Title: Simulating crop root systems using *OpenSimRoot*

Running Head: *OpenSimRoot*

Authors: Ernst D Schäfer[1], Markus R Owen[1], Johannes Postma,[2] Christian Kuppe[2], Christopher K Black[3], Jonathan P Lynch[3]

[1] University of Nottingham, University Park Campus, Nottingham, NG7 2RD, United Kingdom

[2] Forschungszentrum Jülich, GmbH, 52425 Jülich, Germany

[3] Pennsylvania State University, 201 Old Main, University Park, Pennsylvania 16802, United States of America

Corresponding author email address: schaferscientificmodelling@protonmail.com

Summary:

Functional-structural plant models are valuable modelling tools in analysing plant development. A functional-structural plant model combines a three-dimensional representation of plant structure with models for physiological functions in order to better understand plant development. We present a guide to simulating crop root systems with *OpenSimRoot*, a feature-rich, highly cited and open source functional-structural root architecture model. We describe in detail how to create your own input files in conjunction with some examples. The aim of this guide is to highlight the potential of computational modelling in biology and to make modelling more accessible to the plant science community.

Keywords:

plant development; root system architecture; mathematical modelling; computational modelling; organ growth; nutrient uptake; functional structural plant modelling;

1. Introduction

Functional-structural root system architecture models combine a three-dimensional representation of root systems with models of physiological functions and are a key tool in the study of root systems (1). Instead of focusing on the effects of single genes at a cellular or even plant organ scale, root system architecture models help researchers gain insight in the effects of phenes on overall root system and crop development. The earliest versions of these models represented pre-defined root architectures, but after integrating new insights in root responses to local soil conditions and plant signalling mechanisms, they are presently able to simulate root growth in response to dynamic and heterogeneous environments and incorporate feedbacks that regulate root and shoot development (2, 3, 4, 5, 6, 7, 8, 9).

Functional-structural root system architecture models are often used to generate hypotheses about the effects of root architectural traits on root distribution in the soil and on the ability of root systems to extract resources from dynamic soil environments (10, 11). They can also provide insights into the viability of different foraging strategies in scenarios where there is competition between different plants or there are multiple resource constraints (12, 13). Another useful feature of models is that they allow researchers to vary variables that are very difficult or even outright impossible to control in field conditions and evaluate the utility of traits in these scenarios. Due to advances in computing, simulating root systems takes a smaller and smaller fraction of the time it takes to grow a similar root system in the field. This means that simulation models can explore parameter spaces much faster than would ever be possible through field experiments, even more so because researchers using simulation models have perfect control over external factors such as atmospheric and soil conditions, which is not the case for field studies.

Experimental biologists increasingly depend on theoreticians to build models and use them to test hypotheses. In order to make models more accessible to experimentalists, as well as facilitate the sharing of expertise across different modelling teams, recent years have seen efforts to make source code of published models available, i.e. open source (14). This allows researchers to learn from each

other's expertise, to better assess the relevant modelling assumptions and to correct errors, compare results (15) and share ideas.

OpenSimRoot allows researchers to simulate root systems based on growth, branching and functioning rules for different root classes. It can model photosynthesis, carbon allocation, growth responses to carbon shortages, nutrient and water flows through the soil, nutrient and water uptake by roots, nutrient stress responses, root hairs, root cortical aerenchyma formation, root cortical senescence, plant respiration, root exudation, root responses to local nutrient conditions and many other processes that are relevant in plant development. Although a general description of *OpenSimRoot* has been published (14), details on the implementation of the various models are encapsulated in individual modules and can currently only be understood by studying the source code. Rather than providing detailed descriptions of every single module, this document aims to provide readers with a comprehensive overview of all the information needed to create and run *OpenSimRoot* models for their own scientific purposes. It is also meant as an introduction for aspiring *OpenSimRoot* developers who wish to add new models and functionality to answer questions it was previously not possible to address. Finally, it explains how to use the graphical user interface (GUI) that was created to make creating new *OpenSimRoot* models easier and more convenient.

The code of *OpenSimRoot* is modular, and allows different models to be constructed based on an XML metalanguage which describes what modules will be used, i.e. what processes will be simulated. The input files thereby are significantly more complicated than a simple parameter list. To define a model in *OpenSimRoot*, the user must create an input file that specifies which sub-models to use, specifies which root classes are being simulated, defines the branching and growth rules and contains all needed parameters. Because users can define new models without making changes to the source code, *OpenSimRoot* is not a model, but a flexible modelling and simulation tool. Section 2 details how to prepare your system for running *OpenSimRoot*. Section 3 explains how to run *OpenSimRoot* and provides detailed instructions for all the steps needed to create an input file, and Section 4 includes supplementary notes.

2. Materials

You can run *OpenSimRoot* by downloading an executable from the website. Executables for Windows and Linux are provided. Many users may instead want to download the code and compile it themselves, so we provide here a short instruction. *OpenSimRoot* input files use the XML standard, which will be explained briefly in this document. Various tools are available to make creating input files easier, as well as a graphical user interface (GUI) that allows the creation of input files from scratch, see section 3.8. These tools require Python to be installed and can be downloaded from a public repository. Open source C++ compilers and Python are available for free, as is Git, which we use and recommend as a version control program.

