}, with time unit = 1 t.u</p>
   *
   * <p>- called before function <em>distEffectComm(...)</em>.</p>
   *
   * @param t current time
   * @param dt current time step
   * @param count whole system autoVar count (#) ∈[0..*]
   * @param nAdded whole system autoVar nAdded (#) ∈[0..*]
   * @param nRemoved whole system autoVar nRemoved (#) ∈[0..*]
   * @param tsd focal component drivers time since disturbance (y) ∈ 0.0 ∈]1.0,+∞[
   * @param x focal component drivers population size dim = [40] ∈ 0.0 ∈]0.0,+∞[
   * @param focalDrv next drivers for focal component
   * @param di focal component decorators Shannon's diveristy index ∈ 0.0 ∈]0.0,+∞[
   * @param focalDec new decorators for focal component
   * @param K focal component constants carrying capacity dim = [40] ∈ 0.0 ∈]5.0,+∞[
   * @param alpha focal component constants interspecific competition coefficient dim = [40,40] ∈ 0.0 ∈]1.0E-4,1.0[
   * @param r focal component constants growth rate dim = [40] ∈ 0.0 ∈]0.0,1.0[
   * @param random random number generator
   */
  public static void commGrowth(
    double t,                            // current time
    double dt,                           // current time step
    int count,                           // whole system autoVar count (#) ∈[0..*]
    int nAdded,                          // whole system autoVar nAdded (#) ∈[0..*]
    int nRemoved,                        // whole system autoVar nRemoved (#) ∈[0..*]
    double tsd,                          // focal component drivers time since disturbance (y) ∈ 0.0 ∈]1.0,+∞[
    DoubleTable x,                       // focal component drivers population size dim = [40] ∈ 0.0 ∈]0.0,+∞[
    CommGrowth.FocalDrv focalDrv,        // next drivers for focal component
    double di,                           // focal component decorators Shannon's diveristy index ∈ 0.0 ∈]0.0,+∞[
    CommGrowth.FocalDec focalDec,        // new decorators for focal component
    DoubleTable K,                       // focal component constants carrying capacity dim = [40] ∈ 0.0 ∈]5.0,+∞[
      DoubleTable alpha,                                 // focal component constants interspecific competition coefficient dim =
[40,40] ∈ 0.0 ∈]1.0E-4,1.0[
    DoubleTable r,                       // focal component constants growth rate dim = [40] ∈ 0.0 ∈]0.0,1.0[
    Random random) {                     // random number generator
  // commGrowth ---- Code insert Begin-->
  // increment 'time since disturbance'
  focalDrv.tsd = focalDrv.tsd + 1;
  double[] dxdt = new double[x.size(0)];
  for (int i = 0; i < x.size(0); i++) {
    double sum = 0;
    for (int j = 0; j < alpha.size(1); j++)
      sum += alpha.getByInt(i, j) * focalDrv.x.getByInt(j);
    dxdt[i] = r.getByInt(i) * focalDrv.x.getByInt(i) * (1 - sum / K.getByInt(i));
  }
  for (int i = 0; i < dxdt.length; i++)
    focalDrv.x.setByInt(Math.max(focalDrv.x.getByInt(i) + dxdt[i] * dt, 0.0), i);
  // compute diversity
  double xtot = 0.0;
  for (int i = 0; i < focalDrv.x.size(0); i++)
    xtot += focalDrv.x.getByInt(i);
  focalDec.di = 0.0;
  for (int i = 0; i < focalDrv.x.size(0); i++)
    if (focalDrv.x.getByInt(i) > 0.0)
      focalDec.di -= (focalDrv.x.getByInt(i) / xtot) * log(focalDrv.x.getByInt(i) / xtot);
  // commGrowth ---- Code insert End----<
  }

  /**
   * <p><strong>DistOcc</strong> method of type <em>RelateToDecision</em>: _focal_ establishes a new relation to _other_</p>
   * <p>- applies to relation <em>distEffectComm: { distCat } → { commCat }</em></p>
   *
```

```
    * <p>- follows timer <em>clock1</em> of type {@link ClockTimer}, with time unit = 1 t.u</p>
    *
    * @param t current time
    * @param dt current time step
    * @param count whole system autoVar count (#) ϵ[0..*]
    * @param nAdded whole system autoVar nAdded (#) ϵ[0..*]
    * @param nRemoved whole system autoVar nRemoved (#) ϵ[0..*]
    * @param freq focal component constants disturbance return interval ± 0.0 ϵ]5.0,49.0[
    * @param inten focal component constants disturbance intensity ± 0.0 ϵ]0.0,100.0[
    * @param other_tsd other component drivers time since disturbance (y) ± 0.0 ϵ]1.0,+∞[
    * @param other_x other component drivers population size dim = [40] ± 0.0 ϵ]0.0,+∞[
    * @param other_di other component decorators Shannon's diveristy index ± 0.0 ϵ]0.0,+∞[
    * @param other_K other component constants carrying capacity dim = [40] ± 0.0 ϵ]5.0,+∞[
    * @param other_alpha other component constants interspecific competition coefficient dim = [40,40] ± 0.0 ϵ]1.0E-4,1.0[
    * @param other_r other component constants growth rate dim = [40] ± 0.0 ϵ]0.0,1.0[
    * @param random random number generator
    * @param decider decision function
    * @return true if a new relation is to be set between <em>focal</em> and  <em>other</em>
    */
   public static boolean distOcc(
     double t,                              // current time
     double dt,                             // current time step
     int count,                             // whole system autoVar count (#) ϵ[0..*]
     int nAdded,                            // whole system autoVar nAdded (#) ϵ[0..*]
     int nRemoved,                          // whole system autoVar nRemoved (#) ϵ[0..*]
     double freq,                           // focal component constants disturbance return interval ± 0.0 ϵ]5.0,49.0[
     double inten,                          // focal component constants disturbance intensity ± 0.0 ϵ]0.0,100.0[
     double other_tsd,                      // other component drivers time since disturbance (y) ± 0.0 ϵ]1.0,+∞[
     DoubleTable other_x,                   // other component drivers population size dim = [40] ± 0.0 ϵ]0.0,+∞[
     double other_di,                       // other component decorators Shannon's diveristy index ± 0.0 ϵ]0.0,+∞[
     DoubleTable other_K,                   // other component constants carrying capacity dim = [40] ± 0.0 ϵ]5.0,+∞[
        DoubleTable  other_alpha,                       //  other  component  constants  interspecific  competition  coefficient  dim  =
[40,40] ± 0.0 ϵ]1.0E-4,1.0[
     DoubleTable other_r,                   // other component constants growth rate dim = [40] ± 0.0 ϵ]0.0,1.0[
     Random random,                         // random number generator
     DecisionFunction decider) {            // decision function
   // distOcc ---- Code insert Begin-->
     if (random.nextDouble() < 1.0 / freq)
       return true;
     else
       return false;
   // distOcc ---- Code insert End----<
   }

   /**
    * <p><strong>DistEffectComm</strong> method of type <em>ChangeOtherState</em>: _focal_ changes the state of _other_</p>
    * <p>- applies to relation <em>distEffectComm: { distCat } → { commCat }</em></p>
    *
    * <p>- follows timer <em>clock1</em> of type {@link ClockTimer}, with time unit = 1 t.u</p>
    *
    * <p>- called after function <em>commGrowth(...)</em>.</p>
    *
    * @param t current time
    * @param dt current time step
    * @param count whole system autoVar count (#) ϵ[0..*]
    * @param nAdded whole system autoVar nAdded (#) ϵ[0..*]
    * @param nRemoved whole system autoVar nRemoved (#) ϵ[0..*]
    * @param freq focal component constants disturbance return interval ± 0.0 ϵ]5.0,49.0[
    * @param inten focal component constants disturbance intensity ± 0.0 ϵ]0.0,100.0[
    * @param other_tsd other component drivers time since disturbance (y) ± 0.0 ϵ]1.0,+∞[
    * @param other_x other component drivers population size dim = [40] ± 0.0 ϵ]0.0,+∞[
    * @param otherDrv next drivers for other component
    * @param other_di other component decorators Shannon's diveristy index ± 0.0 ϵ]0.0,+∞[
    * @param otherDec new decorators for other component
    * @param other_K other component constants carrying capacity dim = [40] ± 0.0 ϵ]5.0,+∞[
    * @param other_alpha other component constants interspecific competition coefficient dim = [40,40] ± 0.0 ϵ]1.0E-4,1.0[
    * @param other_r other component constants growth rate dim = [40] ± 0.0 ϵ]0.0,1.0[
    * @param random random number generator
    */
   public static void distEffectComm(
     double t,                              // current time
     double dt,                             // current time step
     int count,                             // whole system autoVar count (#) ϵ[0..*]
     int nAdded,                            // whole system autoVar nAdded (#) ϵ[0..*]
     int nRemoved,                          // whole system autoVar nRemoved (#) ϵ[0..*]
     double freq,                           // focal component constants disturbance return interval ± 0.0 ϵ]5.0,49.0[
     double inten,                          // focal component constants disturbance intensity ± 0.0 ϵ]0.0,100.0[
     double other_tsd,                      // other component drivers time since disturbance (y) ± 0.0 ϵ]1.0,+∞[
     DoubleTable other_x,                   // other component drivers population size dim = [40] ± 0.0 ϵ]0.0,+∞[
     DistEffectComm.OtherDrv otherDrv,      // next drivers for other component
     double other_di,                       // other component decorators Shannon's diveristy index ± 0.0 ϵ]0.0,+∞[
     DistEffectComm.OtherDec otherDec,      // new decorators for other component
     DoubleTable other_K,                   // other component constants carrying capacity dim = [40] ± 0.0 ϵ]5.0,+∞[
        DoubleTable  other_alpha,                       //  other  component  constants  interspecific  competition  coefficient  dim  =
[40,40] ± 0.0 ϵ]1.0E-4,1.0[
     DoubleTable other_r,                   // other component constants growth rate dim = [40] ± 0.0 ϵ]0.0,1.0[
     Random random) {                       // random number generator
   // distEffectComm ---- Code insert Begin-->
   for (int i = 0; i < otherDrv.x.size(); i++)
     if (otherDrv.x.getByInt(i) > other_K.getByInt(i) * inten / 100000.0) {
       otherDrv.x.setByInt(otherDrv.x.getByInt(i) * other_K.getByInt(i) * inten / 100000.0, i);
       // reset time since disturbance
       otherDrv.tsd = 0;
     }
   // compute diversity
   double xtot = 0.0;
   for (int i = 0; i < other_x.size(0); i++)
     xtot += other_x.getByInt(i);
   otherDec.di = 0.0;
   for (int i = 0; i < other_x.size(0); i++)
     if (otherDrv.x.getByInt(i) > 0.0)
       otherDec.di -= (otherDrv.x.getByInt(i) / xtot) * log(otherDrv.x.getByInt(i) / xtot);
   // distEffectComm ---- Code insert End----<
   }

