odd2netlogo - Project Report

ODD
Alexander Magnus Nordnes Strand, Eivind Aamodt
https://github.com/uiano/odd2netlogo

IKT445
19.12.2021

Abstract

NetLogo is a programming language that is used to create simulations and models, but many of the users who want these models are typically not familiar with development. Currently they have to fill in an ODD template that describes what they want to simulate, and then another developer has to create the NetLogo code based on the specification. Odd2netlogo is a programming language which goal is to skip out on the last step. Instead of needing someone else to manually write the code from the specifications, the odd2netlogo program should generate the code automatically.

When work was started on this project the language had already been in development for quite a while so most of the functionality was already implemented. Because of this, it was decided to focus on making some general improvements to the user experience. These improvements mostly consisted of two types; namely editor changes and constraint changes. We have added many instructions such as ”press ctrl+enter” or ”press enter to ...” to make the user experience more streamlined. The constraints we have added are mostly in the initialization part, to stop users from inputting invalid values which would not have worked. The odd2netlogo program should be accessible to everyone, so making the process from start to finish as simple as possible to follow is the number one goal.

1 Introduction

NetLogo is a programming language that is used to create simulations and models through a model called agent-based modelling. These simulations can for example be biological simulations regarding how different blood cells will move and how sheep populations will be affected by releasing wolves into an ecosystem, or social science models about how segregation takes place in a neighborhood.

One of the big problems is that the people who study biology and similar things are typically not inclined to learn how to code just to use this one program. Currently this problem is solved by them filling in all of the specifications in an ODD template, and then the code part is offloaded to other people who then have to implement it based on the specification. An image of the ODD template can be seen in Figure 1.

Outline		Guiding questions	Own ODD+D Model description
	I.i Purpose	I.i.a What is the purpose of the study?	The purpose of the study is to explore under whether university students can experience intense marginalisation (which is how we defined bullying) by the process of selecting interaction partners.
		I.ii.b For whom is the model designed?	For everyone interested in inclusion- exclusion and bullying phenomena.
	I.ii Entities, state variables, and scales	I.ii.a What kinds of entities are in the model?	Agents reflect students and links are the unidirectional relationships of each student with the other.
		I.ii.b By what attributes (i.e. state variables and parameters) are these entities characterized?	Attributes of an agent: **External-characteristics :** the agent's characteristics readily available by just looking at the other agent such as the hair colour (array of numbers), **Internal-characteristics :** the characteristics that are exposed during interactions such as shyness (array of numbers), Tolerance: the range around the ideal value of a characteristic that the agent perceives as acceptable, (same for all characteristics) **#negative interactions:** the number of negative interactions the agent has experienced since the beginning of time , **#positive interactions:** the number of positive interactions the agent has experienced since the beginning of time, **#refused interactions:** the number of refused interactions the agent has experienced since the beginning of time, Attributes of links: **Attraction** for the agent the link is directed to, **Known-indices:** memory for other agent's characteristics,
		I.ii.c What are the exogenous factors / drivers of the model?	We assume that external to the agents, there are idealise values for each external and internal characteristic (**Ex_ideal_chars** and **In_ideal_chars** respectively). Agents

Figure 1: ODD Protocol

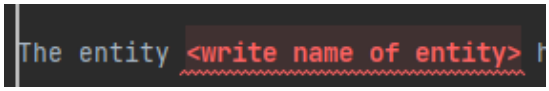
odd2netlogo's goal is the replace the last step where someone else has to sit and write the code that is specified in the ODD schema. Instead what we want is to generate the NetLogo code automatically using MPS. Writing these ODD schemas can take a very long time to do, and if the editor can give suggestions when the user is entering code, it will make the process a lot smoother than it currently is. The editor has to be easy to use, as the main goal is to make agent-based modelling accessible to people who have never written any code before.

2 Language

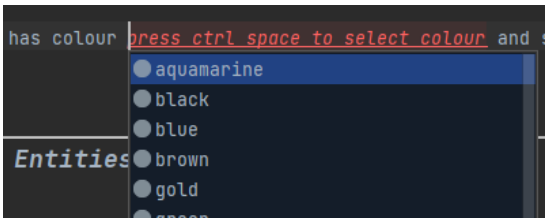
The language odd2netlogo is a language that is run in the Jetbrains MPS editor. One important part about the language is that the user should be given a drop-down menu

with values to choose from. The tool should be designed so that each step of the editor should be self-explanatory, and the fields should be able to be filled in without having any previous development experience, mainly by utilizing the drop-down menus and using default values in some parts.

