SimpactII Documentation

In following pages I will outline what SimpactII is and how to use it. In brief, SimpactII is an agent-based simulation model builder for stochastic simulations of HIV. It was developed by Lucio Tolentino (the author of this document), with help from Wim Delva, in 2013 using the open-source model building tool MASON[[1]](#footnote-1). I call it a model builder, as opposed to a model, because it is flexible as to what types of agents are placed in the model, how infection occurs in the model, and what interventions occur in the model – i.e., it’s a model builder. In this way we allow future modelers to implement vastly different models with ease.

Table of Contents

[1. Getting Started 3](#_Toc353959701)

[1.1 Setting up SimpactII 3](#_Toc353959702)

[1.2 Using the GUI 5](#_Toc353959703)

[1.3 Making a SimpactII Model 8](#_Toc353959704)

[1.4 Getting Some Output and Setting Parameters 10](#_Toc353959705)

[2. Details of SimpactII 11](#_Toc353959706)

[2.1 Constructors 11](#_Toc353959707)

[2.2 Methods 11](#_Toc353959708)

[2.3 Variables 12](#_Toc353959709)

[3. The Algorithm 14](#_Toc353959710)

[3.1 The Agents 15](#_Toc353959711)

[3.1.1 Agent 15](#_Toc353959712)

[3.1.2 Local Agent 16](#_Toc353959713)

[3.1.3 Homosexual Agents 17](#_Toc353959714)

[3.1.4 PTRAgent 17](#_Toc353959715)

[3.1.5 SexWorkerAgent 18](#_Toc353959716)

[3.1.6 Age Agents 18](#_Toc353959717)

[3.2 The Operators 18](#_Toc353959718)

[3.2.1 InfectionOperator 18](#_Toc353959719)

[3.2.2 Time Operator 18](#_Toc353959720)

[3.3 Interventions 18](#_Toc353959721)

[4. Known Issues 18](#_Toc353959722)

# 1. Getting Started

SimpactII was developed in Java for the sole reason that this is what MASON was developed in. While this gives SimpactII a lot of potential in terms of available libraries (additional useful data structures, algorithms, and the like) and in terms of speed (the default population of 1,000 individuals runs in less than a second), it also means that it can be difficult to understand what is going on unless you are familiar with the Java programming language. Because this documentation is about SimpactII, not Java, I assume some basic understanding of Java synatax. A basic idea of formating of if-statement, for-loops, and variable assignment is necessary for initial use. Advanced use (creating your own Agents or Operators) will require more familiarity with Java and programming, including understanding of the computer science buzzwords “inheritance”, “polymorphism”, and “anonymous classing”.

Additionally, as SimpactII relies heavily on the model builder MASON, it may be useful for advanced use to have a conception of how MASON works. Documentation for MASON is provided in the “documents” folder of SimpactII and has a nice tutorial that is really useful.

## 1.1 Setting up SimpactII

While SimpactII is written in Java, it is possible to write scripts in MATLAB and python (actually Jython to be precise). We’ll get to those later, first let’s dive into Java.

NetBeans is a Java Iterative Develop Environment (IDE) which helps you write Java applications. You can download it from <https://netbeans.org/>. SimpactII is distributed as a NetBeans project. You can download it from our git repository on GitHub: <https://github.com/seanluciotolentino/SimpactII/>. You can clone it via git (the git address is on the website), or if you don’t know what that means, you can download a zip file of the project with the nice ZIP button from the above link. Of course, other Java IDEs such as Eclipse work too.

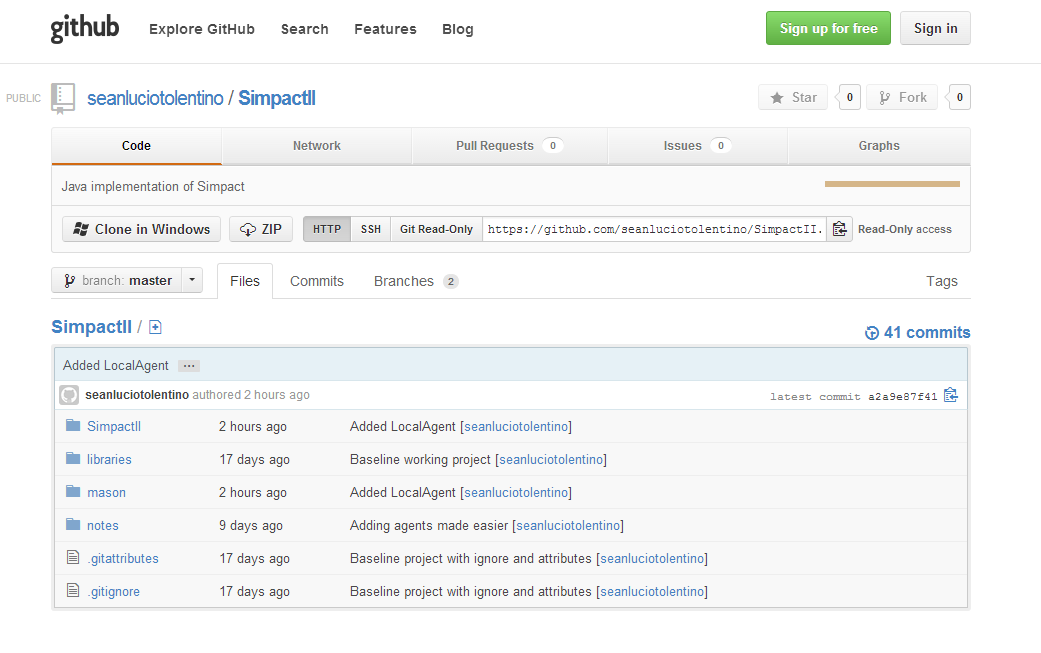
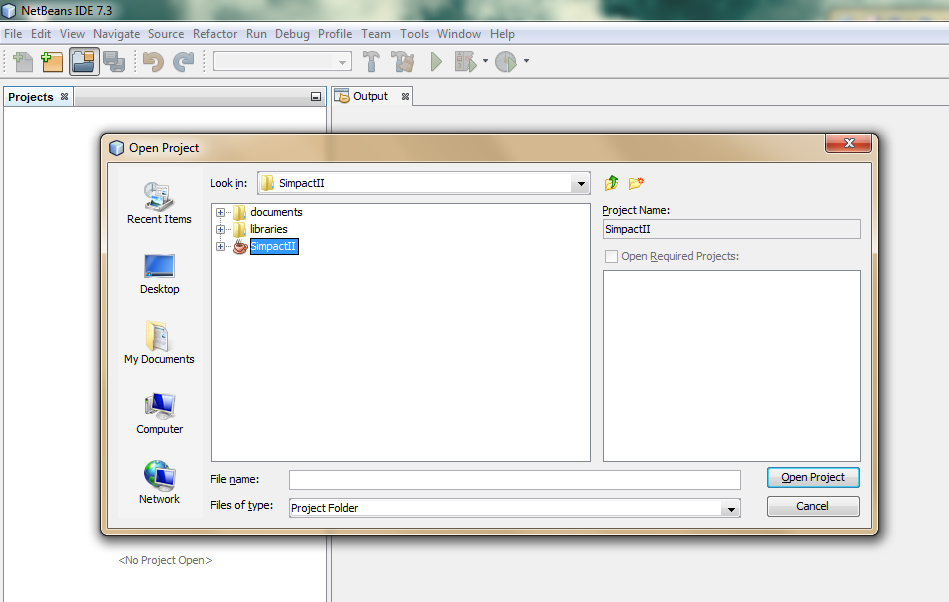
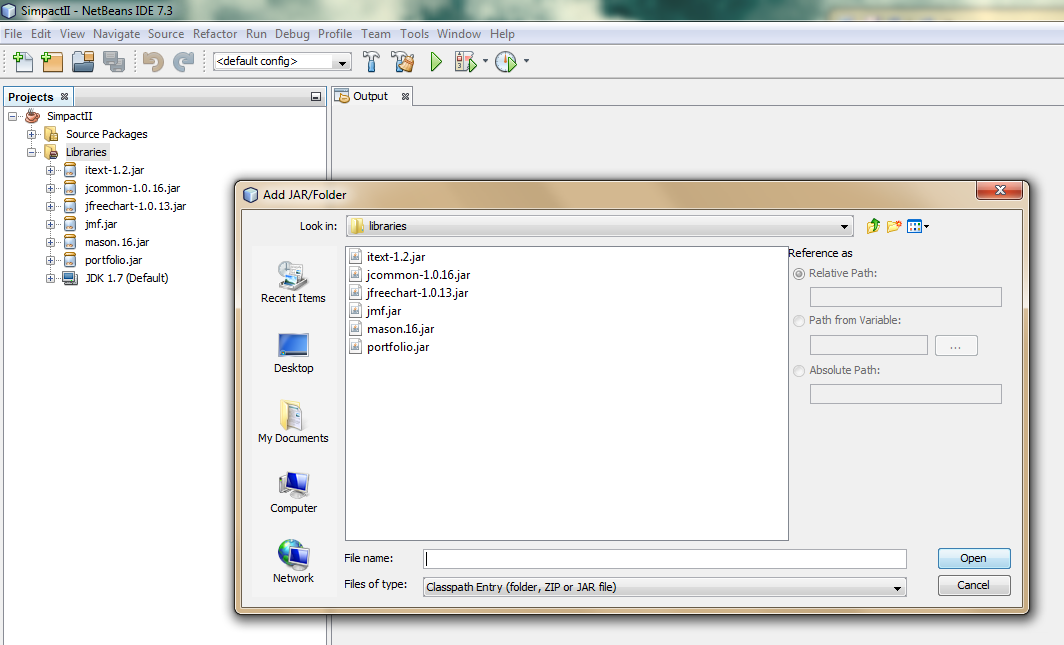


Figure 1: A screenshot of the GitHub website for SimpactII. Click on the ZIP button to download a zip file version of SimpactII, or clone it from the repository using the HTTP address.