}
```

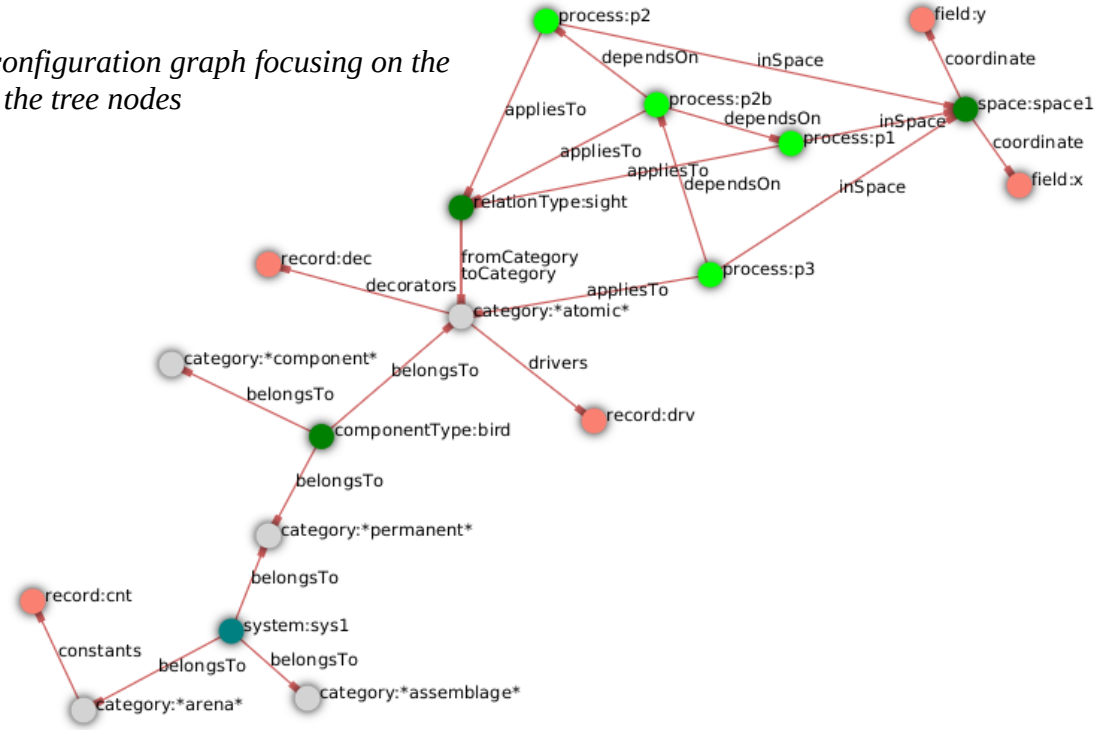
Appendix 5

An individual-based model : Boids

This model is part of the 3Worlds *tutorial* library (**tw-models**). cf. text for general description.

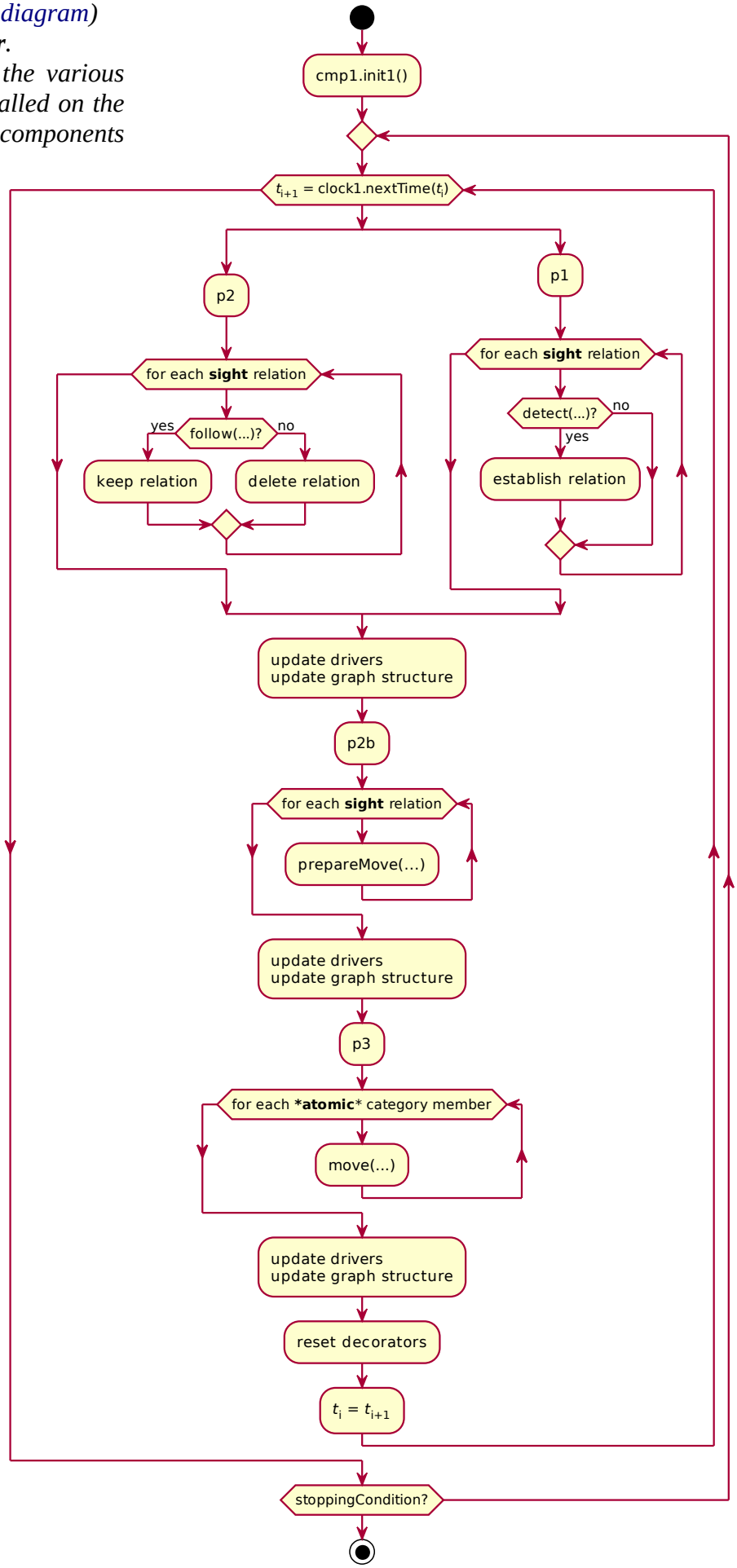


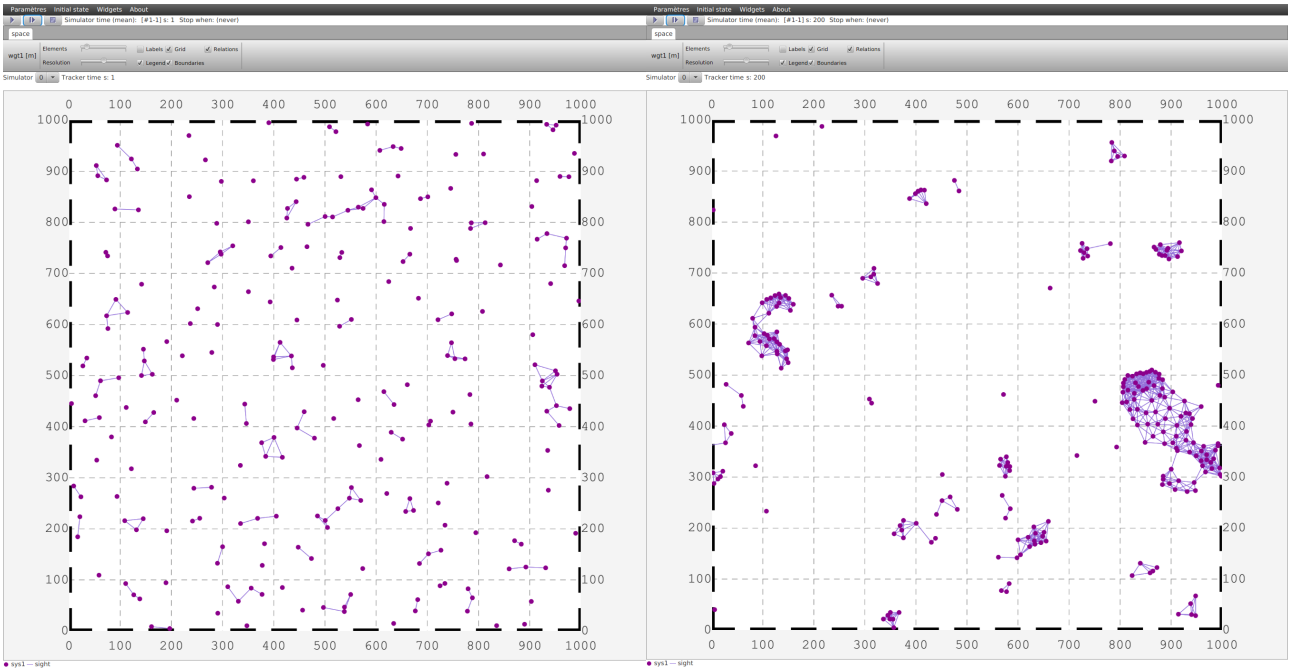
Screen copy of the configuration graph showing the tree structure (green) and cross-links (red)



Screen copy of the configuration graph focusing on the cross-links between the tree nodes

*Flow chart (as UML activity diagram) as exported by **ModelRunner**.*
This shows in which order the various user-defined processes are called on the various user-defined system components representing the ecosystem.





Two screen copies of ModelRunner, at time t=0 (left) and t=200 (right). They represent the map of the space of the model, with the location of every individual bird *as a purple dot, and every individual* sight *relation as a line between two dots. From an initial random pattern of birds, we quickly reach a strongly aggregated pattern with coherent movement of flocks.*

Configuration graph metrics

Metric	Value
1 #Nodes	32
2 #Edges	15
3 #Properties	142
4 configuration size (1+2+3)	189
5 #Drivers	6
6 #Constants	5
7 #Decorators	9
8 #ComponentTypes	1
9 #RelationTypes	1
10 #GroupTypes	0
11 #Lines of code	84
12 #Complexity	16,301

The java code specific to the model. Generated parts are omitted, user-defined edited code is in red:

move(...)

