



ODD - Project Report

Group - ODD

by

Raphaël CHIRON, Nasr JABER ,

Alix LEVRIER, Louis ROMETTE,

https://github.com/uiano/odd2netlogo/tree/ODD_MPS_FallSemester

in

IKT-445

15th December 2023

Generative Programming

Department of Engineering sciences

University of Agder

Contents

Abstract	2
1. Introduction	3
2. Language	4
A. ODD protocol	4
B. ODD2Netlogo solution	4
C. Example of how to use ODD2Netlogo solution	5
3. Solution	6
A. Structure	6
a. Stagnation condition	6
b. SelectN	7
B. Syntax	8
a. Stagnation Condition	8
b. Entity access	8
c. Action call	9
d. SelectN	9
e. Experiments	10
C. Semantics	11
a. selectN	11
b. Action call	12
c. Timed Condition and Quantified condition	13
4. Discussion	13
A. implementation	13
B. An improvement who works well	14
C. Future step and alternatives solution	14
a. SelectN	14
5. Plan and reflection	17
6. Question for the MPS learning quiz	17
7. References	19

Abstract

The aim of our project was firstly to identify areas for improvement in relation to the ODD language based on MPS 2021.2, and then to attempt to modify certain functionalities in order to improve the language. This language is a generation language, taking written specifications and translating them into NetLogo code, the main aim being to provide lambda users with an interface enabling them to generate code. The main approach of our project was to attempt to reproduce various models already present in the NetLogo software libraries, to identify the problems encountered, and then directly modify the various structural, editor and generation/transformation elements of the language. We could observe the generation elements directly on the NetLogo software, after having generated the code in .nlogo format. Overall, we succeeded in making a number of modifications, enabling a better user experience for new users.

1. Introduction

As part of the Generative Programming course, we had to implement a project. This consisted in taking the ODD language in hand through various examples including that of "Fire", "Voting", and "Fire with agent" and simulations via the NetLogo software, then improving and developing them according to different criteria. We had to familiarize ourselves with the software and learn this new programming language. Throughout the project, we encountered a number of errors and problems that we resolved by satisfying the various expectations.

Regarding the structure, this report is first composed of an abstract and an introduction to present the work done in order to have a general idea. The second part concerns the language in which we described the ODD language as well as the one generated to use it under NetLogo and that we therefore implemented. The third part shows the different solutions implemented around three main axes which are: structure, syntax and semantics. The fourth part of this report reflects the various discussions we have had. In other words, the part in which we put forward our project with the improvements we are most proud of, the problems we have encountered as well as the current problems. The fifth part concerns the plan and the reflections around this project, that is to say the organization within the team, the meetings, the communication or even the time that this project took us. Finally, the last part contains in one page the references.

2. Language

A. ODD protocol

The ODD (Overview Design Concepts and Details) protocol is a set of documentation requirements aimed to adequately define agent-based models. It is a standardized document for providing a consistent, logical and readable workflow for agent-based model design. It was launched in 2006 and has since gained widespread acceptance in ecology and other fields. [1]

Even so, there are still limitations to ODD that obstruct its more widespread adoption. For example, authors of ODD model descriptions may struggle with the limited availability of guidance on how to use it and the lack of sufficient details of many ODDs models make it very difficult to implement their code as it provides some ambiguity. [2]

B. ODD2Netlogo solution

To solve the problems encountered with ODD protocol Andreas and Themis have developed a solution called ODD2Netlogo. The aim of this solution is to be able to create social experience simulations more easily than using the ODD protocol. Some advantages of ODD2Netlogo are that it allows you to run as many experiments as you want and to operate without real time.

ODD2Netlogo is an MPS-based software package that allows users who are new to the world of coding to experiment with social simulations digitally. ODD2Netlogo can be divided into several main categories, ranging from global descriptions of model entities to more precise properties. It allows the user to enter these characteristics of his simulation in precise places. You will find in Fig.1 a screenshot of ODD2Netlogo solution.

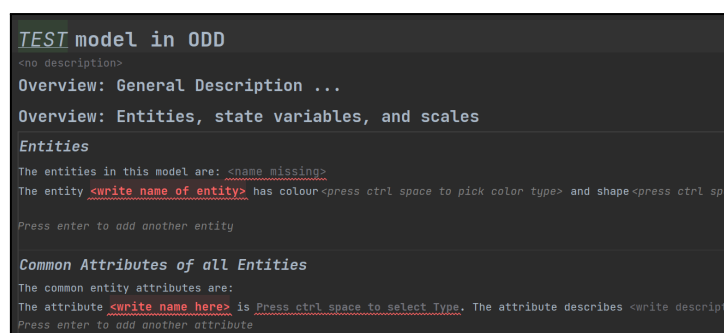


Figure 1: Screenshot of ODD2Netlogo solution

So, as you can see from the screenshot, all the places where you can see red ripples are places where the user has to fill in some information about his model. To do this, they can use the comments (in grey) to help them understand what is expected from him.