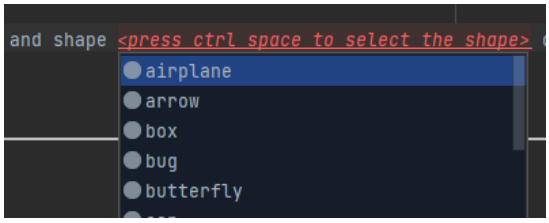
As an example, the step-by-step process of creating an entity can be seen in Figure 2a. The process is self-explanatory as the fields give hints about what the user should do, so it is simple to understand for people who have never written code before.



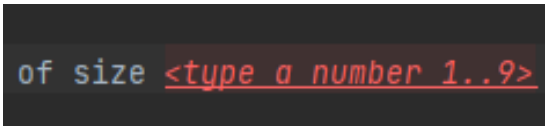
(a) Step 1 - write a name



(b) Step 2 - give it a color from the drop-down menu



(c) Step 3 - give it a shape from the drop-down menu



(d) Step 4 - give it a size

Figure 2: Creating an entity

The first part of the editor is a General Description of the project. Here the user can write what the research question is, what the purpose of the model is et cetera.

The second part of the editor is the Entities, state variables and scales part. Here the user has the ability to add the different entities which are going to be used during the simulations and all of the properties they have. Some examples can be an entity named "sheep" with a property named "hunger".

The third part is Process overview and scheduling, which is where the user adds actions to their entities. These actions can foe example be "move", "eat grass" and similar actions. The actions should mention which entities are supposed to do the actions.

The fourth part is Design concepts, where the user adds rationales and interactions to the different entities and attributes.

The fifth part is Manual Experiments AKA Initialization, where the user should specify how many entities should be created, what the initial attribute values should be, to add some starting values to the simulation.

The last part is Experiments. Here it is important to specify what data should be collected from the simulations and which values should be sampled and at what rate, so it can be used to create graphs during the run and to monitor how changes made to the model and/or the initialization affect the chosen values.

After the user has filled in all of these fields, the odd2netlogo program should convert the inputs into executable NetLogo code. The user does so by right-clicking on the file

and selecting "preview generated text". After the code has been made, the user can simply copy-paste the code into NetLogo where it should be run correctly.

3 Solution

3.1 Structure

The language was already fairly developed by the time the project period started. This includes the structure being near finished, and it was advised against making significant changes to the structure. Thus there were no new concepts, relationships or references.

However there were added some constraints. For example one such constraint was related to the initialization of entities. Previously there had been no constraints on what value the initial amount of entities could be. This does not make sense as there cannot be half an entity or a negative number of entities. Constraints limiting the amount to only positive integers was thus added.

Another constraint was added to some sliders. A slider should not be able to have a granularity of zero, as this would result in the slider being meaningless.

When it came to the attributes it was possible to set upper and lower limits on their values upon creating them. However there was no check in the initialization of said attributes to prevent the user from typing in a value out of range. Thus a constraint was added here as well.

The last constraint was quite challenging to implement but we eventually managed to do it with help from our supervisors. The problem we were facing was that we did not know how to access the upper and lower values from the parent node. An image of what we were trying to do can be seen in Figure 3.