```
// code reproduced from: https://betterprogramming.pub/boids-simulating-birds-flock-behavior-in-python-9fff99375118
double spdX = vx;
double spdY = vy;
double accX = ax;
double accY = ay;
// alignment
double alignmentX = (random.nextDouble()-0.5)*maxForce/10;
double alignmentY = (random.nextDouble()-0.5)*maxForce/10;
if (nLocal>0) {
  double norm = Math.sqrt(sumOfSquares(sumdX,sumdY));
  double avgvx = sumdX/nLocal /norm *maxSpeed;
  double avgvy = sumdY/nLocal /norm *maxSpeed;
  alignmentX = avgvx-vx;
  alignmentY = avgvy-vy;
}
```

```
double cohesionX = 0;
double cohesionY = 0;
// cohesion
if (nLocal>0) {
  // distance to flock barycentre
  double barX = sumX/nLocal-x;
  double barY = sumY/nLocal-y;
  double norm = Math.sqrt(sumOfSquares(barX,barY));
  if (norm>0) { // if non zero net move
    barX = barX/norm*maxSpeed;
    barY = barY/norm*maxSpeed;
  }
  cohesionX = barX-vx;
  cohesionY = barY-vy;
  norm = Math.sqrt(sumOfSquares(cohesionX,cohesionY));
  if (norm>maxForce) {
    cohesionX = cohesionX/norm*maxForce;
    cohesionY = cohesionY/norm*maxForce;
  }
}
// separation
double separationX = 0;
double separationY = 0;
if (nLocal>0) {
  double fleeX = sepX/nLocal;
  double fleeY = sepY/nLocal;
  double norm = Math.sqrt(sumOfSquares(fleeX,fleeY));
  if (norm>0 ) {
    fleeX = fleeX/norm*maxSpeed;
    fleeY = fleeY/norm*maxSpeed;
  }
  separationX = fleeX-vx;
  separationY = fleeY-vy;
  norm = Math.sqrt(sumOfSquares(separationX,separationY));
  if (norm>maxForce) {
    separationX = separationX/norm*maxForce;
    separationY = separationY/norm*maxForce;
  }
}
// update to next step
accX += alignmentX + cohesionX + separationX;
accY += alignmentY + cohesionY + separationY;
focalDrv.x = x+spdX;
focalDrv.y = y+spdY;
// apply speed limit - not needed if initial speed is < max speed.
double v = Math.sqrt(sumOfSquares(vx+accX,vy+accY));
if (v>maxSpeed) {
  focalDrv.vx = (spdX+accX)/v*maxSpeed;
  focalDrv.vy = (spdY+accY)/v*maxSpeed;
}
else {
  focalDrv.vx = spdX+accX;
  focalDrv.vy = spdY+accY;
}
focalDrv.ax = 0;
focalDrv.ay = 0;
[Lines: 64]
```

prepareMove(...)

```
focalDec.nLocal++;
// alignment
focalDec.sumdX += other_vx;
focalDec.sumdY += other_vy;
// cohesion
focalDec.sumX += other_x;
focalDec.sumY += other_y;
// separation
double dist = euclidianDistance(x,y,other_x,other_y);
focalDec.sepX += (x-other_x)/dist;
focalDec.sepY += (y-other_y)/dist;
[Lines: 8]
```

detect(...)

```
if (squaredEuclidianDistance(x,y,other_x,other_y)<sqr(range))
  return true;
return false;
[Lines: 3]
```

init1(...)