I’ll assume that you are able to unzip the folder and place it somewhere convenient for you. Once you’ve installed NetBeans, open the program. You’ll see a pretty start screen. From here, open the SimpactII project.



Sometimes there’s a problem with NetBeans not knowing where the libraries are (even though I’ve told it a million times). To add the necessary libraries, expand the SimpactII project (with the plus sign) and right-click libraries. There should be an option to “Add Jar/Folder”.

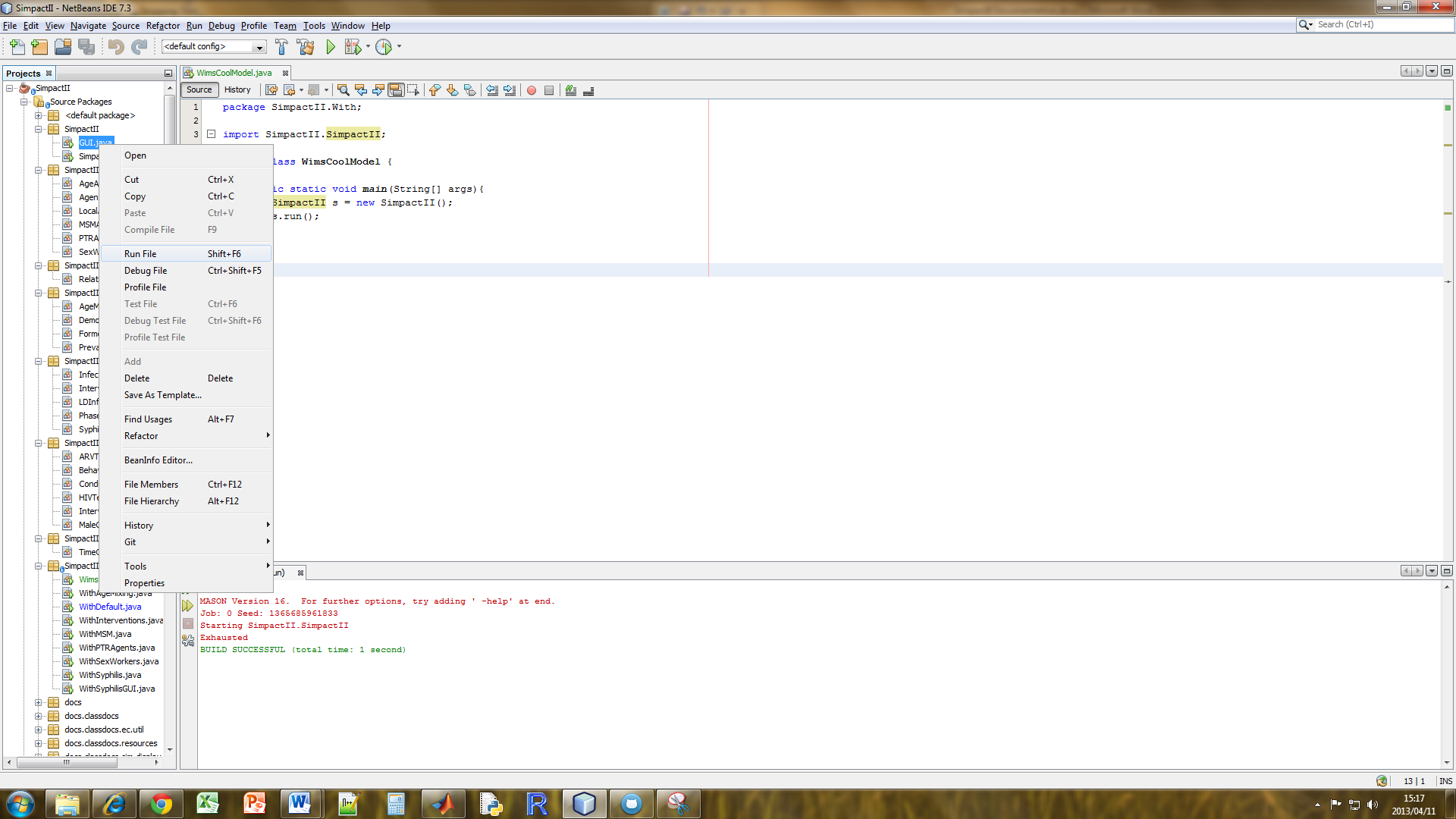


Add them all.

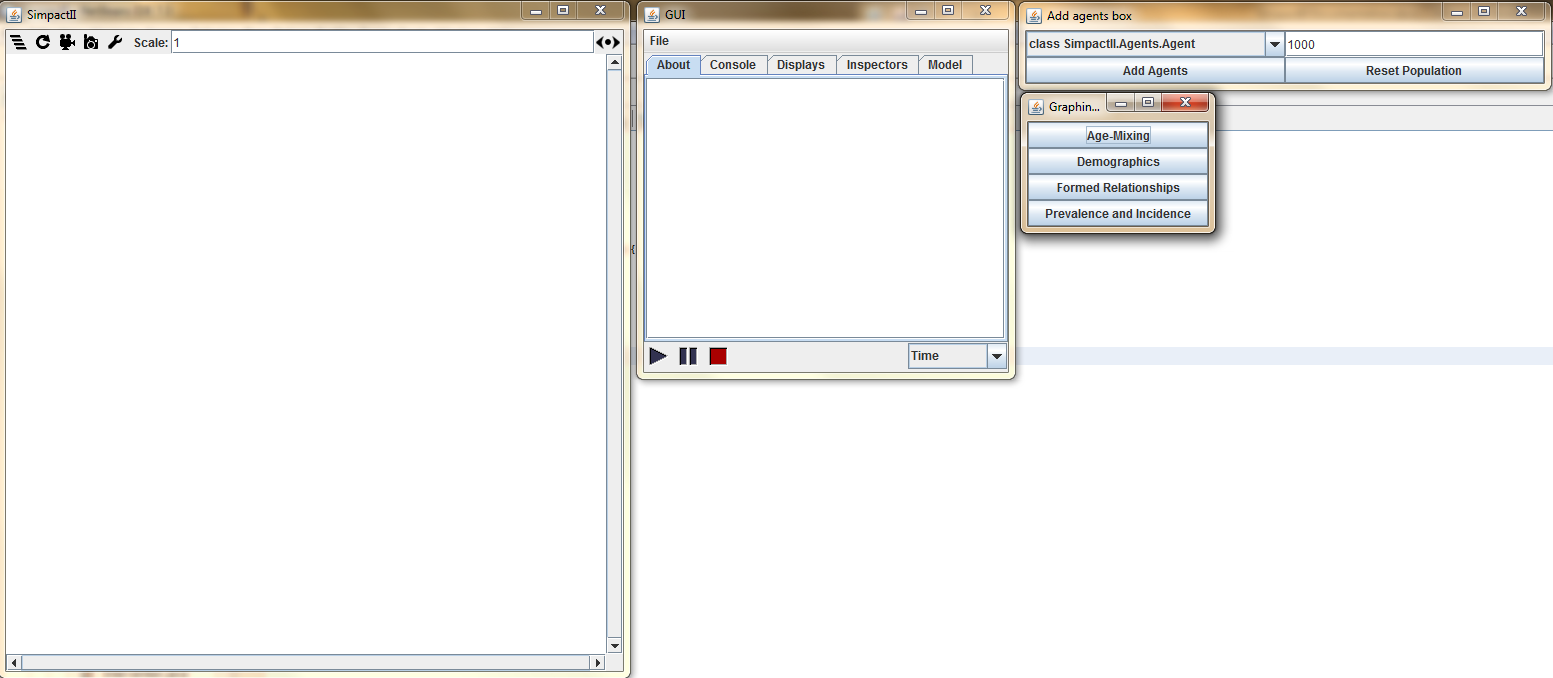
Now you can expand the Source Packages tab and view the source files for SimpactII. Now, you’re all set up!