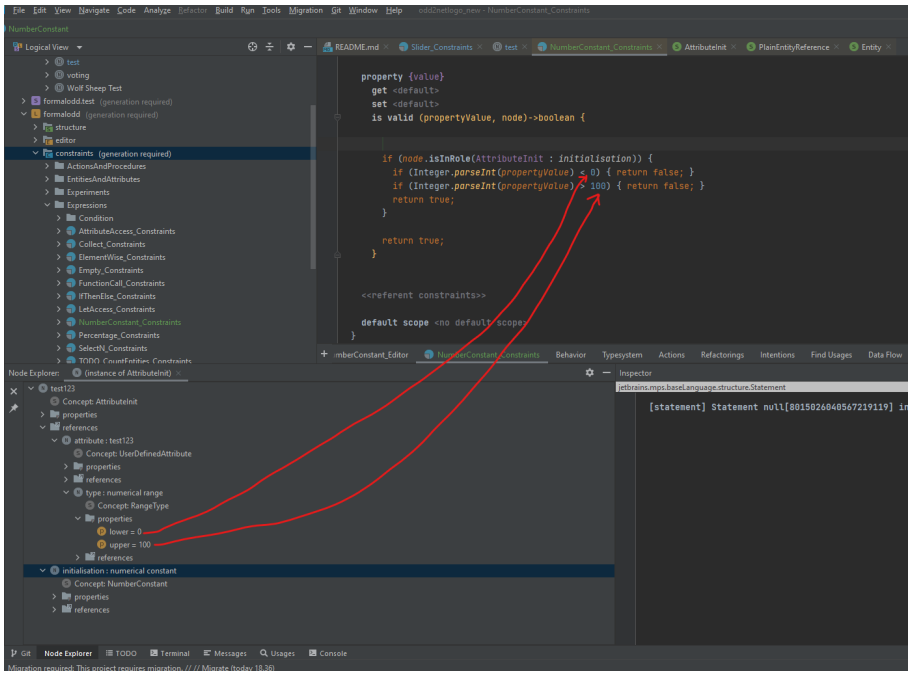


Figure 3: Upper and lower limit

When trying to figure out how we would access the values, we made a syntax tree to better visualize the process. This helped us narrow down the thinking process regarding which nodes we needed to access, and a drawing of the syntax tree can be seen in Figure 4

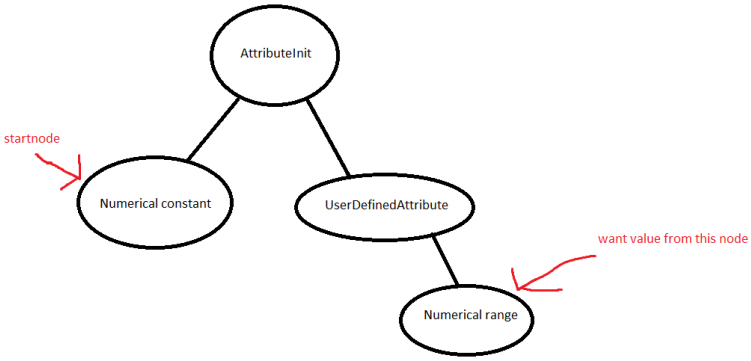


Figure 4: Syntax tree

The code we ended up with was quite complex. First, we had to access the parent node, and then we had to traverse down the node tree until we eventually accessed the numerical range. This was a challenging task because of the lack of documentation and the fact that it was our first time working with the Jetbrains MPS editor. While challenges like these can normally be solved through the use of documentation and searching on the internet, these challenges are domain-specific which makes the process that much harder. The code we ended up with after getting guidance from our supervisors, can be seen here:

```
property {value}
  get <default>
  set <default>
  is valid (propertyValue, node)->boolean {
    if (node.isInRole(AttributeInit : initialisation) &&
        node.parent:AttributeInit.attribute.type.isInstanceOf(RangeType)) {
      if (Integer.parseInt(propertyValue) < Integer.parseInt(node.parent:AttributeInit.attribute.type:RangeType.
          lower)) { return false; }
      if (Integer.parseInt(propertyValue) > Integer.parseInt(node.parent:AttributeInit.attribute.type:RangeType.
          upper)) { return false; }
      return true;
    }
```

Figure 5: Constraint code

3.2 Syntax

As a major reason for the language existing being to make it easier to write in than pure NetLogo some of the focus of this project has been on said part. Particularly when it came to instructing the user on what to do. The user generally has three ways of interacting with the program. They can either press Enter, which is usually used when adding a new element, which can range from creating a new entity, to adding another step to the schedule. They can press ctrl+space, which is used when the user should get a dropdown menu of their options within a template. Finally there are some cases where the user is just expected to type in the text themselves, like typing out an integer or float. However it is not always obvious which of these options are the correct ones to use, so instructions were added as placeholders.

These instructions were implemented in the Editor. Most of them were implemented through the Instructor, with adding the proper instruction in the text* cell. In some cases it was also necessary to add a show-if condition in case the instruction should only show up if certain conditions were met. These were most often to check if there already existed a similar node, or if the node already had at least one of a given child.

3.3 Semantics

Model2Model transformations are at the core of ODD, as the user will write in ODD and output as in the NetLogo model. However as these transformations were already complete by the time the project started the group has not looked closely into how these transformations function.