```
focalDrv.x = random.nextDouble()*1000.0;
focalDrv.y = random.nextDouble()*1000.0;
focalDrv.vx = (random.nextDouble()-0.5)*10; // max speed
focalDrv.vy = (random.nextDouble()-0.5)*10; // max speed
focalDrv.ax = (random.nextDouble()-0.5)*1.0; // maxForce
focalDrv.ay = (random.nextDouble()-0.5)*1.0; // maxForce
[Lines: 6]
```

follow(...)

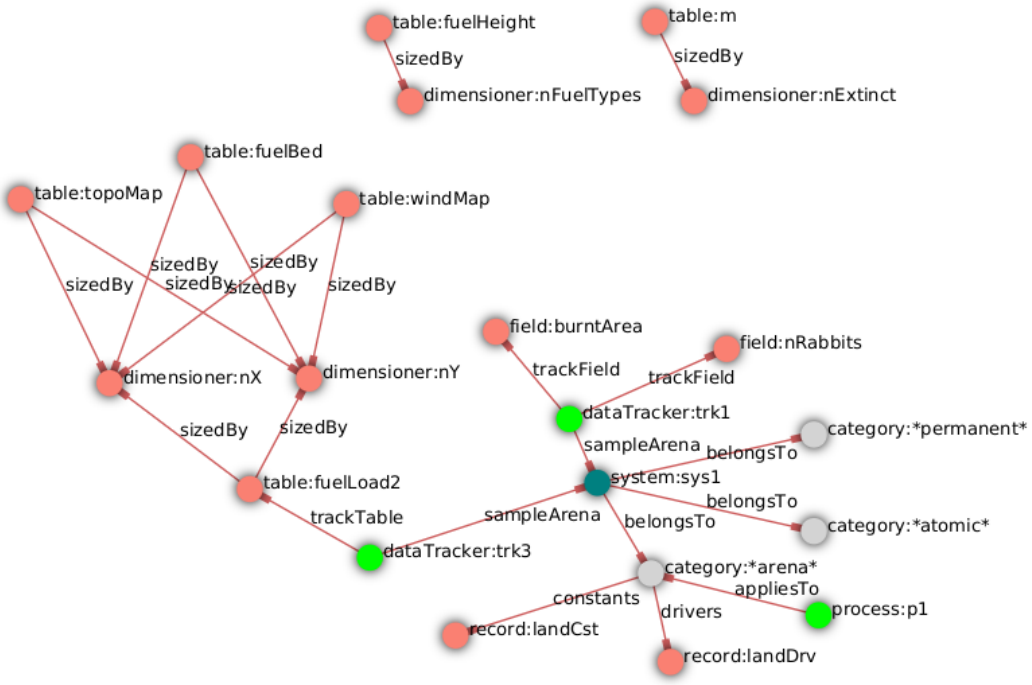
```
if (squaredEuclidianDistance(x,y,other_x,other_y)<sqr(range))
  return true;
return false;
[Lines: 3]
```

Appendix 6

A cellular automaton: the '*Rabbit Rules*' fire spread model

This model is part of the 3Worlds *model* library (**tw-models**). cf. text for general description.

Screen copy of the configuration graph showing the tree structure

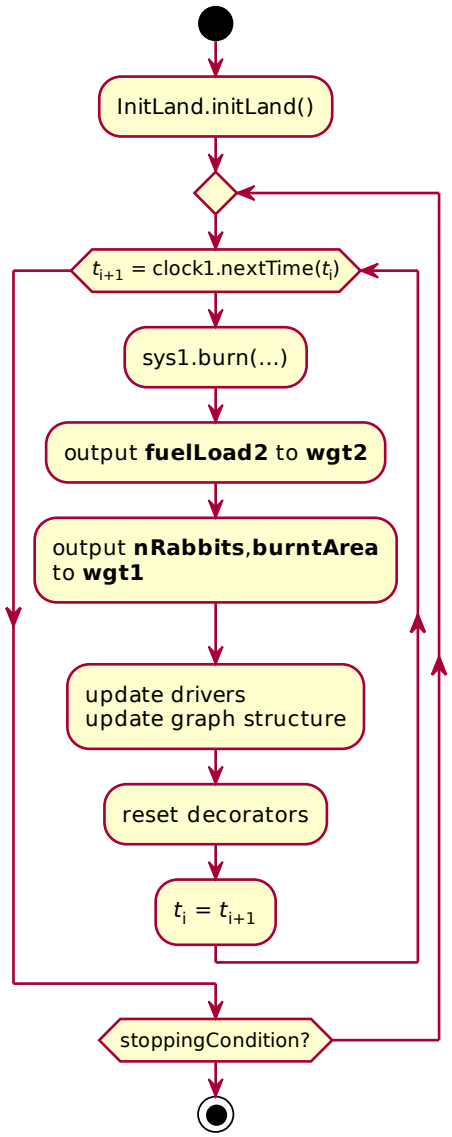


Screen copy of the configuration graph showing the cross-links between tree nodes

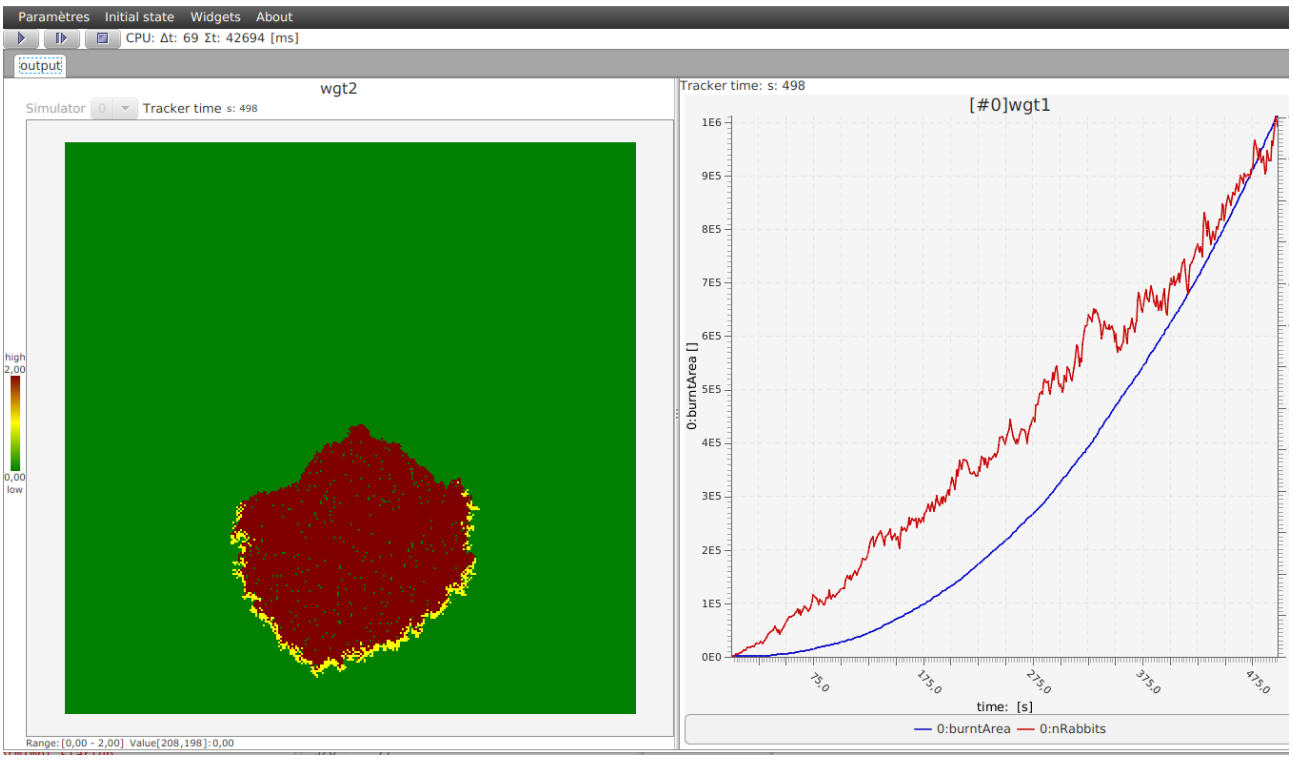
Configuration graph metrics

Metric	Value
1 #Nodes	37
2 #Edges	12
3 #Properties	145
4 configuration size (1+2+3)	194
5 #Drivers	270,011
6 #Constants	90,019
7 #Decorators	0
8 #ComponentTypes	0
9 #RelationTypes	0
10 #GroupTypes	0
11 #Lines of code	148
12 #Complexity	17,638

*Flow chart (as UML activity diagram) as exported by **ModelRunner**. This shows in which order the various user-defined processes are called on the various user-defined system components representing the ecosystem.*



Screen copy of ModelRunner. Left panel, the map of the site, showing the burnt area in brown, the fire front in yellow and the unburnt vegetation in green. Right panel shows the burnt area (blue) and the number of fire pixels (red) as a function of time



The java code specific to the model. Generated parts are omitted, user-defined edited code is in red:

burn(...)

```
int nCellX = topoMap.size(0);
int nCellY = topoMap.size(1);
Set<Fuel> hoppingRabbits = new HashSet<>();
// first loop on burning cells (with rabbits)
// use current drivers as a workspace since it's a table
for (int i=0; i<nCellX; i++)
  for (int j=0; j<nCellY; j++) {
    // fuel in current cell
    Fuel f = fuelBed.getByInt(i,j);
    // wind components in current cell
    Wind w = windMap.getByInt(i,j);
    double u = w.u();
    double v = w.v();
    double S2 = sumOfSquares(u,v);
    double fS = 1.0;
    if (S2>4)
      fS = 0.01068*S2 -0.21*sqrt(S2) + 1.53; //Acthemeier 2013 eq. 7
    // slope components in current cell
    double Cw = Cw0*fS;
    Slope s = topoMap.getByInt(i,j);
    double sX = s.sX();
    double sY = s.sY();
    // if burning:
    if (f.residentRabbitPresent()) {
      // rabbit reproduction
      if (round(f.residentRabbitReproductionTime()) == round(t)) {
        // attempt to create nNewRabbits
        for (int nR = 0; nR<nNewRabbits; nR++) {
          double fh = fuelHeight.getByInt(f.fuelType());
          double zr = 2*(0.1+random.nextDouble())*fh;
          double hoppingTime = t+dt+2*sqrt(2*zr/g); // TODO: scaling of time step and time units
          // hopping distance: wind and slope effect
          double dX = ( (Cw*u*abs(u)+10*Cf*sX*abs(sX))*dt + Ch*zr*(0.5-random.nextDouble()) )*cellSize*3;
          double dY = ( (Cw*v*abs(v)+10*Cf*sY*abs(sY))*dt + Ch*zr*(0.5-random.nextDouble()) )*cellSize*3;
          // where does the rabbit jump ?
          double x = (i+0.5)*cellSize+dX;
          double y = (j+0.5)*cellSize+dY;
          // forget rabbits jumping outside map
          if ((x>0.0)&&(x<xMax)&&(y>0.0)&&(y<yMax)) {
            int cx = (int) floor(x/xMax * nCellX);
            int cy = (int) floor(y/yMax * nCellY);
            Fuel landingCell = fuelBed.getByInt(cx,cy);
            // if landing cell has a rabbit, then no point landing there so forget it.
            if (!landingCell.residentRabbitPresent()) {
              double reproTime = hoppingTime+dt+random.nextDouble()*cellSize; // TODO: scaling of time step and time units
              double proba = random.nextDouble();
              double dieOutTime = m.getByInt(0)*60;
              if (proba<0.5) {
                dieOutTime += m.getByInt(1)*60;
                if (proba<0.2) {
                  dieOutTime += m.getByInt(2)*60;
                  if (proba<0.05) {
                    dieOutTime += m.getByInt(3)*60;
                  }
                }
              }
              dieOutTime += reproTime+dt; // a minimum of 1 time step is required
              // if a hopping rabbit is going to land in my landing cell, then check landing time and
              // keep the earliest one, and forget about new rabbit (it replaced an old one
              if (landingCell.hoppingRabbitExpected()) {
                if (round(hoppingTime)<round(landingCell.hoppingRabbitLandingTime())) {
                  // replace former flying rabbit with a new one with precomputed time values
                  landingCell.hoppingRabbitLandingTime(hoppingTime);
                  landingCell.residentRabbitReproductionTime(reproTime);
                  landingCell.residentRabbitDieOutTime(dieOutTime);
                  hoppingRabbits.add(landingCell);
                }
              }
              // if no hopping rabbit nor resident rabbit present, check there is something to burn and burn it!
              else if (landingCell.fuelLoad()==0) {
                // place a new flying rabbit in this cell with precomputed time values
                landingCell.hoppingRabbitExpected(true);
                landingCell.hoppingRabbitLandingTime(hoppingTime);
                landingCell.residentRabbitReproductionTime(reproTime);
                landingCell.residentRabbitDieOutTime(dieOutTime);
                hoppingRabbits.add(landingCell);
              }
            }
          }
        }
      }
      // rabbit death after burning
      else if (round(f.residentRabbitDieOutTime()) == round(t)) {
        f.residentRabbitPresent(false);
        // do not change hopping rabbit time, just in case one is already present. (normally no because it should have died)
        f.fuelLoad((byte) 2); // set fuel to burnt
      }
    }
    if (f.hoppingRabbitExpected())
      hoppingRabbits.add(f);
  } // 1st loop on all cells
// all resident rabbits have been cleaned up or have reproduced, so now handle hopping rabbits that may land
// 2nd loop on hopping rabbits only
for (Fuel f:hoppingRabbits) {
  if (round(f.hoppingRabbitLandingTime()) == round(t)) {
    if (!f.residentRabbitPresent()) { // normally this should always be true, but who knows what can happen...
      f.residentRabbitPresent(true);
      f.hoppingRabbitExpected(false);
      // times for resident rabbit were already set at hopping rabbit creation, no need to change them
      f.fuelLoad((byte) 1); // set fuel to burning
    }
```