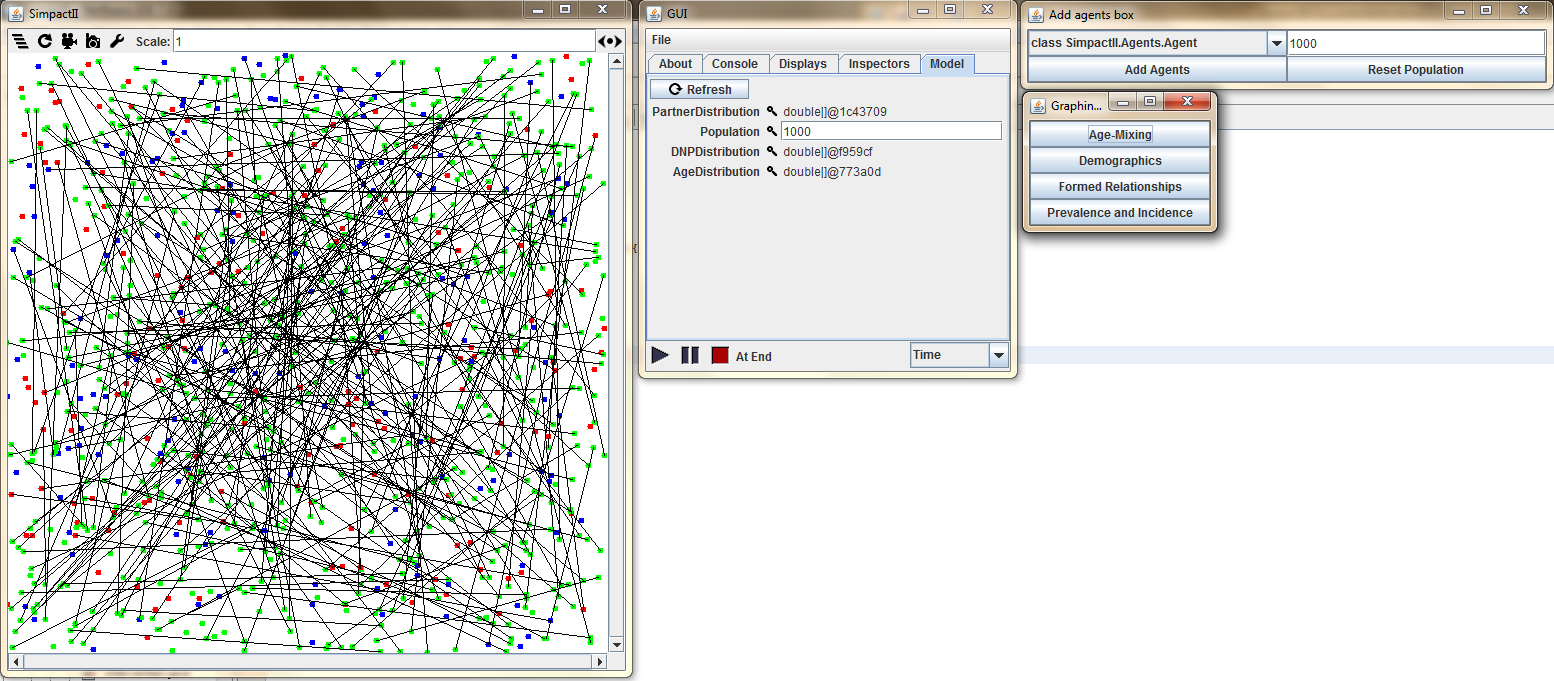
## 1.2 Using the GUI

Before building models, we’ll discuss how to use the GUI. Right-click “GUI” in the SimpactII package and run file.



The basic components of the GUI will pop up. If you did the MASON tutorial (like I said you should do), it’ll look familiar. Our GUI has a couple extra boxes though. The “Add Agents Box” lets you add specific kinds of agents to your model. Differences between agents will be discussed later. For now you can add the regular Agent. Click the play button and watch the sexual network and disease progression be simulated.





The simulation will automatically stop after 10 years by default. Note that blue dots are male agents, red are female agents, and green dots are infected agents. From here you can use the buttons in the “Graphing Options” pane to investigate what happened in the model. The buttons should produce graphs that look similar to this:

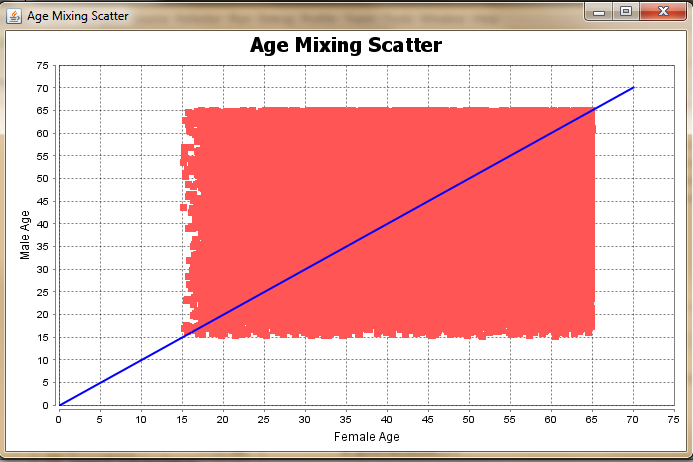


Figure 2: Age mixing scatter plot shows the age of female and male at the time of relationship initialization. If individuals formed relationships exclusively with individuals their own age, relationships would form along the blue line.

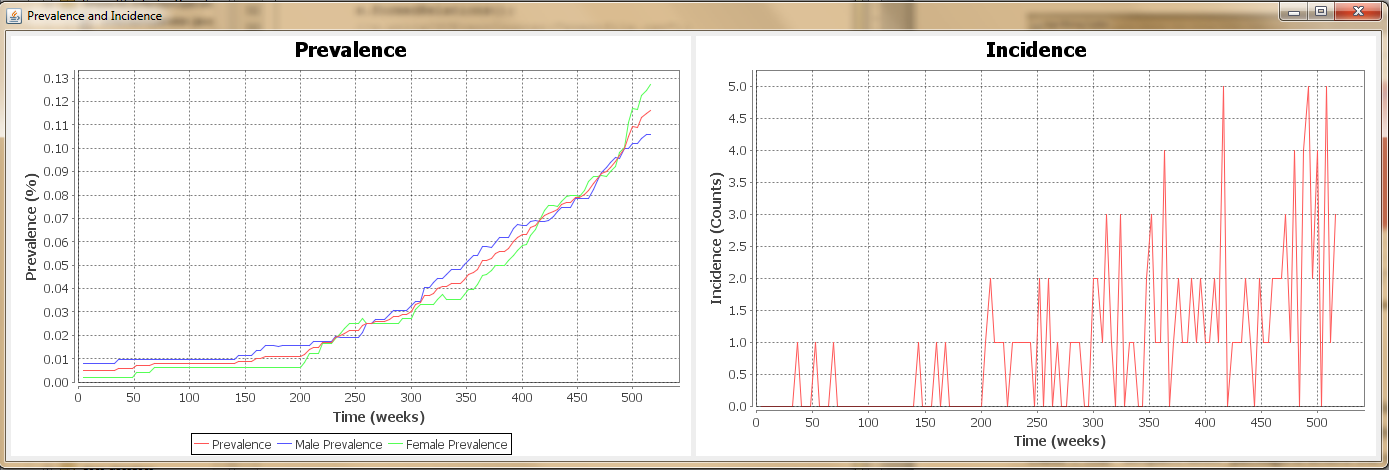


Figure 3: Prevalence (gender stratified, and population aggregate) and Incidence for the run of the model.

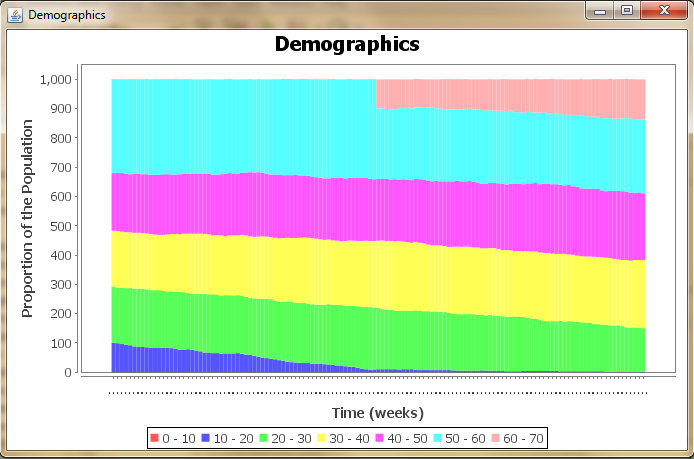


Figure 4: Age break down of agents in the model. Every timestep a stacked bar chart is created representing the demographics of the simulation.