One of the problems encountered with the ODD protocol is that several people with different backgrounds are working on the same experiment, which can lead to ambiguity when it is implemented. By using ODD2Netlogo this problem disappears. Our people with social knowledge can easily code their simulation themselves and avoid any ambiguities.

C. Example of how to use ODD2Netlogo solution

Let's take the Wolf Sheep example to have a deeper understanding of how it works. First, the user will have access to a screen with a general description of the model, entity declarations, scheduling planning, etc. and he will have to fill in the different sections, as shown in the figure below.

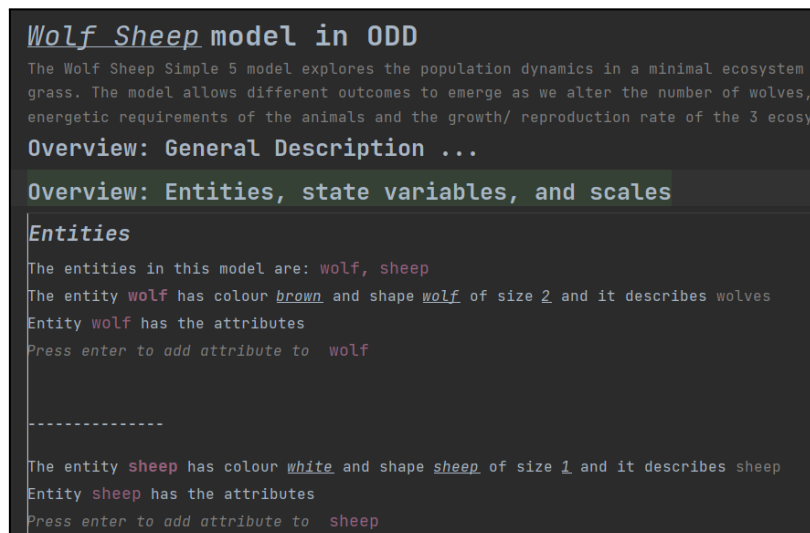


Figure 2: Screenshot of the Wolf Sheep example MPS description

Then, when the code description is fully completed, the user can generate the ODD code by clicking on "Preview generated text":

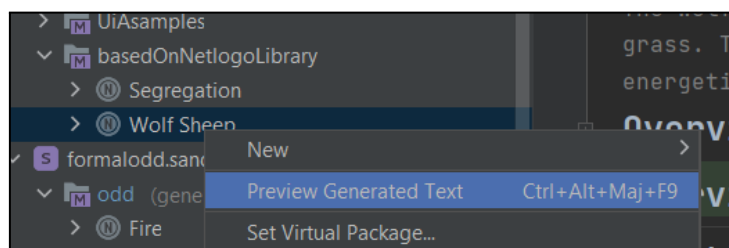


Figure 3: Screenshot of the "Preview Generated Text" feature

This generate a .nlogo file as shown below:

```

1  globals [
2    __INTERNAL__stop ; variable to indicate that there is a stop
3  ]
4
5  breed [ wolfs wolf ] ; wolves
6
7  breed [ sheeps sheep ] ; sheep
8
9  turtles-own [
10   energy ; agents own energy
11 ]
12
13 patches-own [
14   grass-amount ; patches have grass
15 ]
16
17 to setup
18   clear-all
19   set __INTERNAL__stop false
20   ask patches [
21     set grass-amount max (list 0 min (list 10 (precision (random-float (10)) 2)))

```

Figure 4: Screenshot of the .nlogo file generated

Then the user have to open it directly in NetLogo and can access to all the functionalities of the model:

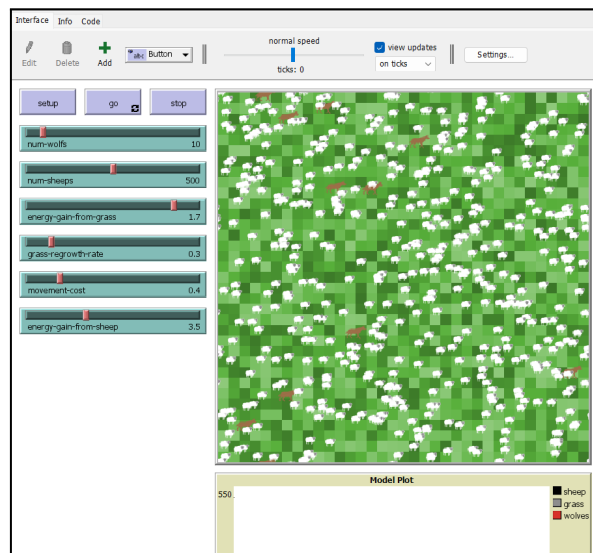


Figure 5: Screenshot of ODD2Netlogo solution

Thanks to the NetLogo application, we can also see the code that has been generated.