4 Discussion

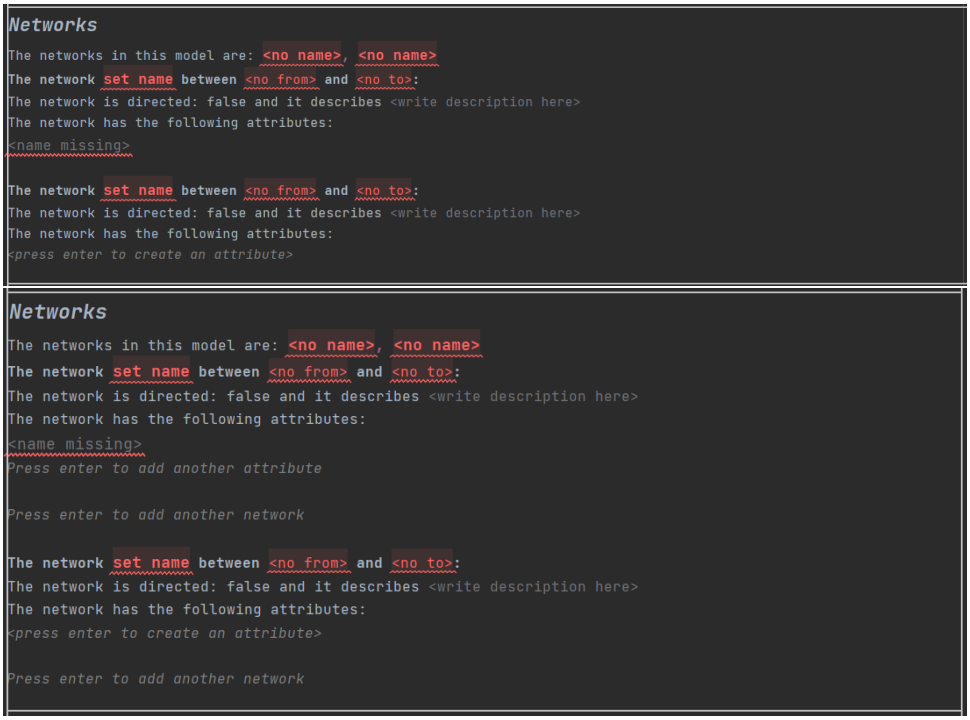


Figure 6: Old vs new Editor. Instructions added on how to create new networks and to create new attributes for each network. "placeholder text for the attributes themselves yet to be implemented

The implementation covered a large part of the language before the project started. NetLogo code examples like wolf-sheep was already possible to recreate in ODD. The focus was therefore less on the functionality, and more on making sure the user would use the language correctly by improving the editor and constraints. In that regard the implemented changes can be seen as improvements. The results can be seen in in examples like Figure 6, which showcases that the changes work as intended. As can be seen from the figure the instructions for adding what kind of attribute is missing, instead there

is just a red empty line. The problem here was that `GeneralAttribute` implements `IDescribedConcept` which puts limits on how the editor as they are generally meant to be abstract. There were some suggestions to look into base language to see how similar scenarios were solved there, but unfortunately there was not enough time for that. There were few considerations that had to be made, as it was often explained by the supervisor how she wanted certain things to look or function.

When creating the constraints it was very important to access the correct nodes. When you initialize the number of entities (for example number of wolves) the value is oftentimes set using a slider. This is because when you have a slider it is possible to change the values between different runs, and to observe how the initial number affects the outcome of a given program. If there are 500 sheep but only one wolf, the sheep may be able to survive. But if there are 200 sheep and 50 wolves, the outcome will be very different.

But the sliders are also oftentimes used when setting different values for attributes. This can for example be how much energy the sheep regains from eating grass or how much energy it expends when it moves. While the number of entities should only be integers, the attributes can contain floats. Because of this, it is not as simple as putting a constraint on the slider to only allow integers, as it should have different rules based on what type of value it is modifying.

The slider contains four different values: `Start value`, `Min value`, `Max value` and `granularity`, and all these values needed different constraints depending on the context. This was done using if-tests and the `isInRole` function. The `isInRole` function checks if the current node is modifying the value of a given type, for example `initialisationNumberOfEntities` which is the number of entities. Because of this if-test the slider value constraints will behave different when modifying the number of entities and when modifying values for a given attribute. The code for `start value` can be seen in Figure 7

```
property {startValue}
  get <default>
  set <default>
  is valid (propertyValue, node)->boolean {
    if (node.isInRole(Entity : initialisationNumberOfEntities)) {
      if (propertyValue.contains(".")) { return false; }
      if (propertyValue.contains("-")) { return false; }
      return true;
    }
    return true;
  }
```