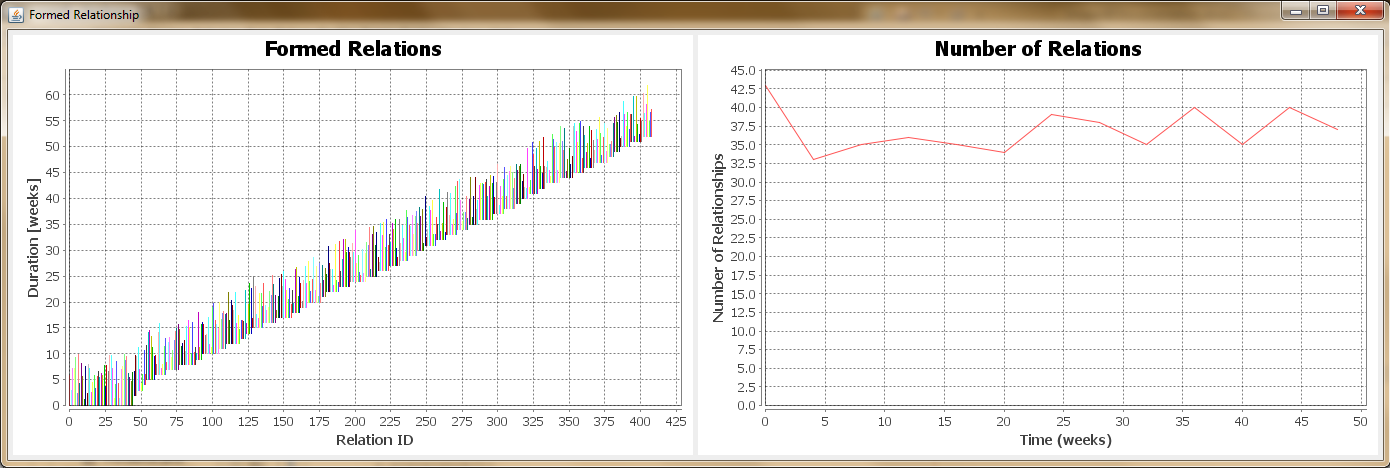
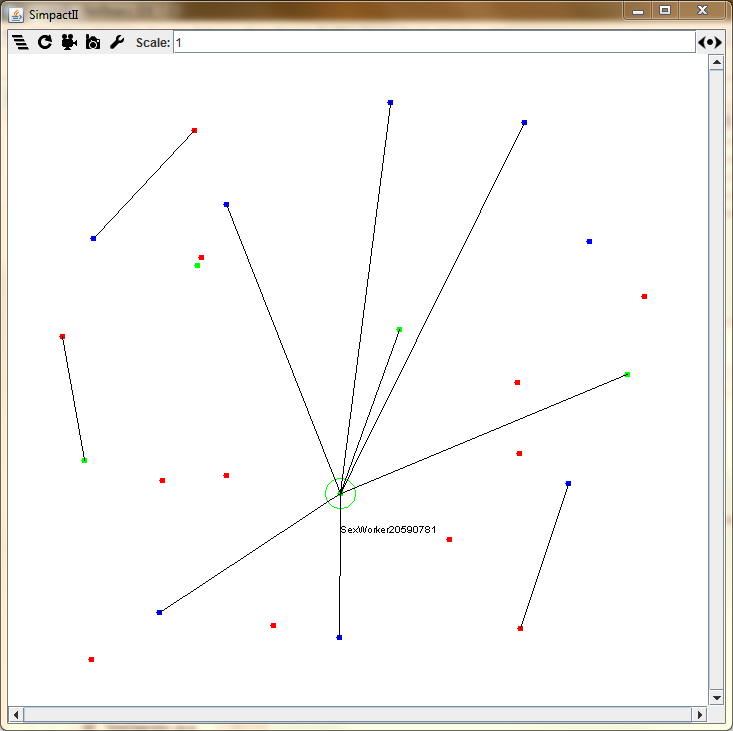
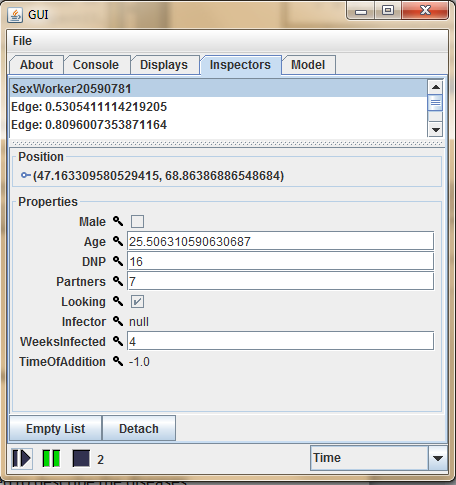


Figure 5: Formed relations and number of relations in the simulation. Each relation is represented as a line which denotes the duration of the relationship. Note that this is with a smaller population and run for less time in order to better visualize details.

Additionally, all of the nodes are click and draggable. When you click you can see what type the agent is. In the figure below I’ve selected a sex worker agent.



Double click and some attributes of the agent appear:

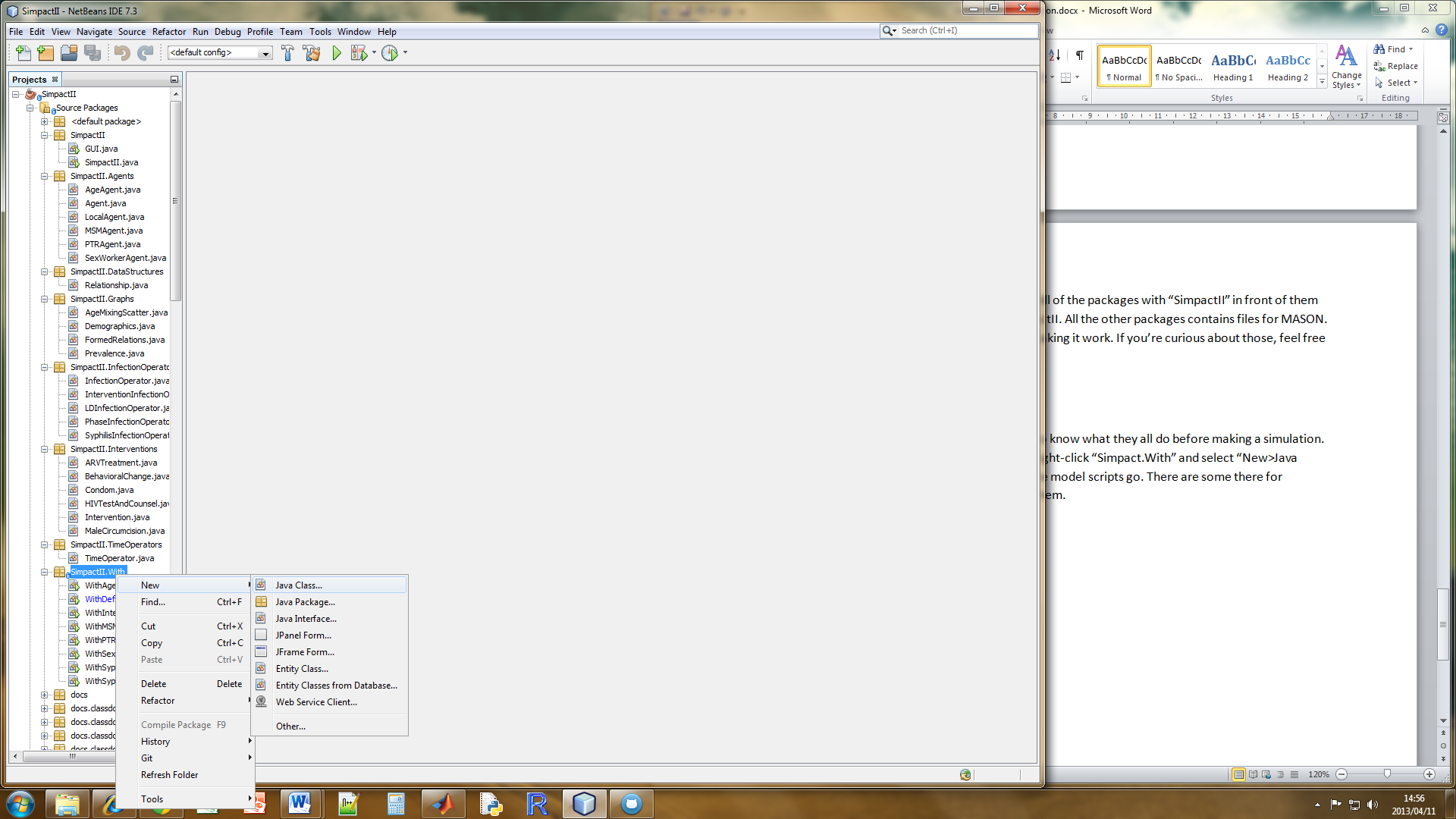


You can see the agents age, gender (no checkmark means this agent is female), desired number of partners (DNP), and other stuff. You can also check out the model tab to find out about the number of agents in the simulation, or the distribution of DNP or age.

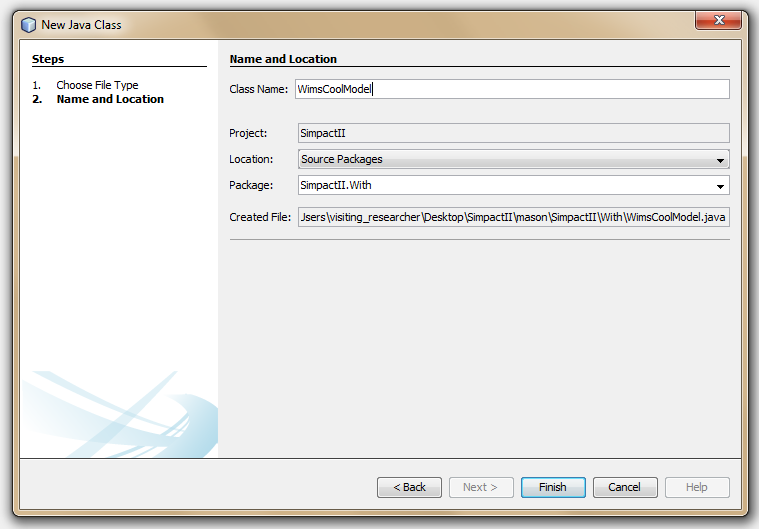
## 1.3 Making a SimpactII Model

The GUI is pretty cool, but it is limited in that when you create agents, you do not get to set their attributes (age mixing preferences, degree distribution) and you can not specify the length of the simulation. (You can however create a custom SimpactII model, as described below, and then launch the GUI for that implementation.)

The alternative, if you want to have a more detailed model, is to write your own. That is the focus of this section. Go a head and expand the “SimpactII” folders. There’s a lot of files there, but we do not need to know what they all do before making a simulation. So, to make your own SimpactII model, simply right-click “Simpact.With” and select “New>Java Class”. The “Simpact.With” package is where the model scripts go. There are some there for examples / testing. Feel free to get ideas from them.

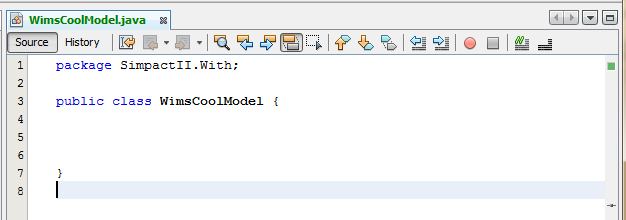


In the next dialogue you are asked for the class name. This is the name of your model. I am calling mine “WimsCoolModel”.



Click finish.