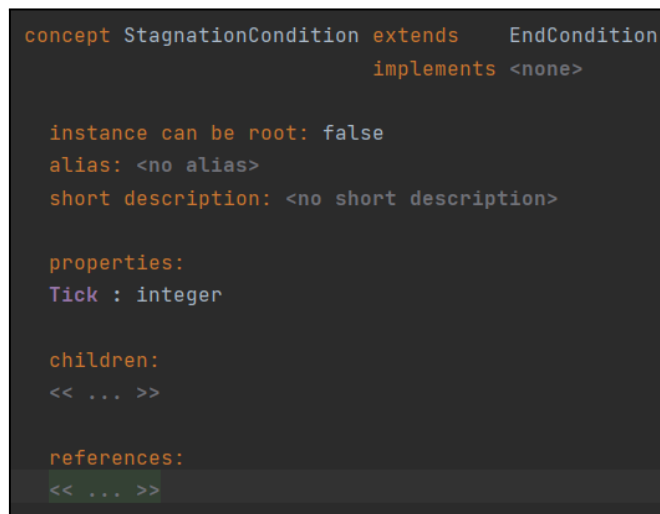
3. Solution

In the purpose of improving the language, we tried to cover all its different aspects: structure, syntax and semantics. This section will describe all of our different implementations for this purpose.

A. Structure

a. Stagnation condition

To improve the structure aspect, we implemented a new concept: StagnationCondition. The purpose of this concept is to stop the running model when no entities or environment entities are changing anymore. This allows us to avoid infinite running models. You can find below a screenshot of the StagnationCondition structure that we have added.



```
concept StagnationCondition extends EndCondition
    implements <none>

    instance can be root: false
    alias: <no alias>
    short description: <no short description>

    properties:
    Tick : integer

    children:
    << ... >>

    references:
    << ... >>
```

Figure 6: Screenshot of the StagnationCondition concept specifications

We can see that StagnationCondition extends EndCondition, according to its feature, and has a property Tick that represents the time that the model is waiting for a movement of an entity before stopping.

b. SelectN

Then we decided to change the SelectN structure by replacing SelectExpression by EntityAccess, which allows us to restrict the user's choices when using selectN. Indeed, after this modification, the user no longer has access to the indices, but can only choose on which entities to perform selectN.


```

instance can be root: false
alias: <no alias>
short description: <no short description>

properties:
  where : Where
  partner : SpecialEntities

children:
  count : Expression[1]
  argument : EntityAccess[1]

```

Figure 7: Screenshot showing the structure modification of the SelectN concept

We have also changed the definition of SelectN properties to remove the default selection in the "where" and "partner" properties. This allows the user, when using selectN, to have a better understanding of the different choices available to him.

```

enumeration Where

members:
  • sameSpot same_spot
  • neighbour neighbour_spot
  • anywhere anywhere (default)

default member: anywhere

```

```

enumeration Where

members:
  • sameSpot same_spot
  • neighbour neighbour_spot
  • anywhere anywhere

default member: null

```

Figure 8 & 9: Screenshots showing the structure modification of the "where" property. on the left, the definition of propriety before it was modified and on the right, the definition after it was modified.

```

enumeration SpecialEntities

members:
  • me first_partner (default)
  • other second_partner

default member: me

```

```

enumeration SpecialEntities

members:
  • me first_partner
  • other second_partner

default member: null

```

Figure 10 & 11: Screenshot showing the structure modification of the "partner" property. on the left, the definition of propriety before it was modified and on the right, the definition after it was modified.

B. Syntax

To improve the syntax aspect, we tried to modify some editor specifications, here are the main trials we did.

a. Stagnation Condition

We've added an editor to the new stagnation concept. In the Fig..12, ticks represents the number of time steps to wait until the entities (and all attributes linked to the entities) stop moving, before the simulation stops.

```
<default> editor for concept StagnationCondition
node cell layout:
[ /
[> No entities and environment entities (and all their properties) have change in { | Tick | } time step(s) |< ]
/ ]
```

Figure 12: Screenshot of the StagnationCondition concept editor specifications

b. Entity Access

For the entity access editor, we've added a comment to make it easier for the user to understand what's expected of him. This new comment is particularly useful when the user defines SelectN.

```
<default> editor for concept EntityAccess
node cell layout:
[ - | % | entity | /empty cell: | press ctrl space to select the entity | % | - ]

inspected cell layout:
<choose cell model>
```