Figure 7: Constraint start value of slider

There was some implementations which where discussed, but did not end up being implemented in the end. These include a change to the structure regarding certain default values. This could be seen with the `specialEntities`, where the default entity is `first partner`. In many cases this can be confusing for the user, as it is not evident that they can change it to another partner. However there are cases where having it default to the default is useful, as seen in Figure 8; it is useful to have the user select the partner in

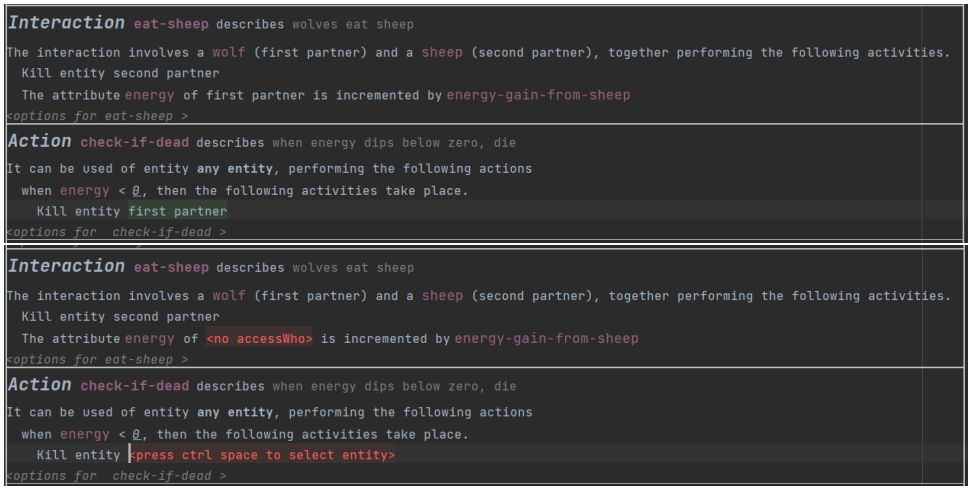


Figure 8: Example with and without default value

Kill, however for the case of the increase function it is more questionable. If the default had to be removed however it would only result in the user having to write more code from the experiments done. However as that would be a change to the structure it would mean we would also have to change the code in all the examples so they now function properly, which we unfortunately did not have time to do. There was also a suggestion that a default value would be set in the editor under specific circumstances, but there was not enough time to look into that.

There was also some tasks there was not enough time to complete. These include not having the time to implement constraints on which attributes that show up in the menu. In ODD it is possible to have global attributes, but also attributes that should only be accessable to either environments or entities. However this is not considered with the current implementation, and so entities can access attributes which should be only available to environments and vice versa.

An example of this can be that the initial value of ”hunger” can be set to ”hunger” itself. Having the ability to initialize attributes based on other attributes values is an essential part of the simulations, such as hunger being a value that is based on the entity’s energy attribute, so the ability to utilize other attributes should not be disabled. But the value should not be able to be based on itself at it would be an undefined value. Therefore the ability to initialize ”X = X” should not be allowed, and is listed under future work.

5 Plan and reflection

After suggestions from the supervisor the first step was for all the group members to familiarize themselves with the language through recreating one or more of the examples. In addition to getting to understand how the language worked it was useful to find where certain instructions were missing and other similar shortcomings. Afterwards the observations were presented to the supervisor and it was discussed which would be the best to work on. Originally there were three group members working on the project, however one member had to leave the project which resulted in less workforce than

desired and initially planned for. It was then discussed if a wide variety of different tasks would be looked into, or if most of the chosen tasks should be similar to each other. The conclusion was that due to the group being shorthanded more similar tasks would be worked on. From then on most of the tasks were given on a weekly basis during supervisor meetings. Within the group it was decided that both would look over the task list without formally assigning specific tasks to each member. That way both members would get a better understanding of the language, and was able to help each other out. This worked out fine at the start, but probably resulted in some wasted hours as both members looked into the same tasks as time went on. However it did end up with one member focusing more on constraints while the other more on the editor.

Not as much work was done in the end as would have been desired. One reason being that the group underestimated how long it would take to properly understand how the language was built up to implement certain changes, which was a much bigger task than simply figuring out how to use the language. Another reason was that both members choose to prioritize more pressing hand-ins and exams in other courses for periods, which resulted in less than the suggested 70h to work on the project as a whole.