You’ll get something that looks like this:



From here on out, I will start putting the code into Word Text boxes so you can copy / paste and save some time. Now to actually do something with SimpactII we must create a SimpactII object. We do this in a main method. For non-Java users, this is the function that is called when you are running “WimsCoolModel”. In the main method, make a SimpactII object.

package SimpactII.With;

public class WimsCoolModel {

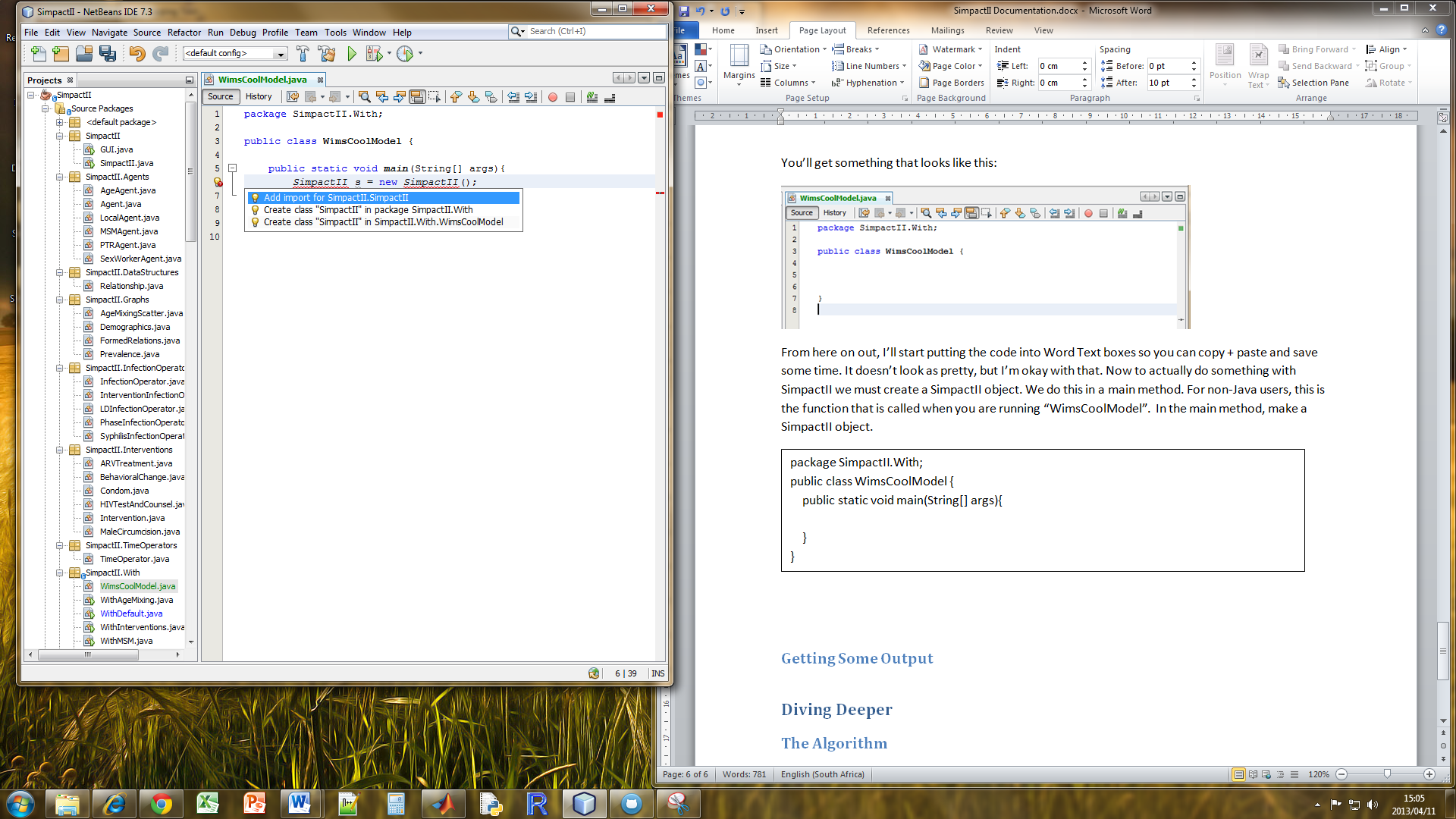
public static void main(String[] args){

**SimpactII s = new SimpactII();**

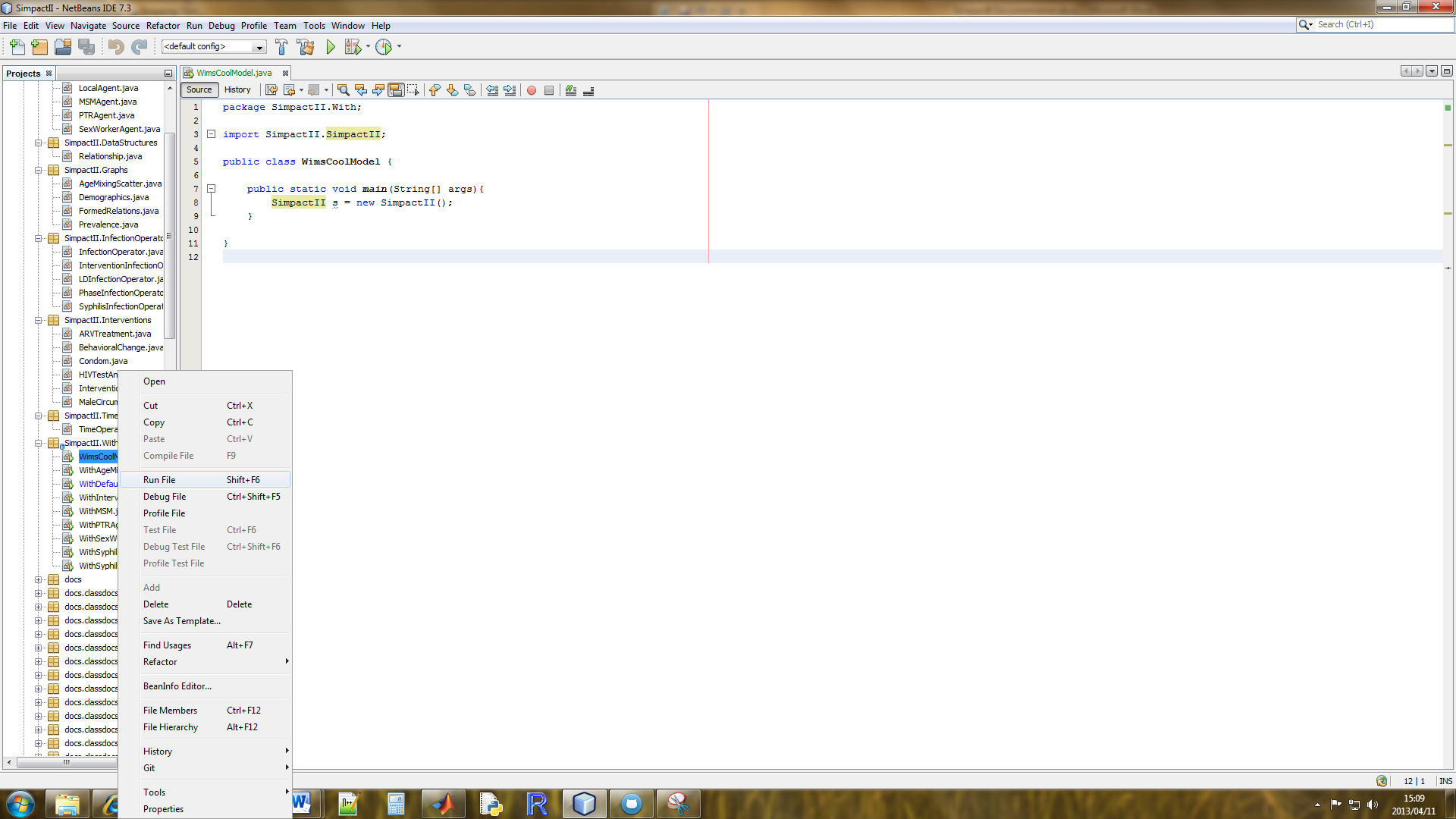
}

}

You’ll probably have your “SimpactII” underlined in red. This is because your model doesn’t know what a “SimpactII” is – you’ll have to tell it explicitly where to find it. I.e., you have to import SimpactII.SimpactII. The little lightbulb on the left (if you’re using NetBeans) can help: Click on the lightbulb and select “Add Import for SimpactII.SimpactII”. Netbeans will add the import line for you and everything should be okay. Note that this will happen a lot (you have to import from different projects all the time).



You can run your model by right-clicking “WimsCoolModel” and selecting “Run File”.



An output dialogue will appear that says “Build Successful”. That was not that interesting though because you did not actually run anything. Below your first line, now type s.run(). Run your model again.

package SimpactII.With;

public class WimsCoolModel {

public static void main(String[] args){

SimpactII s = new SimpactII();

s.run();

}

}

Now you’ll see a little more output at the bottom of your screen. Since you didn’t add any agents, you are running with the default values. In particular, 1000 Agents which form relationships based on a desired number of partners (power law distributed) and gender which runs for 10 years. In a bit we’ll talk about setting some parameter values and changing the simulation up just a bit.

## 1.4 Getting Some Output and Setting Parameters

Now we want to make those pretty graphs that we made when we were using the GUI. Additionally, we want to change the number of years the simulation is run for. To do this we call methods of the simulation and change the variables of the simulation:

public class WimsCoolModel {

public static void main(String[] args) { //for running from the command line

SimpactII s = new SimpactII(); //construct SimpactII instance

s.numberOfYears = 1; //setting a public variable / parameter of the model

s.run();

s.agemixingScatter(); //calling a method of the SimpactII object

s.demographics();

s.prevalence();

s.formedRelations();

}

}

For more information about these graphs see **Using the GUI** above. In the example above, you have *constructed* a SimpactII instance, set *public variables* (parameters) of the model, and called *methods* of the SimpactII instance. In the next section we provide more details of the constructors, variables, and methods of SimpactII.