Figure 13: Screenshot showing the editor modification of the "EntityAccess" concept

Now, the user will have the message "press ctrl space to select the entity" when a cell is empty, instead of nothing.

c. Action Call

We've modified the ActionCall editor to allow you to add conditions on neighbors. Indeed, it was not possible to add SelectN after calling an interaction (in the schedule before).

```
<default> editor for concept ActionCall
node cell layout:
[> ?* R/O model access * Perform the ? inter action ( % called % -> { name } ) ? with ? ( < % SelectN % /empty cell: | press ctrl and space if y
inspected cell layout:
<choose cell model>
```

Figure 14: Screenshot showing the editor modification of the ActionCall concept

What's more, we've restricted the appearance of this possibility to the interaction call only. This was made possible by adding a condition to the inspector. You can find the associated screenshot below.

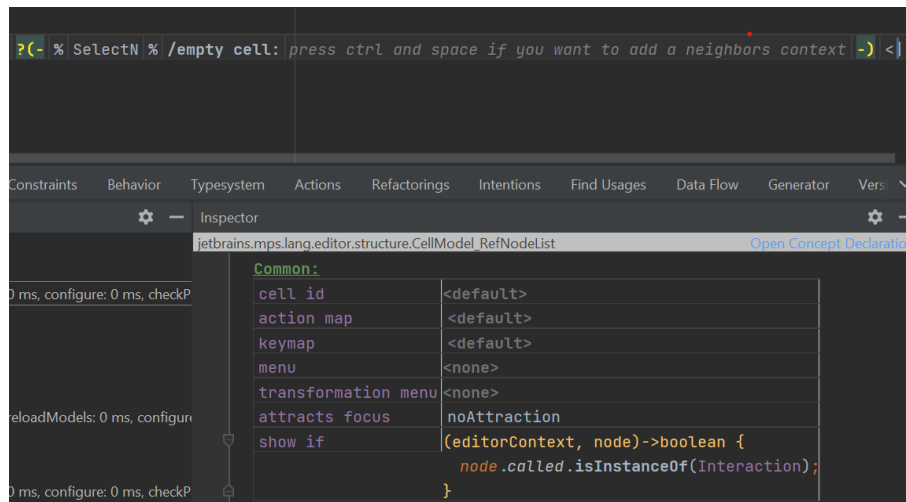


Figure 15: Screenshot showing the editor modification of the ActionCall concept

d. SelectN

The SelectN concept is the concept responsible for detecting the neighbors of a patch. At the beginning, it was not possible to add something else after "Perform the interaction X.". But that was a problem, as both for the Fire and for the Voting example we had to detect the neighbors and we had to copy paste the line from the Wolf Sheep example, which was already working. Then we modified the editor aspect in this direction:

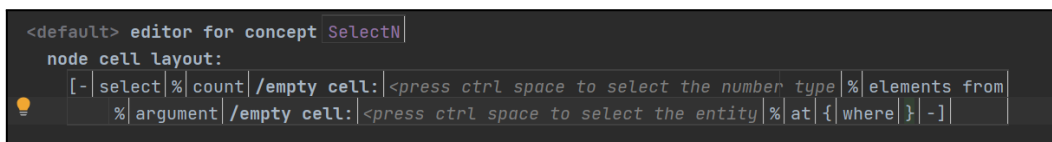


Figure 16: Screenshot of the SelectN editor description

Then, we run to this result:

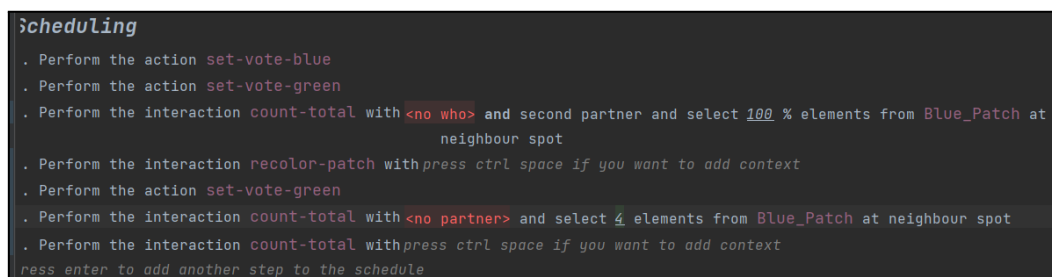


Figure 17: Screenshot of the Scheduling after the modifications in the editor

Now we have access to this feature, and we can perform interactions including neighbors.