```
  }
}
// copy current to next drivers
// 3rd loop
int ba = 0;
int nr = 0;
for (int i=0; i<nCellX; i++) {
  for (int j=0; j<nCellY; j++) {
    Fuel current = fuelBed.getByInt(i,j);
    Fuel next = focalDrv.fuelBed.getByInt(i,j);
    next.fuelLoad(current.fuelLoad());
    next.fuelType(current.fuelType());
    next.residentRabbitPresent(current.residentRabbitPresent());
    next.hoppingRabbitExpected(current.hoppingRabbitExpected());
    next.residentRabbitReproductionTime(current.residentRabbitReproductionTime());
    next.residentRabbitDieOutTime(current.residentRabbitDieOutTime());
    next.hoppingRabbitLandingTime(current.hoppingRabbitLandingTime());
    if (current.fuelLoad()==2)
      ba++;
    if (current.residentRabbitPresent())
      nr++;
    focalDrv.fuelLoad2.setByInt(current.fuelLoad(),i,j);
    Wind ctwind = windMap.getByInt(i,j);
    Wind nxtwind = focalDrv.windMap.getByInt(i,j);
    nxtwind.u(ctwind.u());
    nxtwind.v(ctwind.v());
  }
}
focalDrv.burntArea = ba*cellSize*cellSize;
focalDrv.nRabbits = nr;
[Lines: 107]
```

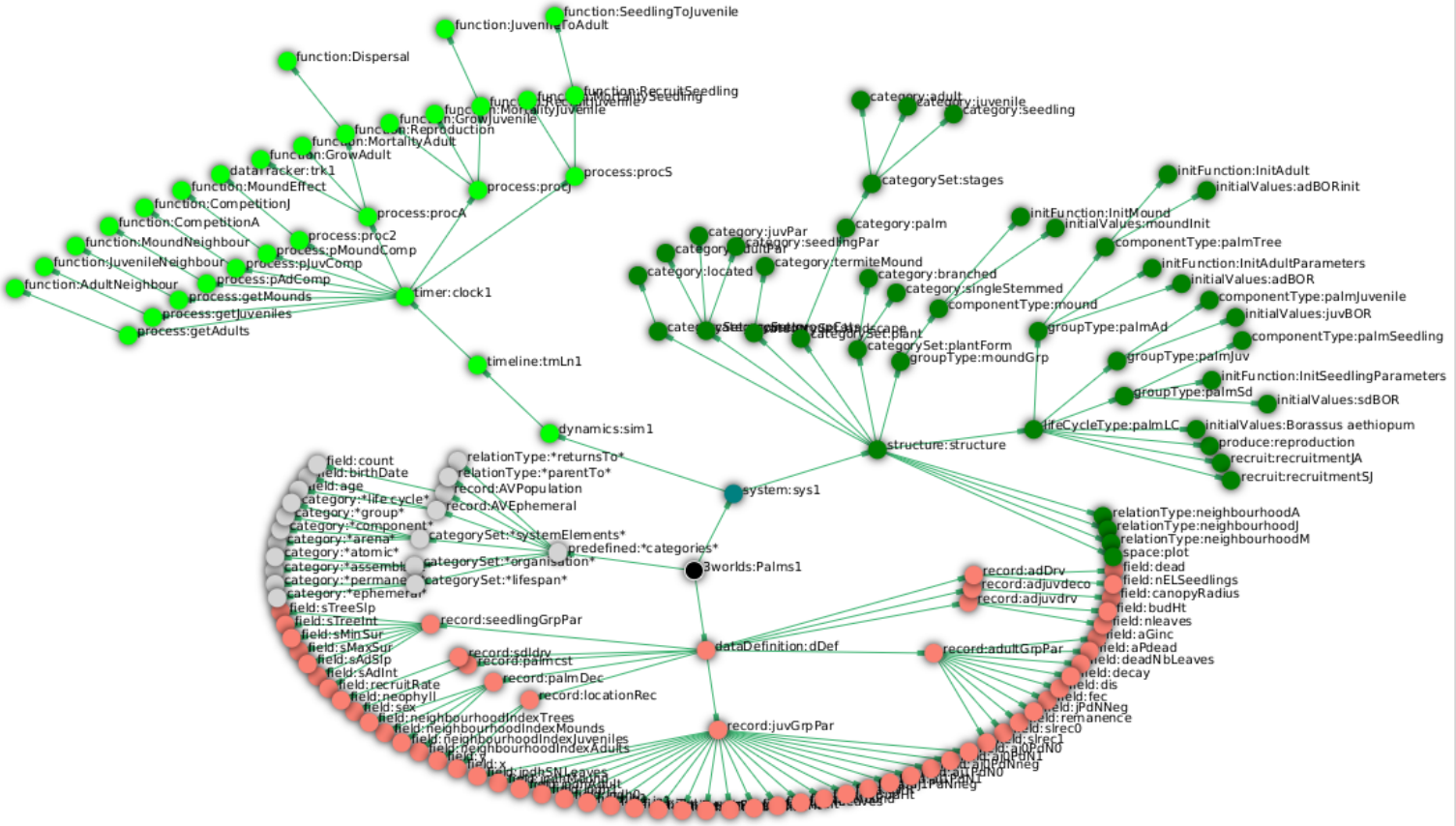
initLand(...)