# 2. Details of SimpactII

In the following sections we outline in more detail the constructors, methods, and variables in SimpactII.

## 2.1 Constructors

Constructors are how you actually create the basic model. There are four ways to do this as outlined in Table. Note that the first constructor is the one used in the example.

|  |  |
| --- | --- |
| **Call** | **Description** |
| SimpactII() | Default constructor: Reseeds the random number generator with the current time of the system and does not add any agents (this allows a user to specify agents to add – if no agents are added to the simulation, 1000 basic agents are added). |
| SimpactII( long seed ) | Reseeds the random number generator with the provided seed. |
| SimpactII( int population ) | Adds “population” number of Agents (basic agents) |
| SimpactII( long seed, int population) | Reseends the random number generator and adds “population” number of Agents to the simulation. |

Table 1: Table of constructors for SimpactII.

## 2.2 Methods

|  |  |
| --- | --- |
| **Method** | **Description** |
| start() | This method is called by the SimState main loop. To run the simulation, instead use the **run** method. |
| addAgents (Class agentClass, int N, HashMap<String, Object> attr) | Add N agents of type agentClass to the population. A HashMap<String,Object> may be provided if the agent class requires additional arguments. |
| addAgents(Class agentClass, int number) | Add N agents of type agentClass to the population. |
| addIntervention(Class c, double start, double spend) | Add an new intervention of type C with start time start and spending amount **spend**. |
| addIntervention(Intervention i) | Add an intervention **i** that has already been initialized. |
| resetPopulations() | Remove all the added agents from the lists. |
| formRelationship(Agent agent1, Agent agent2, double duration) | Adds an edge to the network between **agent1** and **agent2**. If **duration** is 0, the next value from the relationship distribution will be drawn. |
| dissolveRelationship(Edge e) | Removes the relation represented by the edge **e** from the network. Both acting agents in the relationship are informed so that each are able to update their own number of partners counters. |
| addAttribute(String key, Object value) | Adds the attribute **key** to every agent with the values **value**. |
| launchGUI() | Launch the GUI interpretation of this model. |
| agemixingScatter() | Generate the age mixing scatter plot. This is a scatter of all relationships, with age of male along the y-axis, and age of the female along the x-axis. |
| demographics() | Generate the demographics plot. This is a series of stacked bar charts denoting the distribution of ages at different time steps of the model. |
| formedRelations() | Generate the formed relations plot. This is a visual representation of the duration of relationships and the number of relationships at any given time. |
| prevalence() | Generate the prevalence plot. This is a time series plot depicting prevalence of HIV, stratified by gender, over time. Incidence (raw number of cases each week) is displayed in a second graph. |
| writeCSVRelations(String filename) | Generate a comma separated values (csv) file of all relationships which occurred in the simulation. |
| writeCSVPopulation(String filename) | Generate a comma separated values (csv) file of the details of individuals within the population. |
| writeCSVEventCounter(String filename) | Generate a comma separated values (csv) file of the time at which events occurred in the simulation. Currently the only event is infection, but will later include intervention events as well. |

Table 2: Table of methods for SimpactII.

## 2.3 Variables

|  |  |  |  |
| --- | --- | --- | --- |
| **Variable Name** | **Type** | **Default Value** | **Description** |
| world | Continuous2D | Continuous2D(1.0,100,100) | This is where the agents are placed. When initialized an agent generates a random coordinate in this space and places itself on the map. You can make it larger or smaller if the default of (100,100) is insufficient. |
| network | Network | Network(false) | The sexual network that forms from relationship formation and dissolution. In our model, the agents that are created are the nodes, and the relationships for the edges. Edges of the network are objects themselves that have methods that can be called. In particular, *edge*.getInfo( ) will return the weight of the edge which is the time till dissolution of the edge. |
| numberOfYears | double | 20 | The number of years that the simulation should run for. |
| degrees | Distribution | PowerLawDistribution(2.0,1) | The distribution where each of the agents should get their desired number of partners from. (Called by Agent constructor.) |
| ages | Distribution | UniformDistribution(15,65) | The distribution where each of the agents should get their age from. (Called by Agent constructor.) |
| relationshipDurations | Distribution | UniformDistribution(1,10) | The distribution where duration of relationships should be pulled from. (Called by the method formRelationship(…) ) |
| timeOperator | TimeOperator | TimeOperator() | More about this in Time Operator section. |
| infectionOperator | InfectionOperator | InfectionOperator() | More about this in InfectionOperator section. |
| myRelations | Bag | Bag() | The bag which keeps track of all the relationships that ever occur. It is a bag of “Relationship” objects which contain the two agents in the relationship and starting and stopping time. |
| myAgents | Bag | Bag() | The bag which keeps track of all the agents that ever existed in the simulation. |
| myInterventions | Bag | Bag() | The bag which keeps tracks of interventions that are too happen and have happened in the simulation. |

Table 3: Table of class variables of SimpactII.

Knowing these methods, constructors, and variables allows you to create many different models based on the basic agents. However, the real power of SimpactII is the flexibility and heterogeneity that it allows. In the next section I will outline the simple algorithm behind SimpactII and the different agents that can be added to the simlation.

# 3. The Algorithm

The algorithm behind SimpactII was inspired in large part by the Complex Agent Network model (CAN) developed by Sloot et al [add citation]. The idea behind the algorithm is a complex network approach to modeling HIV. In short, they have three operators which perform certain actions every time step, which in their model is one year. SimpactII inherits two of the operators from this model, the infection operator and the time operator. As the names should imply, these operators control infections that occur and the progression of time (individuals age are incremented, relationship durations are decremented). We break from their convention of a demographic-reshuffling operator which controls formation and dissolution of relationships so as to allow our agents to have more flexibility as to how to form relationships.

Our base model, from which we add complexity, initializes 1,000 agents each with a gender (gender ratio of 0.5), an age (uniformly distributed between 15 and 65), and desired number of partners (DNP – power law distributed 2,1). At each time step, which in our model is one week, we ask each agent whether if they would like to form more relationships and if so, with whom. The heterogeniety of our model comes from the fact that each agent has different criteria (as enumerated in 3.1 The Agents) for forming a relationship. In our base model, individuals form relationships if their number of partners is less than their desired number of partners. When a relationship is formed in the model, the duration is pulled from a distribution (uniform 1 to 5 weeks with the default).

After each agent is allowed to form relationships, the operators are called. The TimeOperator increments the ages of all agents who are still alive, and decrements the duration of relationship in the network. If any agent is above the threshold for maximum age (set at 65 years old in our model) they are removed from the system. If the duration of a relationship become negative (it has ended), the edge in the network is removed and the respective agents number of partners are decremented. On the next step they will try to find a new partner again.

The infection operator performs initial infections, increments the number of weeks infected (for infected individuals only) and performs infections in the simulation. By default, the probability of an HIV-postive individual infecting an HIV-negative individual in a single week is 0.02 (based on approximately 2 sexual acts per week). However, it is widely accepted that this is a naïve implementation since HIV infectivity is known to change with CD4 count and viral load in phases. For this reason simulations can use the PhaseInfectionOperator with takes primary, asymptomatic, and latent stage infection into account. More about infection operators in 3.2.1 InfectionOperators.

This process of first calling agents to form relations, second progressing time, and finally performing infections is repeated for the duration of the simulation (10 years be default). In this way a dynamic sexual network is formed and HIV is simulated to progress through the network.

In prose, the algorithm may be obtruse. For this reason we present the psuedo-code of the algorithm in Figure 6.

**Algorithm** *SimpactII*

1. **repeat**
2. //let agents form relations
3. **for** agent **from** 1 to N **do**
4. **if** agent.*isLooking*() **then**
5. **for** otherAgent **from** 1 to N **do**
6. **if** agent.*isLookingFor*(otherAgent) **then**
7. *formRelationship*( agent , otherAgent)
8. **end** otherAgent **for**
9. **end** agent **for**
11. //progress time with operator
12. timeOperator.*progressTime*()
14. //perform infections with operator
15. infectionOperator.*performInfections*()
16. **until** time > endTime

Figure 6: Psuedo-code for the SimpactII algorithm. At each step three things happen: (1) Agents with less than the desired number of partners form new relationships; (2) Time progresses such that agents ages are incremented and relationship durations are decremented by one week; (3) Infections occur in serodiscordant relationships.

The algorithm outlined above is simple and is what makes SimpactII easy to use. However, this simplicity should not be confused for lacking complexity. The calls to the agents “*isLooking*” and “*isLookingFor*” methods allows us to have a heterogenous population in which agents have different attributes that they are “looking for”. For example, our basic agent “is looking” when his / her partners is less than his / her desired number of partners, and he / she “is looking for” an agent of the oppisite gender that “is looking”. The differences between the agents and how they form relationships are outlined below.

## 3.1 The Agents

The current version of SimpactII has several agents which can be used in a simulation. The first is the basic agent which all other agent inherit characteristics from. Table 4 indicates the methods of a basic agent. The other agents use the methods of the basic agent unless otherspecified in their table of methods. Additional methods (that are specific to the agent and not overriding the basic agent) are in *italics*.

Note that agents have an “attributes” table: A hash table where additional attributes of an agent can be stored. For example, a LocalAgent (3.1.2 Local Agent) has the concept of a radius around him or her from which to form relationships. Additional attributes that are specific to the agent are underlined. Some examples are provided.