e. Experiments

The concept Experiments is a concept that appears in every model creation, it allows to modify the initialization values or the simulation end conditions. You can see an illustration below:

```
Details: Manual Experiments

Initialize environment entities
Environment Blue_Patch is located initially everywhere
Environment Green_Patch is located initially by density percentage 50

Initialize environment attributes
The initial value of attribute Vote is a random value between 0 and 1
The initial value of attribute Total is <no initialisation>

Visualization of Data in Manual Experiments
Press enter to add data for visualization

Appearance
press enter to add an appearance to your simulation

Simulation end
Any of the following conditions end the simulation:
press enter to add end condition
```

Figure 18: Screenshot of the Experiments part in a model creation

In the Appearance section, the message "press enter to add an appearance to your simulation" was not there, and there was an empty field at this place. To have this description, we modified the Experiments concept editor as follow:

```
Initialization editor for concept Experiments
node cell layout:
[ /
  Visualization of Data in Manual Experiments
  ( / %dataPresentation% / )
  /empty cell: Press enter to add data for visualization
  <constant>
  Appearance
  %defaultWorld /empty cell: press enter to add an appearance to your simulation %
/ ]
```

Figure 19: Screenshot showing the modifications on the editor of Experiments concept

And then we have this in the model creation:

```
Appearance
press enter to add an appearance to your simulation
```

Figure 20: Screenshot showing the modification generated in the model creation.

C. Semantics

To improve the semantics aspect, we tried to modify some textgen specifications, here are the main trials we did.

a. selectN

For the semantics part, we mainly tried to improve the text generation for the SelectN concept. The main issue at the beginning was that it was not working with environment entities. To achieve to make it works, we modified the SelectN_TextGen as follow:

```
text gen component for concept SelectN {
  (node)->void {
    string other = "";
    string ColourCheck = "";
    if (node.parent.isInstanceOf(ForEach) && node.parent:ForEach.who.is(other)) {
      other = "other";
    }
    if (node.parent.isInstanceOf(ActionCall)) {
      other = "other";
    }
  }
  if (node.count.isInstanceOf(Percentage)) {
    if (node.argument.isInstanceOf(EntityAccess) && node.argument:EntityAccess.entity.
      isInstanceOf(EnvironmentEntityReference)) {
      ifInstanceOf (node.argument:EntityAccess.entity:EnvironmentEntityReference.envEntity.defaultColour is
        ColourConstant constant) {
        ColourCheck = "pcolor = " + constant.colour.name;
      }
      ifInstanceOf (node.argument:EntityAccess.entity:EnvironmentEntityReference.envEntity.defaultColour is
        ScaledColour constant) {
        ColourCheck = "shade-of? pcolor " + constant.colour.name;
      }
      append {neighbors4} { with patches [ ] } ${ColourCheck} { }];
    } else {
      append {up-to-n-of ((num-} ${node.argument} { * } ${node.count} { } ) / 100} ${other} ${node.argument};
    }
  } else {
```

Figure 21: Screenshot showing the modifications in the SelectN textgen code (1/2)

```
    } else {
      if (node.argument.isInstanceOf(EntityAccess) && node.argument:EntityAccess.entity.
        isInstanceOf(EnvironmentEntityReference)) {
        ifInstanceOf (node.argument:EntityAccess.entity:EnvironmentEntityReference.envEntity.defaultColour is
          ColourConstant constant) {
          ColourCheck = "pcolor = " + constant.colour.name;
        }
        ifInstanceOf (node.argument:EntityAccess.entity:EnvironmentEntityReference.envEntity.defaultColour is
          ScaledColour constant) {
          ColourCheck = "shade-of? pcolor " + constant.colour.name;
        }
        append {neighbors} ${node.count} {with patches [ ] } ${ColourCheck} { }];
      } else {
        append {up-to-n-of } ${node.count} { } } ${other} ${node.argument};
      }
    }
    if (node.where.is(sameSpot)) {
      append {-here};
    }
  }
}
```

Figure 22: Screenshot showing the modifications in the SelectN textgen code (2/2)

b. Action Call

We also modified the text generation for the ActionCall concept, because when an action was called there were some problems with the environment entities. Here you can see the modified textgen code:

```
if (wrap) {
  append indent {ask };
  if (node.called.actor.isInstanceOf(ConcreteEntityReference)) {
    if (node.called.actor:ConcreteEntityReference.generalEntity.isInstanceOf(EnvironmentEntity)) {

      ifInstanceOf (node.called.actor:ConcreteEntityReference.generalEntity:EnvironmentEntity.defaultColour is
        ColourConstant constant) {
        ColourCheck = "pcolor = " + constant.colour.name;
      }
      ifInstanceOf (node.called.actor:ConcreteEntityReference.generalEntity:EnvironmentEntity.defaultColour is
        ScaledColour constant) {
        ColourCheck = "shade-of? pcolor " + constant.colour.name;
      }
      append {patches with [ ] ${ColourCheck} { }};
    } else {
      append ${node.called.actor:ConcreteEntityReference.generalEntity.name} {s};
    }
  } else {
    append ${node.called.actor};
  }
  append { [] \n;
  increase depth;
}
```