```
int nCellX = topoMap.size(0);
int nCellY = topoMap.size(1);
//site dimensions
focalCnt.xMax = nCellX*cellSize;
focalCnt.yMax = nCellY*cellSize;
// fuel height dep. on type
focalCnt.fuelHeight.setByInt(1.0,0);
focalCnt.fuelHeight.setByInt(1.25,1);
focalCnt.fuelHeight.setByInt(2.0,2);
focalCnt.fuelHeight.setByInt(5.0,3);
focalCnt.fuelHeight.setByInt(20.0,4);
// setting the topography, wind map, and initial vegetation map
// use data files for more elaborate things
for (int i=0; i<nCellX; i++) {
  for (int j=0; j<nCellY; j++) {
    Slope s = topoMap.getByInt(i,j);
    s.sX(0.0);
    s.sY(0.0);
    Wind w = windMap.getByInt(i,j);
    if (j<nCellY/2) {
      w.u(0.0);
      w.v(-1);
    }
    else {
      w.u(0.0);
      w.v(-1);
    }
    Fuel f = focalDrv.fuelBed.getByInt(i,j);
    // fuel type is grass - means height is 1.25
    f.fuelType(1);
    // fuel load is 0 (unburnt)
    f.fuelLoad((byte) 0);
    f.residentRabbitPresent(false); // except the central one !
    f.hoppingRabbitExpected(false);
    f.residentRabbitReproductionTime(0.0);
    f.residentRabbitDieOutTime(0.0);
    f.hoppingRabbitLandingTime(0.0);
  }
}
// ignition: the initial rabbit !
Fuel f = focalDrv.fuelBed.getByInt(nCellX/2,nCellY/2);
f.residentRabbitPresent(true);
f.residentRabbitReproductionTime(1.0);
f.residentRabbitDieOutTime(2.0);
focalDrv.nRabbits = 1;
[Lines: 38]
```

Appendix 7

The Lamto palm tree dynamics model (LPTD)

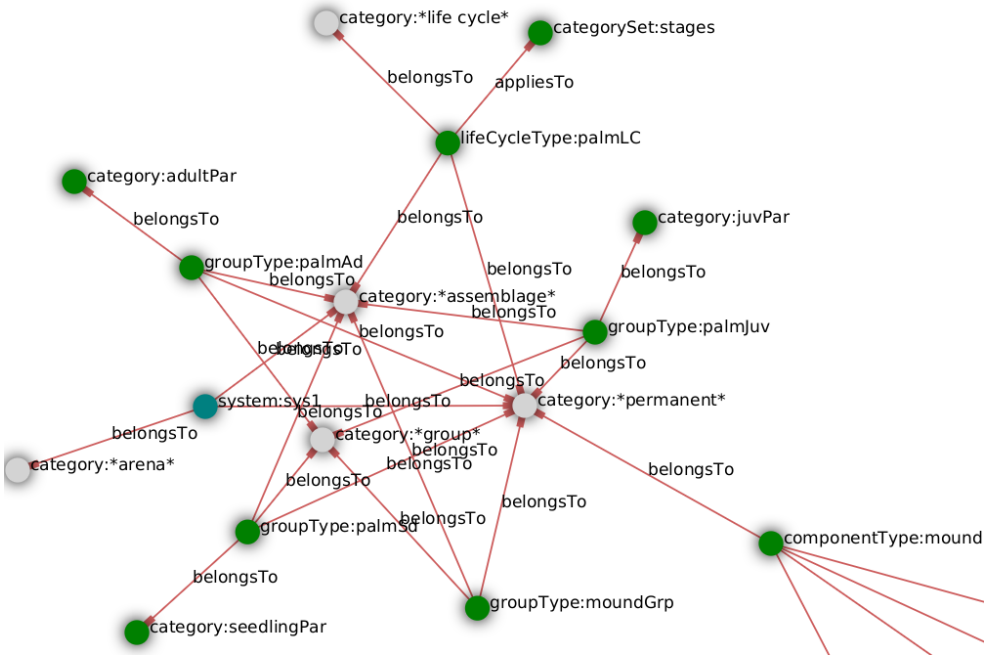
This model is part of the 3Worlds *model* library (**tw-models**). cf. text for general description.



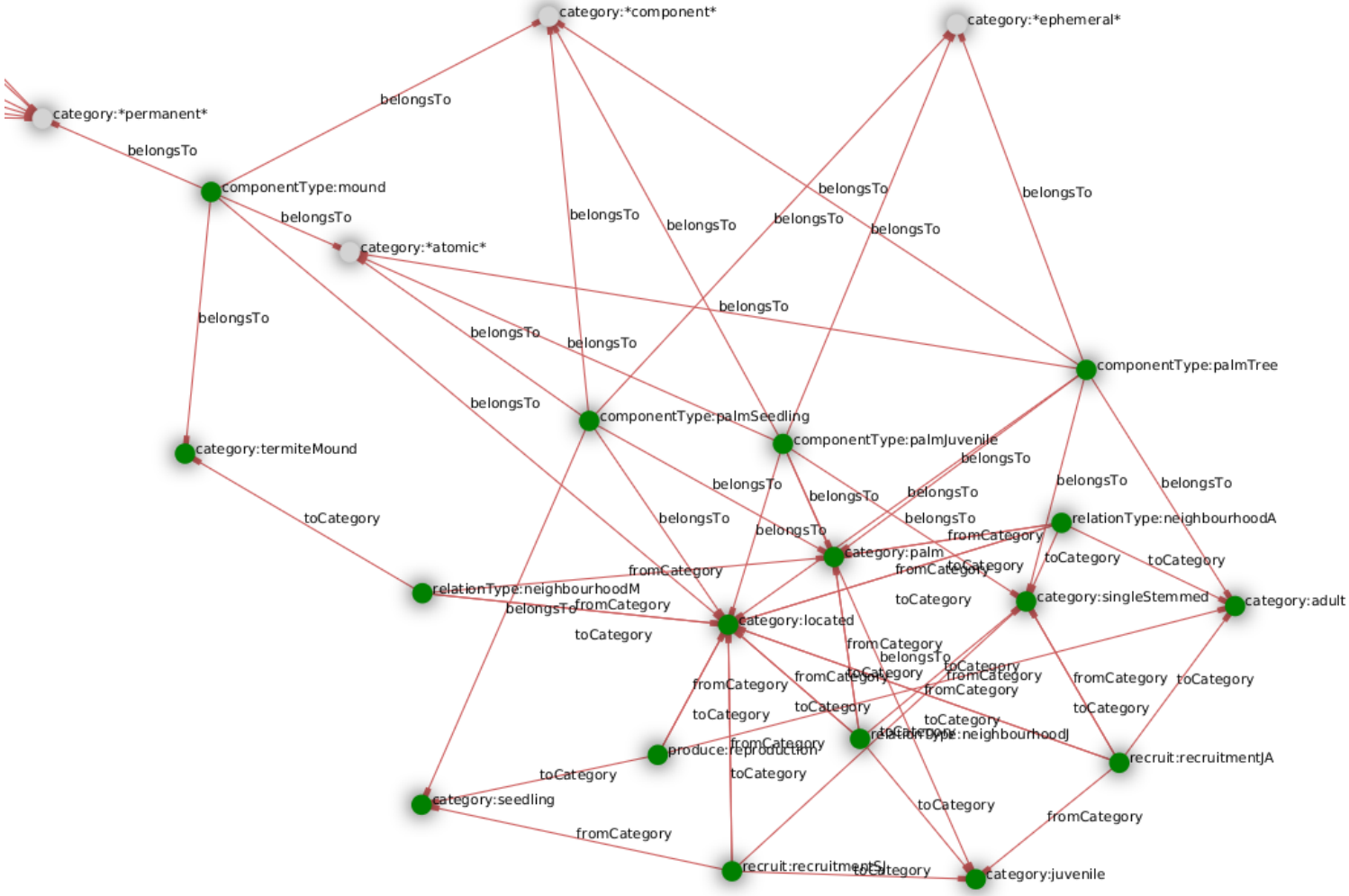
Screen copy of the configuration graph showing the tree structure

Configuration graph metrics

Metric	Value
1 #Nodes	125
2 #Edges	116
3 #Properties	415
4 configuration size (1+2+3)	656
5 #Drivers	5
6 #Constants	43
7 #Decorators	5
8 #ComponentTypes	4
9 #RelationTypes	3
10 #GroupTypes	0
11 #Lines of code	164
12 #Complexity	60,109

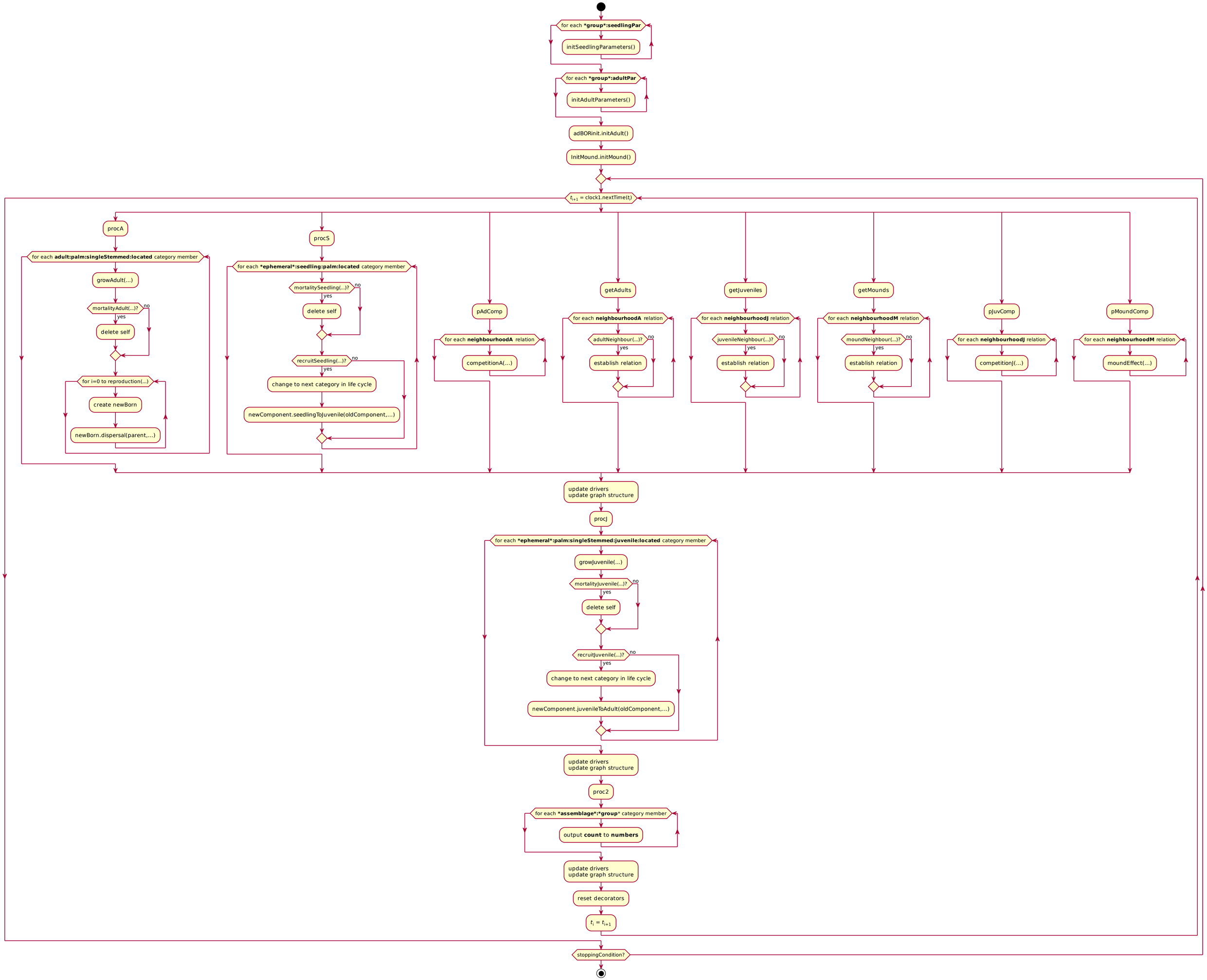


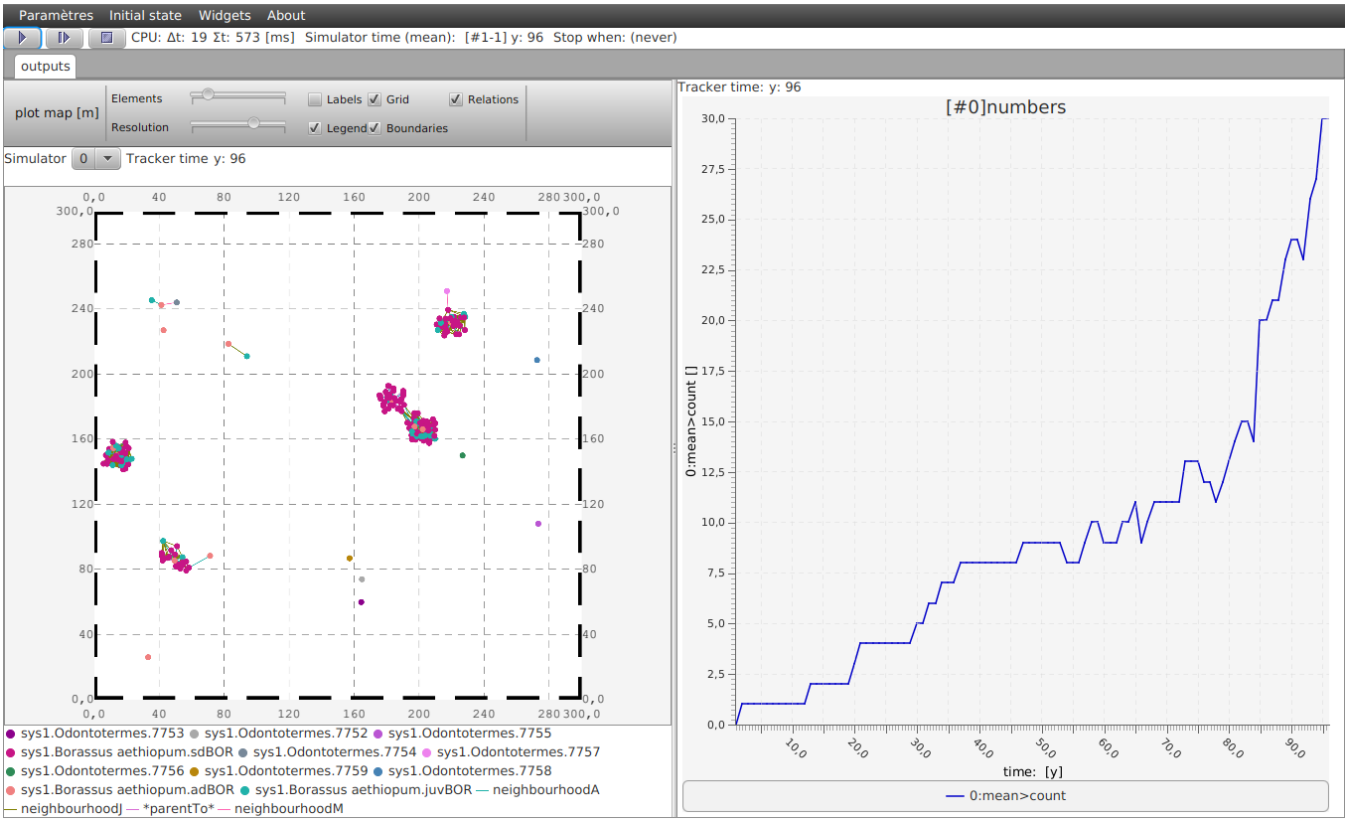
Two screen copies of the configuration graph showing the cross-links between the tree nodes



Model constants

Constants	name	Description	Dimensions	Type
palmcst [palm]	***sex***	female?	scalar	Boolean
seedlingGrpPar [seedlingPar]	***sAdSlp***	Seedling-adult comp slp.	scalar	Double
	recruitRate	Recruitment proba.	scalar	Double
	sAdInt	Seedling-adult comp int.	scalar	Double
	sTreeInt	Seedling-tree comp int.	scalar	Double
	sMinSur	Min Survival Rate	scalar	Double
	sMaxSur	Max Survival Rate	scalar	Double
	sTreeSlp	Seedling-tree comp slp.	scalar	Double
juvGrpPar [juvPar]	***aj0PdNneg***	P(dN=-1|H=0): intercept	scalar	Double
	j0BudHt	P(Rec): Height slope	scalar	Double
	j0NLeaves	P(dN|H=0): nleaves slope	scalar	Double
	j1Juvenile	P(dN|H>0): Juvenile slope	scalar	Double
	j1Adult	P(dN|H>0): Adults slope	scalar	Double
	ajPR	P(Rec): int.	scalar	Double
	aj1PdN0	P(dN=0|H>0): intercept	scalar	Double
	aj1PdN1	P(dN=1|H>0): intercept	scalar	Double