### 3.1.1 Agent

|  |  |
| --- | --- |
| **Method** | **Description** |
| step(SimState s) | Usually called by the scheduler, this method lets the agent perform the actions as he / she would in a week. |
| possiblePartners(SimpactII state) | Specifies which agents this agent is interested in forming a relationship with. The default agent is interested in all other agents. |
| isLooking() | Base check to see if this agent is interested in forming relationships this round. |
| isLookingFor(Agent other) | When going through the bag of possible partners, ask this agent if he / she is looking for the agent other. |
| isSeeking(Agent other) | Called when another agent wants to form a relationship with this agent. Indicates the question, "does this agent want to have a relationship with other agent?" |
| informRelationship(Agent other) | When a relationship is formed this method is called in order to increment this agents number of partners. If the agent has a preference for the length of the relationship, a non-zero double will be returned, else zero will be returned. |
| informDissolution() | Decrement number of partners. |
| remove() | Return whether the removal criteria of the agent has been met. |
| replace(SimpactII state) | Returns an agent which replaces this agent after he/she has been removed. |

Table 4: Methods of a basic Agent. All other agents inherit these methods or override them.

### 3.1.2 Local Agent

This kind of agent forms relationships with other agents that are only within a certain radius of him/herself. The difference in the sexual network produced is apparent from the visual representation in Figure 7. To do this, the agent overrides the methods in the table below.

|  |  |
| --- | --- |
| **Method** | **Description** |
| possiblePartners | Returns partners that are within radius of the agent. Radius is a settable attribute of the agent. |
| isSeeking | Returns whether other agent is within the radius of this agent. Additionally, the default isLooking() and heterosexual preference must be met. |
| getRadius() | Returns the radius attribute for this agent. |

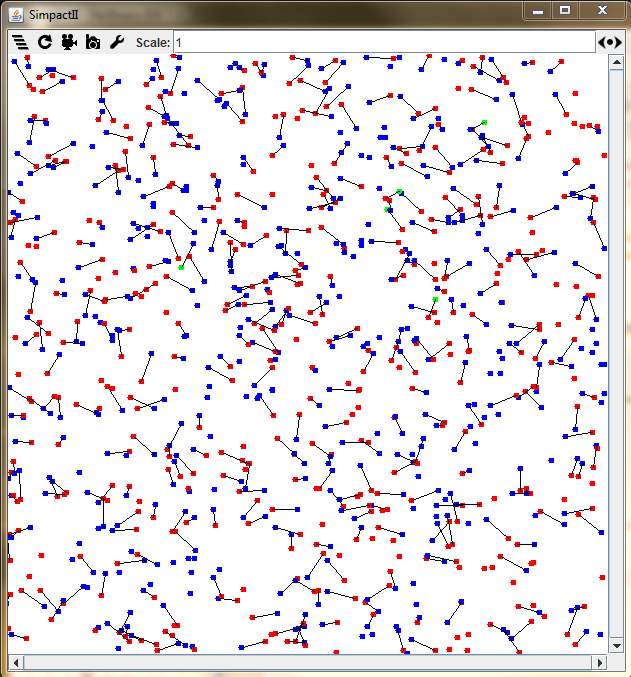
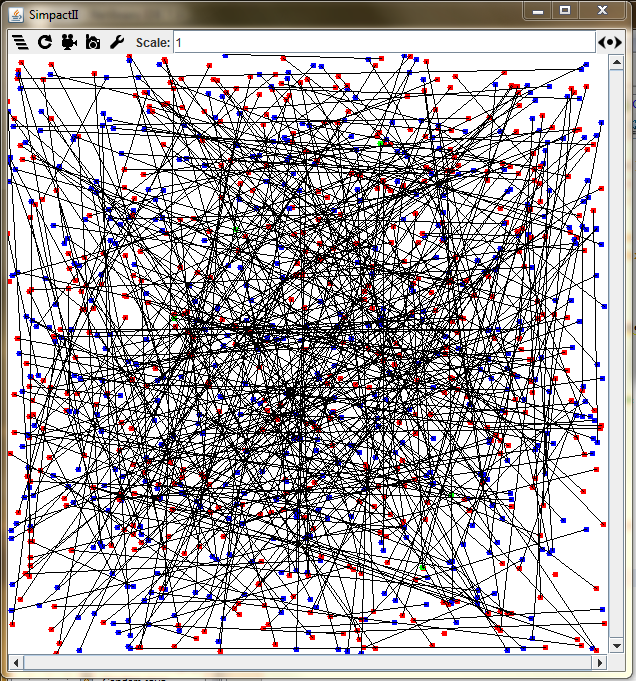
 

Figure 7: A visual representation of the difference between LocalAgents (left) and basic Agents (right).

### 

### 3.1.3 Homosexual Agents

#### MSMAgent

This kind of agent represents men-who-have-sex-with-men (MSM). These agents are initialized as male and form relationships exclusively with other MSMAgents.

|  |  |
| --- | --- |
| **Method** | **Description** |
| isLookingFor | Returns true if the other agent is male, and seeking this agent (a male agent) |
| isSeeking | Return true if partners is less than desired number of partners and the other agent is male. |

#### BiAgent

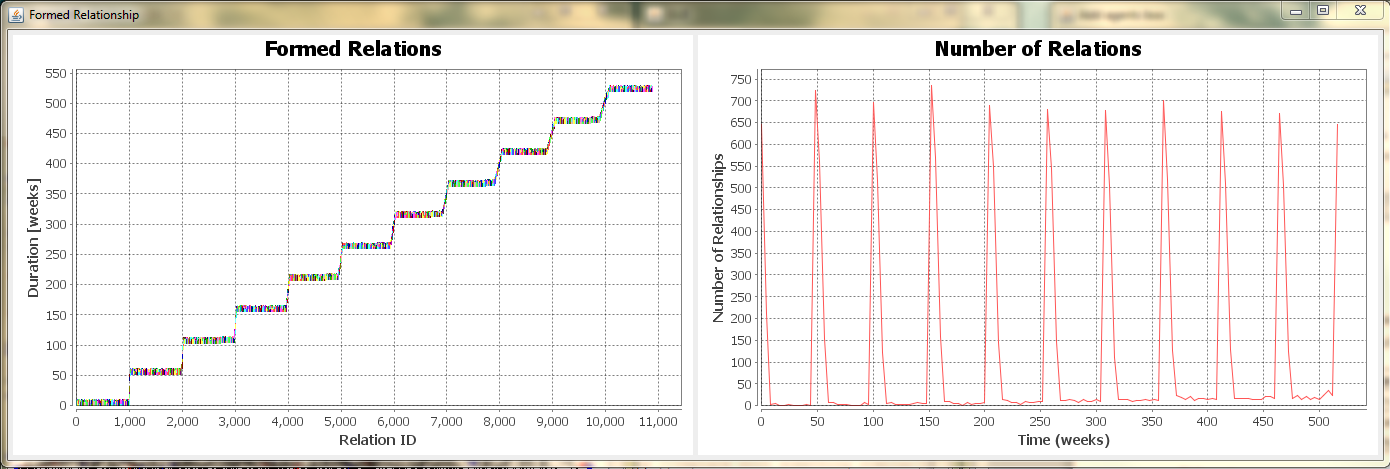
This kind of agent represents agent who from relationships with both men and women. These agents, similar to MSMAgents, are initialized as male, but form relationships regardless of sex.

### 3.1.4 PTRAgent

This kind of agent allows a user to specify the partner turnover rate (PTR) of the agent. Practically this means that a user can set the partnersPerYear attribute of an agent, and the agent will not form more relationships than this. If a value for partnersPerYear is not provided, a default value of 2 is used. This works by changing the isLooking method so that individuals are only interested in forming relationships if they have not exceeded their number of partners for the year. In the implementation this happens by an anonymous agent reseting the number of partners this year to zero at the beginning of each year (every 52 weeks).

|  |  |
| --- | --- |
| **Method** | **Description** |
| isLooking | Returns true if the agent has not exceed their number of partners this year. |
| informRelationship | Increments the number of partners this year then calls the super.informRelationship (does whatever the parent class does). |
| replace | Returns an agent with the same attributes as itself |
| *resetPartners()* | Resets the number of partners to zero. |

The graph below shows the relationship dynamics of 1000 PTRAgents with partnersPerYear equal to the default of 2. As you can see agents form many relationships in the beginning of the year, quickly exhaust their number of partners per year, and are unable to form more relationships until the beginning of the next year. This overly emphatic example is a product of short relationship durations which mean relationships end quickly and hence partners per year quotas are used up quickly.



Below is an example of how to create a model with PTRAgents that have a different number of partnersPerYear. In the example we create a SimpactII object like normal. We create a HashMap and put “partnersPerYear” with the value “3”. Then when we add our special PTRAgents, we pass the additional argument of the hashmap “attr” that we created.

package SimpactII.With;

import SimpactII.Agents.PTRAgent;

import SimpactII.SimpactII;

import java.util.HashMap;

/\*\*

\* @author Lucio Tolentino

\*/

public class WithPTRAgents{

public static void main(String[] args){

SimpactII s = new SimpactII();

HashMap attr = new HashMap<String,Object>();

attr.put("partnersPerYear",3);

s.addAgents(PTRAgent.class, 100, attr );

s.numberOfYears = 10;

s.run();

s.formedRelations();

}

}

### 3.1.5 SexWorkerAgent

This kind of agent represents behaviour of a sex worker. The agent is initialized to be female, and has a desired number of partner of 16: She can have as many a sixteen partners in a week. However, each relationship lasts only one week. SexWorkerAgents, addition to always being female, are between the ages of 15 and 30. Once a SexWorkerAgent turns 30, she is replaced by a female basic Agent. When the basic Agent is replaced (usually at the age of 65), she is replaced by a SexWorkerAgent by default. This keeps the number of sex workers, and ex-sex-workers constant through out the simulation. Below are methods which a SexWorkerAgent overrides.

|  |  |
| --- | --- |
| **Method** | **Description** |
| informRelationship | Returns the next value from the sex worker relationship duration (default is one week) |
| remove | Return true when the sex worker agent reaches the MAX\_AGE (default is 30 years old) |
| replace | Returns an Agent to replace her at MAX\_AGE. More technically, the basic agent is actually an anonymous class of Agent which overrides the replace function such that the replacing agent is replaced by a sex worker agent. |

### 3.1.6 Age Agents

This kind of agent is conscious of his or her age, and what it implies for the relationships that he or she wants to form. In the real world, which we are trying our best to simulate, individuals form relationships with individuals of a similar age. For this reason we have created a few different age-based agents which we describe below.