Figure 23: Screenshot showing the modifications in the ActionCall textgen code

Here is a before-after modification in the generated code:

```
ask patches [
  Extinguishing self
]
```

Figure 24: Figure showing what was generated before modifications

```
ask patches with [ pcolor = red ] [
  Extinguish self
]
```

Figure 25: Figure showing what was generated after modifications

c. Timed Condition and Quantified condition

The third example of what we have done to improve the transformation aspect is to change the end of the simulation condition because in some cases it generates an extra word which is not useful.

In fact, on this screenshot you can see that the word calculate has been added, so that's what we wanted to change. This is the problem we have tried to solve.

```
if all? patches [pcolor != red] [calculatate stop];
```

Figure 26: Figure showing what was generated before modifications

After some analysis we found that this word is generated when the user uses the "TimedCondition" concept and in some cases the "QuantifiedCondition" concept. So in the textgen we've removed the extra word.

```
text gen component for concept TimedCondition {  
  (node)->void {  
    append indent {if ticks } ${node.operator.presentation} { } ${node.value} { [ stop ] ; timed end of simulation} \n;  
  }  
}
```

Figure 27: Screenshot showing the modifications in the TimedCondition textgen code

```
if (node.quantifier.is(NONE)) {  
  append indent {if not any? } ${node.entity.name} {s [ stop ]} \n;  
}  
if (node.quantifier.is(ALL)) {  
  append indent {if count turtles = count} ${node.entity.name} {s [ stop ]} \n;  
}
```

Figure 28: Screenshot showing the modifications in the QuantifiedCondition textgen code.

4. Discussion

A. implementation

Our implementation covered the language in all aspects:

- Structure, with an additional feature forbidding having infinite running NetLogo programs, and with a concept correction.
- Editor, with much additional information helping the lambda user of ODD2NetLogo to know what he has to do.
- Constraints, with an improvement of selectN editor (when to show or not this concept)
- Transformations, with an improvement of the SelectN textgen in the case of environment entities, and with an improvement of the ActionCall textgen .

B. An improvement who works well

One of the things we've improved that works well is the way we call interactions when we have environment entities. In fact, we've changed the Action Call textgen to add a section on how code is generated for environment entities.

Here you can find a screenshot of one of our models where we have defined an "Extinguish" action to be applied only to the fire entity (which is red in color).

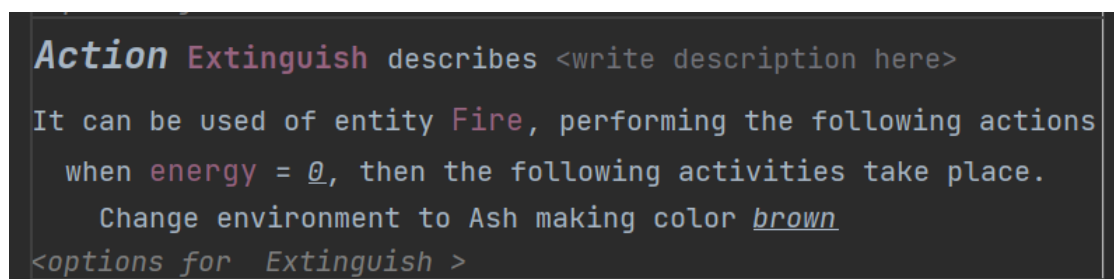


Figure 29: Screenshot showing the action extinguishing in the user interface.

And when we look at the code generated by ODD, we can see that our extinguish action is called with the condition that it only applies to the red entity, which is fire.

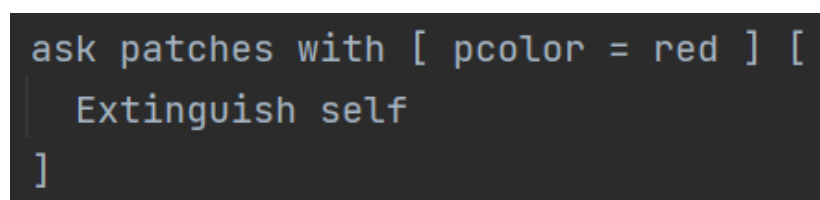


Figure 30: Screenshot showing a part of the generate code (when extinguish is called)

C. Future step and alternatives solution

a. SelectN

For the SelectN editor modifications, we achieved to unlock the feature of performing interactions with neighbors as shown previously, but the new implementation need some more update.