2.1. Required software: C++ compiler

To compile the code the user needs a C++ compiler and probably software that can manage the compilation processes. Makefiles are currently provided which can be used with the make command, but loading the code in a correctly setup IDE such as Visual Studio or Eclipse CDT also works. The code compiles with GCC, MINGW, CLANG and MSVC (Visual Studio), provided they are current and support the C++14 standard. Windows users may download and install a C++ compiler, for example from https://gcc.gnu.org/ or https://visualstudio.microsoft.com/. Linux and Mac users should find that a C++ compiler is already included with their operating system or may install them through their application repositories. In recent Windows versions, the Windows-Subsystems for Linux (WSL) can be enabled, and the Microsoft application store provides Linux distributions including the ability to use the distribution provided compilers.

2.2. Source code

Download the *OpenSimRoot* source code from https://gitlab.com/rootmodels/OpenSimRoot/-/archive/master/OpenSimRoot-master.zip and unpack it in the directory where you wish to install *OpenSimRoot*.

Users familiar with Git should use it to get the latest *OpenSimRoot* code updates. After creating an account on https://gitlab.com/ and setting up an SSH key, the code can be cloned using **git clone git@gitlab.com:rootmodels/OpenSimRoot.git**.

2.3. Compile *OpenSimRoot*

To compile *OpenSimRoot*, follow these steps:

1. Open a terminal/command window in the *OpenSimRoot* directory. On Windows this is done by typing **cmd** in the start menu and then using the **cd** command to navigate to the *OpenSimRoot* directory, or by holding shift and right clicking in the *OpenSimRoot* directory and selecting "**Open command window here**" from the context menu. On Linux, right click in the *OpenSimRoot* directory and select "**Open terminal here**" from the context menu.
2. Use **make all** in the **OpenSimRoot/StaticBuild** or **OpenSimRoot/StaticBuild_win64** directory. Alternatively, execute the command **bash build.sh** (in the top-level directory). See note 1. Note that **build_win64.sh** is not provided for Windows users, but for those that want to build Windows executables on Linux systems using the MinGW GCC compiler.
3. To clean up temporary files use **cleanup.sh**.

2.4. Test *OpenSimRoot*

In the *OpenSimRoot* directory, either in a command window (cmd, or better powershell in Windows) or terminal (Linux/macOS), run **OpenSimRoot/StaticBuild/OpenSimRoot** (Linux/macOS) or **OpenSimRoot\StaticBuild_win64\OpenSimRoot.exe -h** (Windows), you should see the following output:

```
E:\Werk\OpenSimRoot>OpenSimRoot\StaticBuild_win64\OpenSimRoot.exe -h
Usage:  OpenSimRoot   [OPTIONS]      [FILE]
OpenSimRoot simulates a model defined in FILE


 -f, --file                 specify simulation file, default last argument
 -h, --help or /?           print this help message
```

```
 -v                    be verbose with warnings
 -q                    be quite with warnings
 -l, --list            print list of registered functions
 -V, --verify          Experimental function for verifying input files
 -ww                   warnings to screen

Examples:
 OpenSimRoot -h              Prints this message
 OpenSimRoot runModel.xml    Runs the SimRoot model

Support at j.postma@fz-juelich.de
Built on Jul 17 2019, git 2ff025e112
Licensed to you under the GPLv3 license.

There are no warnings.
Simulation took (hours:minutes:seconds): 0:0:0
```

Now use **bash runTests.sh**. This will run a number of test simulations to ensure various parts of the code are functioning correctly. One of the tests depends on the programming language R being installed. If R is not installed on your system, use **bash runTestsEngine.sh** and **bash runTestsModule.sh** to run all the tests that do not depend on R. If the tests complete correctly, finished testing engine with error status 0, finished testing modules with error status 0 and exiting with error status 0 will be printed to screen.

3. Methods

OpenSimRoot implements different sub-models for processes that are required to model root system development and functioning. They include models for root growth and branching, root nutrient

uptake, soil water and nutrient transport, photosynthesis, nutrient stress and many others. The model is defined by the input file, which specifies which models are included, what root classes are simulated, which branching rules are used, as well as plant and environmental parameters and simulation settings. We first describe how to run *OpenSimRoot*, assuming you already have an input file available. We then describe all the operations needed to create any input file you might need in detail. First, we describe how to add or remove models, then we describe how to change parameters, then we describe how to create a new model and finally we explain how to add or remove root classes.

3.1. Running *OpenSimRoot*

Windows: Create a folder where you wish the output files to be saved. Open a command window there and use **C:\OpenSimRoot\OpenSimRoot\StaticBuild_win64\OpenSimRoot.exe C:\InputFiles\inputFile.xml** where "**InputFiles**" is the directory where your input file is located and "**inputFile.xml**" is the input file you wish to run. Note that we assumed copy of *OpenSimRoot* is located in **C:**, if this is not the case replace this by the appropriate directory. Alternatively, double-click **OpenSimRoot.exe**, located in **C:\OpenSimRoot\OpenSimRoot\StaticBuild_win64**. This will open a file selection browser. Use this to select the input file you want to run. The output files will be created in the same directory as the input file.