	jpdh0		scalar	Double
	j0Mound	P(dN|H=0): mounds slope	scalar	Double
	jpdhMound		scalar	Double
	jpdh1		scalar	Double
	jpdhSNLeaves		scalar	Double
	jDeathRate	death rate	scalar	Double
	jpdhAdult		scalar	Double
	aj0PdN0	P(dN=0|H=0): intercept	scalar	Double
	aj0PdN1	P(dN=1|H=0): intercept	scalar	Double
	jTol	growth deficit tolerance	scalar	Double
	jb1		scalar	Double
	j1Tree	P(dN|H>0): Trees slope	scalar	Double
	aj1PdNneg	P(dN=-1|H>0): intercept	scalar	Double
	ja1		scalar	Double
	j1BudHt	P(dN|H>0): Height slope	scalar	Double
locationRec [located]	***x***	x spatial coordinate	scalar	Double
	y	y spatial coordinate	scalar	Double
adultGrpPar [adultPar]	***aGinc***	adult BudHeight growth coefficient	scalar	Double
	fec	fecundity * # leaves	scalar	Double
	remanence	seedling remanenc	scalar	Double
	decay		scalar	Double
	slrec0	rect els to sls alive	scalar	Double
	slrec1	rect els to sls dead	scalar	Double
	deadNbLeaves	Mortality : nbf slope	scalar	Double
	jPdNNeg	Adult P(dN=-1)	scalar	Double
	aPdead	Mortality : int	scalar	Double
	dis	dispersal parameter	scalar	Double


for each *group*:seedlingPar
initSeedlingParameters()
for each *group*:adultPar
initAdultParameters()
adBORinit.initAdult()
InitMound.initMound()
t_{i+1} = clock1.nextTime(t_i)
procA
for each adult:palm:singleStemmed:located category member
growAdult(...)
mortalityAdult(...)?
no
yes
delete self
for i=0 to reproduction(...)
create newBorn
newBorn.dispersal(parent,...)
procS
for each *ephemeral*:seedling:palm:located category member
mortalitySeedling(...)?
no
yes
delete self
recruitSeedling(...)?
no
yes
change to next category in life cycle
newComponent.seedlingToJuvenile(oldComponent,...)
pAdComp
for each neighbourhoodA relation
competitionA(...)
getAdults
for each neighbourhoodA relation
adultNeighbour(...)?
no
yes
establish relation
getJuveniles
for each neighbourhoodJ relation
juvenileNeighbour(...)?
no
yes
establish relation
getMounds
for each neighbourhoodM relation
moundNeighbour(...)?
no
yes
establish relation
pJuvComp
for each neighbourhoodJ relation
competitionJ(...)
pMoundComp
for each neighbourhoodM relation
moundEffect(...)
update drivers
update graph structure
procJ
for each *ephemeral*:palm:singleStemmed:juvenile:located category member
growJuvenile(...)
mortalityJuvenile(...)?
no
yes
delete self
recruitJuvenile(...)?
no
yes
change to next category in life cycle
newComponent.juvenileToAdult(oldComponent,...)
update drivers
update graph structure
proc2
for each *assemblage*:*group* category member
output count to numbers
update drivers
update graph structure
reset decorators
$t_i = t_{i+1}$
stoppingCondition?



Screen copy of ModelRunner. Left panel, the map of the site, where every dot represents a palm tree or a mound. Colors differ for every palm tre stage, and for every termite mound. Right panel shows the number of juveniles as a function of time

The java code specific to the model. Generated parts are omitted, user-defined edited code is in red:

moundEffect(...)