Select "selectN" as a user

When we want to select "selectN" in a model, the user must type ctrl and space after the "with". However, the user is offered choices he shouldn't have, so we tried to solve this problem by changing the structure and editor of the "ActionCall" concept. As you can see below we have changed one of the children of ActionCall by selectN.

```
concept ActionCall extends Activity
    implements <none>

    instance can be root: false
    alias: <no alias>
    short description: calls a procedure or interaction

    properties:
    << ... >>

    children:
    actuals : Actual[0..n]
    SelectN : SelectN[0..n]
```

Figure 31: Screenshot showing the modifications in the ActionCall structure.

```
h ?(-)%|SelectN%|/empty cell:|press ctrl and space if you want to add a neighbors context
```

Figure 32: Screenshot showing the modifications in the ActionCall editor.

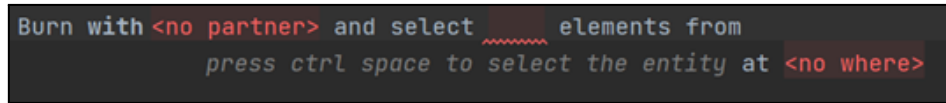
However, when we try to check in the user interface whether this modification has worked, we notice that something strange is happening. In fact, when the user types control and space, he's presented with a panoply of possible choices.

```
Burn with press ctrl and space if you want to add a neighbors context
Burn by fir
and a Tree
here>
```

Burn	^codes (odd
Decrease	^codes (odd
Dummy	^codes (odd.Forest fire with
Extinguish	^codes (odd
Find_new_spot	^codes (odd.Segregation
check-if-dead	^codes (odd.Wolf Sheep
eat-grass	^codes (odd.Wolf Sheep
eat-sheep	^codes (odd.Wolf Sheep
initiation	^codes (odd

Figure 33: Screenshot showing the user interface choices

Then, no matter which one the user chooses, only the concept "selectN" will be displayed, as you can see in the screenshot below.



```
Burn with <no partner> and select _____ elements from
press ctrl space to select the entity at <no where>
```

Figure 34: Screenshot showing the user interface selectN

Substitute menu

Again in our "selectN" concept, we've given the user the option of manually entering the number of neighbors on which the interaction is to be performed. However, if the user chooses the numerical constant type, then the number four should be entered automatically (as it is the only one recognized by netlogo).

To do this, we have added a SelectN substitute menu, in order to, when we select "numerical constant" in the menu, we have "4" by default. You can find it in the screenshot below:



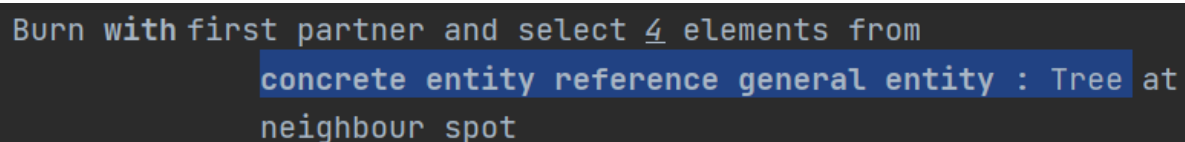
```
substitute menu for concept SelectN : default
|
reference actions called(output concept: default)
  matching text <default>
  visible matching text <empty prefix><ref. presentation><empty suffix>
  description text (parentNode, currentTargetNode, link, editorContext, model, pattern, referencedNode
    if (currentTargetNode.count.isInstanceOf(NumberConstant)) {
      return "4";
    } else {
      return "";
    }
  }
```

Figure 35: Screenshot showing the substitute menu in SelectN concept.

Unfortunately, this implementation did not work. When we tried to build, nothing changed.

Textgen of SelectN: Entity type

Another trial was about the generation of code in the selectN which was not accurate:



```
Burn with first partner and select 4 elements from
concrete entity reference general entity : Tree at
neighbour spot
```

Figure 36: Screenshot showing the reflective editor of Tree in the user interface

Then, we tried to use ConcreteEntityReference in the TextGen of SelectN:

```

if (node.count.isInstanceOf(Percentage)) {
  if (node.called.actor.isInstanceOf(ConcreteEntityReference) &&
      node.called.actor:ConcreteEntityReference.generalEntity.isInstanceOf(EnvironmentEntity)) {
    ifInstanceOf (node.called.actor:ConcreteEntityReference.generalEntity:EnvironmentEntity.defaultColour is
        ColourConstant constant) {
      colourCheck = "pcolor = " + constant.colour.name;
    }
    ifInstanceOf (node.called.actor:ConcreteEntityReference.generalEntity:EnvironmentEntity.defaultColour is
        ScaledColour constant) {
      colourCheck = "shade-of? pcolor " + constant.colour.name;
    }
    append {neighbors4} { with patches [ ] ${colourCheck} { } };
  } else {
    append {up-to-n-of ((num-) ${node.entiy} { * } ${node.count} { } / 100) ${other} ${node.entiy};
  }
}