#### BandAgeAgent

This kind of agent forms relationships based on the attributes band and offset. The offset determines how far away from “perfect mixing” (agent only forms relationships with individuals of the same exact age) the agent is. Note that this offset is from a male perspective: An male agent with an offset of -2 indicates that he prefers agents that are two years younger than him; A female agent with an offset of -2 indicates that she prefers male agents who are two years older. By similar logic, a male agent with an offset of 2 prefers female agents two years older, and a female agent with an offset of 2 prefers male agents two years younger.

The band indicates the plus/minus around the offset. A 25-year-old male with a band of 2 years, and offset of -3 prefers female agents between the age of 24 and 20. Note that preference here indicates that an individual will form a relationship with probability 1.

Below is an example of the probability of a 30-year-old male forming a relationships with females of different ages. His offset is -5, and his band is 5. This means he will attempt to form relationships exclusively with women that are between 20 and 30 years old.



Figure 8: Example of the probability of 30-year-old males probability of forming a relationship with a female of a certain age. In this example, the male has an offset of -5 and a band of 5.

#### ConeAgeAgent

Since the band / offset age agent can be somewhat restrictive, we developed the ConeAgeAgent. This agent has a preferredAgeDifference (age difference with the highest probability), a preferredAgeDifferenceGrowth(the rate at which preferredAgeDifferencegrows with age – older agents have a larger age preference), preferredAgeDifferenceDispersion (the amount which preferred age difference matters) and probabilityMultiplier (which indicates the shape of the probability curve). The probability of a male agent *i* and female agent *j* is then given by:

Where is the probabilityMultiplier, is the age difference between the two, is the preferred age difference, is the mean age of the couple, is the preferredAgeDifferenceGrowth, and is the preferredAgeDifferenceDispersion.

Figure 9 presents a visual representation of this equation. Figure 10 shows how the dispersion and growth variables effect preferred age difference for different age combinations. Note that since the agent must calculate the mean and age difference for every attempt to form a relationship, the simulation runs slower with these agents.



Figure 9: A graphical representation of the probability of relationships forming for different probability multipliers. In this example the preferred age difference is 5 after accounting for dispersion and growth with mean age.



Figure 10: How preferred age difference can change with dispersion and growth. Here the baseline preferred age difference is 0.5, preferred age dispersion is 0.01, preferred age growth is -1.0, and the probability multiplier is -0.01.

## 3.2 The Operators

The operators are the “housekeeping” agents of the simulation. While not agents in the sense that they form relationships with other agents, they are Steppable objects (implement the Steppable interface from MASON). At each timestep, after agents have been allowed to form relationships, these agents perform actions which progress the simulation. Each are described in more detail in the following subsections.

### 3.2.1 InfectionOperator

The Infection Operator, as the name implies, the infection operator performs infections in the system. In short, the operator iterates through agents, and when it finds an HIV-positive, transmits HIV to each of the agents partners with some probability. After performing probablistically the indepedent infections, the operator increments the number of weeks the agent is infected by 1 week.

In this way agents form relationships, and the infection operator transmits HIV through these relationships. The fact that these two operations are handled by separate operators is convenient in that it allows us to change how infection occurs without changing how relationships occur (in computer science speak, the program is modular).

The default probability of transmission per sexual act is 0.01 (the weekly probability is 0.02). However, as eluded to in [Section the algorithm], the probability of transmission is not necessarily constant over time. There is evidence that infectivity is relative to phase of infection. For this reason we have implemented the PhaseInfectionOperator which returns a different probability depending on the number of weeks the individual has been infected. The default values are given in the table below, but are, of course, adjustable.

|  |  |  |
| --- | --- | --- |
| **Phase** | **Length of Phase** | **Probability of Transmission** |
| Primary Infection | 12 weeks (3 months) | 0.032 |
| Asymtomatic | 384 weeks (8 years) | 0.0035 |
| AIDS | Till death | 0.0152 |

Table 5: The default lengths and probability of transmission for the phases of HIV.

In addition to the flexibility of transmission with respective to number of weeks infected, we can customize other events that occur parallel to HIV infection. By this, I mean that the transmission of other sexually transmitted diseases (STDs) can be model in parallel to HIV. As an example, we consider the STD syphilis which is a curable bacterial infection which causes a large sore (called a chancre – named specifically for the ancient disease) and rashes.

The SyphilisInfectionOperator mimics the operations of the default InfectionOperator, but performs syphilis infections with some probability as well. This is an extremely useful feature for a couple reasons: (1) We can model multiple STDs in addition to HIV, and (2) we can investigate the effect of an STD treatment intervention (since co-infected with other STDs like syphilis increase the probability of infection).

### 3.2.2 Time Operator

The Time Operator, as the name implies, modifies time specific variables such as age and relationship duration. More specifically, the TimeOperator is called every time step to increment each agent’s age by one week, decrement each relationship’s duration by one week, and perform the removal and replacement of individuals whom meet the criteria.

The default value for replacement is 65. This means that once an individual reaches this age they are removed from the system. They are replaced either by an agent of the same class (the default constructor is called), or an agent specified by the agent’s “replace” method. This is discussed more in Section 3.1.1 Agent.

## 3.3 Interventions

Of course, one of the fundamental reasons for modeling HIV is to model interventions aimed at disrupting for spread. In SimpactII implementing interventions is easy and similar to adding agents to the simulation. The example below shows how to add a condom distribution campaign which starts at the beginning of year 2, and spends $1,000 to your simulation.

public class WithInterventions {

public static void main (String[] args){

SimpactII s = new SimpactII();

s.addAgents(Agent.class, 1000, ageAttributes );

s.infectionOperator = new InterventionInfectionOperator(s.infectionOperator);

s.addIntervention(new Condom(2,1000));

s.run();

s.prevalence();

}

}

Figure 11: Example of adding an intervention to a simulation.

The example in Figure 11 shows how to add a condom distribution campaign to the simulation. To do this a new “Condom” object is created which has the arguments 2, and 1000 passed to it. These are the start and spend amounts. We also must change the infectionOperator for the simulation to an “InterventionInfectionOperator”. This special “InfectionOperator” augments the transmission probability by some probability change.

In the sections below we detail the implementation of some of our interventions. This our only basic implementations – more complexity may be desirable for more sophisticated models.

3.3.1 Condoms

3.3.2 Male Circumcision

3.3.3 HIV Test and Counsel

3.3.4 Antiretro Viral Treament

3.3.5 Behavioural Change Campaign

3.3.6 Additional interventions

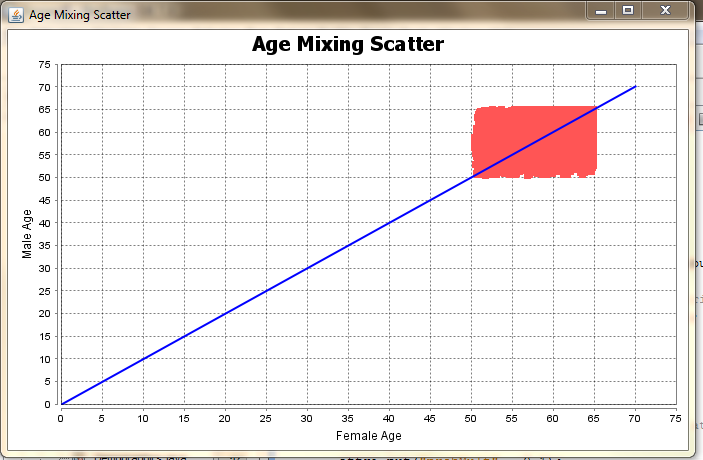
# 4. Examples and Uses

Sooner or later you’re going to find that the base functionality that SimpactII has is not quite good enough. Luckily, SimpactII was written to be flexible to new effects and subpopulations. Below you’ll find tips and suggestions for using SimpactII to fit your needs.

### 4.1 Writing custom agents / operators

### 4.2 Writing custom CSV Files

# 5. Known Issues

* WriteEventCSV needs to be updated – does not include intervention events.
* Attributes are copied over to the replacement agent – including infectivity / infectivity change
* AIDS death only augments time till death by most recent stint of ARV use (does not account for individuals whom are on and off repeatedly).
* Local agents will only form relationships with agents if they are the initiator (the basic agent is returned during a possiblePartners method call). If the basic agent “seeks” the local agent, the result will be false because the basic agent does not have a location attribute, even if basic agent is within local agents radius.
* This sometimes happens: 
* There’s no such thing as a MultiAgent – an agent that follows multiple roles

1. <http://cs.gmu.edu/~eclab/projects/mason/> [↑](#footnote-ref-1)