```
// relativedistance weight, ie = 1 if at focal location, =0 at searchRadius distance
double weight = 1-euclidianDistance(x,y,other_x,other_y)/15.0;//searchRadius;
focalDec.neighbourhoodIndexMounds += weight;
[Lines: 2]
```

mortalityAdult(...)

```
if (dead>=group_remanence)
  return true;
else
  return false;
[Lines: 4]
```

juvenileNeighbour(...)

```
return true;
[Lines: 1]
```

growJuvenile(...)

```
double crownSurface = PI*sqr(canopyRadius);
double niMound = neighbourhoodIndexMounds/crownSurface; // TODO: replace by neighbourhood index
double niTree=0;
double niAdult = neighbourhoodIndexAdults/crownSurface;
double niJuv = neighbourhoodIndexJuveniles/crownSurface;
// increment in number of leaves
double[] proba = new double[3];
if (budHt>0.0) {
  double neighb = group_j0Mound*niMound + group_j0NLeaves*nleaves;
  double logit = neighb + group_aj0PdNneg;
  proba[0] = 1/(1+exp(-logit));
  logit = neighb + group_aj0PdN0;
  proba[1] = 1/(1+exp(-logit));
  logit = neighb + group_aj0PdN1;
  proba[2] = 1/(1+exp(-logit));
}
else {
  double neighb = group_j1Tree*niTree + group_j1Adult*niAdult + group_j1Juvenile*niJuv + group_j1BudHt*budHt;
  double logit = neighb + group_aj1PdNneg;
  proba[0] = 1/(1+exp(-logit));
```

```
  logit = neighb + group_aj1PdN0;
  proba[1] = 1/(1+exp(-logit));
  logit = neighb + group_aj1PdN1;
  proba[2] = 1/(1+exp(-logit));
}
double real = random.nextDouble();
int result = 3;
for (int i=0; i<3; i++)
  if (real<proba[i]) {
    result = i;
    break;
}
int dN = result-1; // possible outcomes: -1,0,1,2
// increment in bud height
double neighb = group_jpdhMound*niMound+ group_jpdhAdult* niAdult+ group_jpdhSNLeaves*sqr(nleaves);
double logit = neighb + group_jpdh0;
proba[0] = 1/(1+exp(-logit));
logit = neighb + group_jpdh1;
proba[1] = 1/(1+exp(-logit));
proba[2] = 1.0;
real = random.nextDouble();
result = 2;
for (int i=0; i<2; i++)
  if (real<proba[i]) {
    result = i;
    break;
}
double dH=0.0;
if (result==1)
  dH = 0.05;
else if (result>=2)
  dH = 0.2;
focalDrv.budHt = budHt+dH;
focalDrv.nleaves = nleaves+dN;
focalDec.canopyRadius = budHt+0.5+0.06*nleaves;
[Lines: 53]
```

mortalitySeedling(...)

```
// TODO: dummy:
double RUTree=1;
final double canopyRadius = 0.5;
double RUAdult = neighbourhoodIndexAdults/(PI*sqr(canopyRadius));
double logitTree = max(-10,min(group_sTreeInt + group_sTreeSlp*RUTree,10));
      double logitAd = max(-10,min(group_sAdInt + group_sAdSlp*RUAdult,10));
      // survival probability
      double psur = max(group_sMinSur,min(group_sMaxSur,min(1/(exp(-logitTree)+1),1/(exp(-logitAd)+1))));
return decider.decide(1-psur);
[Lines: 7]
```

dispersal(...)

```
double teta = random.nextDouble()*2*PI;
double r = sqrt(log(1-random.nextDouble()*0.6535)/(-group_dis));
otherCnt.x = x+r*cos(teta);
otherCnt.y = y+r*sin(teta);
otherDrv.neophyll = 1;
otherCnt.sex = random.nextBoolean();
[Lines: 6]
```

initSeedlingParameters(...)

```
focalCnt.sMinSur = 0.9;
focalCnt.sMaxSur = 0.97;
focalCnt.sTreeInt = 9.0;
focalCnt.sTreeSlp = -1200.0;
focalCnt.sAdInt = 5.8;
focalCnt.sAdSlp = -50.0;
focalCnt.recruitRate = 0.01;
[Lines: 7]
```

recruitSeedling(...)

```
return recruit.transition(decider.decide(group_recruitRate));
[Lines: 1]
```

growAdult(...)

```
// mortality
if (dead==0) {
  double logit = group_aPdead + group_deadNbLeaves  * nleaves ;
  double proba = 1/(exp(-logit)+1);
  if (random.nextDouble()<proba)
    focalDrv.dead=1;
  else
    focalDrv.dead=0;
}
else
  focalDrv.dead = dead+1;
if (nleaves==0)
  focalDrv.dead = 1;
// reproduction
if (sex) { // females only
  if (dead==0)
    focalDrv.nELSeedlings = group_fec*nleaves;
  else
    focalDrv.nELSeedlings = nELSeedlings*group_decay;
}
```

```
// growth
if (dead==0) {
  int dN = 0;
  if (random.nextDouble()<group_jPdNNeg)
    dN = -1;
  double dH = group_aGinc*(1-budHt/20);
  focalDrv.budHt = min(30.0,budHt+max(dH,0.0));
  focalDrv.nleaves = nleaves+dN;
}
[Lines: 26]
```

juvenileToAdult(...)

```
otherCnt.sex = sex;
otherCnt.x = x;
otherCnt.y = y;
otherDrv.budHt = budHt;
otherDrv.nleaves = nleaves;
[Lines: 5]
```

competitionJ(...)

```
// relativedistance weight, ie = 1 if at focal location, =0 at searchRadius distance
double weight = 1-euclidianDistance(x,y,other_x,other_y)/15.0;//searchRadius;
focalDec.neighbourhoodIndexJuveniles += weight*other_nleaves;
[Lines: 2]
```

adultNeighbour(...)

```
return true;
[Lines: 1]
```

mortalityJuvenile(...)

```
      double pSur = 1- group_jDeathRate;
      if (nleaves<=0.0)
        pSur = 0;
      else if (log(budHt+0.01)>=group_ja1+group_jb1*log(nleaves+group_jTol))
        pSur=0;
      return decider.decide(1-pSur);
[Lines: 6]
```

initAdultParameters(...)

```
focalCnt.aPdead = 1.21;
focalCnt.deadNbLeaves = -0.74;
focalCnt.remanence = 10.0;
if (focalCnt.remanence>0.0)
  focalCnt.decay = exp(log(0.01)/focalCnt.remanence);
else
  focalCnt.decay = 0.0;
focalCnt.fec = 4;
focalCnt.dis = 0.011;
focalCnt.jPdNNeg = 0.15;
focalCnt.aGinc = 0.2;
focalCnt.slrec0 = 0.01;
focalCnt.slrec1 = 0.01;
[Lines: 13]
```

moundNeighbour(...)

```
return true;
[Lines: 1]
```

seedlingToJuvenile(...)

```
otherCnt.sex = sex;
otherCnt.x = x;
otherCnt.y = y;
otherDrv.budHt = 0.0;
otherDrv.nleaves = neophyll;
[Lines: 5]
```

initAdult(...)

```
focalCnt.sex = random.nextBoolean();
// TODO: missing space limits !!!
focalCnt.x = random.nextDouble()*300;
focalCnt.y = random.nextDouble()*300;
focalDrv.nleaves = 15;
focalDrv.budHt = 10.0;
focalDrv.dead = 0;
focalDrv.nELSeedlings = random.nextDouble()*20;
[Lines: 7]
```

initMound(...)

```
focalCnt.x = random.nextDouble()*300;
focalCnt.y = random.nextDouble()*300;
[Lines: 2]
```

competitionA(...)

```
// relativedistance weight, ie = 1 if at focal location, =0 at searchRadius distance
double weight = 1-euclidianDistance(x,y,other_x,other_y)/15.0;//searchRadius;
focalDec.neighbourhoodIndexAdults += weight*other_nleaves;
[Lines: 2]
```

reproduction(...)

```
if (sex) {
  if (dead==0)
    return group_slrec0*nELSeedlings;
  else
    return group_slrec1*nELSeedlings;
}
else
  return 0.0;
[Lines: 8]
```

recruitJuvenile(...)

```
double logit = group_ajPR + group_j0BudHt*sqr(budHt);
double pRec = 0.0;
if (budHt>=8.0)
  pRec = 1-1/(exp(-logit)+1);
return recruit.transition(decider.decide(pRec));
[Lines: 5]
```