```

Figure 37: Screenshot showing a modification of the TextGen of SelectN (1/2)

```

if (node.called.actor.isInstanceOf(ConcreteEntityReference) &&
    node.called.actor:ConcreteEntityReference.generalEntity.isInstanceOf(EnvironmentEntity)) {
  ifInstanceOf (node.called.actor:ConcreteEntityReference.generalEntity:EnvironmentEntity.defaultColour is
      ColourConstant constant) {
    colourCheck = "pcolor = " + constant.colour.name;
  }
  ifInstanceOf (node.called.actor:ConcreteEntityReference.generalEntity:EnvironmentEntity.defaultColour is
      ScaledColour constant) {
    colourCheck = "shade-of? pcolor " + constant.colour.name;
  }
  append {neighbors4} { with patches [ ] ${colourCheck} { } };
} else {
  append {up-to-n-of } ${node.count} { } ${other} ${node.entiy};
}

if (node.where.is(sameSpot)) {
  append {-here};
}

```

Figure 38: Screenshot showing a modification of the TextGen of SelectN (2/2)

Unfortunately, this modification to the concept selectN Textgen didn't work. Code generation is impaired when this code is used.

b. EnvironmentEntity

We also wanted to add a constraint to only have the possibility to choose between the protagonists of an interaction when we select the elements at neighbor spots, like in the screenshot below:

```

. Perform the interaction count-total with <no who> and second partner and select 100 % elements from Blue_Patch at
  neighbour spot

```

Figure 39: Screenshot showing the interaction description in the model

We do not really know how to do it, but that can be an interesting feature to add in the future.

5. Plan and Reflection

First of all, regarding the management and planning of this project, we started working on it after we had chosen our subject, at the beginning of the semester. We worked regularly throughout the semester, right up to the deadline. From an organizational point of view, each member of the group decided to work independently. However, we sometimes met up on campus to work together when we encountered problems or when we were faced with uncertainties or misunderstandings. Alix worked deeper than the others on the project, and we agreed on a 40% (Alix) - 20%-20%-20% (Raphaël, Nasr, Louis) repartition of the workload (where the average workload is 25% for 4 people).

In addition, when we encountered difficulties, we contacted our supervisors and quickly organized meetings, both in person and remotely using the Teams video conferencing platform. The aim of these meetings was to highlight our progress on the various tasks and to get answers to our questions. In parallel, we used a Discord server where we discussed the project, the various tasks to be carried out, our problems and the solutions we had implemented.

6. Questions for the MPS Learning Quizz

In this part of the report, each member of the group proposes one question that could be used for the MPS learning quiz, such that the students next year avoid our problems. These questions are related to the MPS tool and are mentioned below (the correct answer is in bold):

- 1) For generation languages, how do we generate the target code?
 - **We have to click on the model in the logical view and select "Preview generated code"**
 - We have to copy paste the generation code into the code section of the target language editor
 - We have to download the model as an MPS file and load it into the target language editor
 - None of the above
- 2) How to find the different utilizations of a concept?
 - Right click on the concept on the logical view menu and select "Find utilizations"
 - **Right click on the concept on the logical view menu and select "Find usages"**
 - Right click on the concept on the logical view menu and select "Show Node in Explorer"
 - None of the above
- 3) When you modify the editor section of a concept, how to add a "show if" condition?
 - Right click on the area we want the show if and select "Show Node in Explorer"
 - Type Ctrl and space and select "show if" directly in the editor
 - **Right click on the area we want the show if and select "Inspect Node"**
 - None of the above.
- 4) When you modify the textgen section of a concept, how can you see which concepts descend from an attribute you're using ?
 - **Right-click on the attribute in the structure, then click on "show concept in hierarchy"**
 - Move your cursor over the target in question. Indications of its parents appear
 - There's no simple method: you have to look directly in the tree and find where the attribute comes from.
 - None of the above

7. References

- **[1]** The ODD Protocol for Describing Agent-Based and Other Simulation Models: A Second Update to Improve Clarity, Replication, and Structural Realism by Volker Grimm and other <https://www.jasss.org/23/2/7.html>
- **[2]** From Comses Net <https://www.comses.net/resources/standards/>