Linux/macOS: Create an empty directory where you wish the output files to be saved. Open a terminal there and type **~/OpenSimRoot/OpenSimRoot/StaticBuild/OpenSimRoot ~/InputFiles/inputFile.xml**, where "**InputFiles**" is the directory where your input file is located and "**inputFile.xml**" is the input file you wish to run. Note that we assumed copy of *OpenSimRoot* is located in your home directory, if this is not the case insert the appropriate path in the command above.

If there is any problem with the input file, the simulation will stop and display an error message, see note 2 for common errors and solutions.

OpenSimRoot can generate different types of output, depending on what is specified in the input files. The **tabled_output.tab** is a tab-separated text file that contains the values of plant-scale state variables

at regular intervals. It lists the plant dry weight, total root length, total nutrient uptake, etc. We recommend you always let *OpenSimRoot* generate this file. The file contains 6 columns, which contain the following:

- The first column contains the name of the variable.
- The second column contains the time.
- The third column contains the value of the variable at the given time.
- The fourth column contains the rate at which the variable changes, if applicable.
- The fifth column contains the unit of the variable.
- The sixth column contains the path of the variable. This is used to distinguish different variables that have the same name, such as the total uptakes of different nutrients.

See note 3 for some sample tabled output. The tabled output can be imported into Excel or a similar program for processing and creating figures. R users can use the **read.table(filename, header=T)** command. If R is installed, a quick plot can be created by combining the **fgrep** command with the script named **plot** in the directory **OpenSimRoot/scripts**. For example **cat tabled_output.tab | grep -f 'plantDryWeight' | plot** will plot the plant dry weight against the time.

OpenSimRoot can also save the geometry of the root system at specified intervals, including the values of root segment specific state variables. It can also save three-dimensional information about the soil. The VTU and VTP formats are available for the root system geometry, while VTU is used for the soil. Programs like ParaView can be used to visualise this data in 3D and see the distribution of various state variables across the root system and render videos of the growing root system. See figure 1 for a visualisation of a simulated maize root system created with ParaView.

3.2. Creating input files

Included with the *OpenSimRoot* code are models for barley, bean, maize and squash. Simulating these species with slightly different parameters is straightforward. They are a good point of departure for creating new input files for other species. Note that parametrisation of a new species requires a significant amount of work, given the many parameters used. *OpenSimRoot* input files are written in

the XML markup language format; see note 4 for an explanation of the structure of XML files. *OpenSimRoot* looks for certain types of tags in input files; see note 5 for an overview of all the tag types and their usage.

Section 3.8 explains how to download and run the graphical user interface (GUI) that was created to make creating *OpenSimRoot* models easier. It provides a convenient tool that automates the more tedious document editing operations described in the preceding sections. While it is possible to create models using section 3.8 and the GUI, we recommend that you do read the preceding sections first in order to gain a better understanding of the structure of *OpenSimRoot* input files.

3.3. Activating or deactivating modules

OpenSimRoot contains a lot of different models that simulate different processes, not all of which are required for every simulation study. A set of models that simulate a certain feature is referred to as a module. See note 6 for an overview of the different modules. Every module has at least one associated template file which can be found in the directory **OpenSimRoot/inputFiles/templates**. Including a template file in the input file through reference (or copying the contents) means that the associated module is included in the simulation. For example, to include root hairs in the simulation, add the line

```
<SimulaIncludeFile fileName="templates/plantTemplate.IncludeRootHairs.xml"/>
```

to the input file. Removing a module is done by deleting the reference to the associated template(s). Of course, different modules require different parameters. However, since including unused parameters does not alter the simulation behaviour, all parameters are included by default, and adding or removing modules can be done by adding or removing tags that refer to the relevant template. To see all available templates, please see the **OpenSimRoot/inputFiles/templates** directory.

3.4. Editing parameters

To edit parameters, we follow these steps.

- It is probably wise to make a local copy of the **InputFiles** directory, or use a version control tool to track your changes. With a tool like diff or Meld you can display changes.

- We first decide in which file(s) we want to change parameters. Let's assume that we want to change a parameter in the maize model.
- Open the file **runMaize.xml**. It should look something like this (comments removed for the sake of readability):

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<?xml-stylesheet type="text/xsl" href="XML/treeview.xsl"?>
<SimulationModel xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="../scripts/XML/SimulaXMLSchema.xsd">
   <SimulaBase name="soil"></SimulaBase>
   <SimulaBase name="plants">
      <SimulaBase name="maize" objectGenerator="seedling">
         <SimulaConstant name="plantType" type="string">maize-aerenchyma</SimulaConstant>
         <SimulaConstant name="plantingTime"  unit="day" type="time">0</SimulaConstant>
         <SimulaConstant name="plantPosition" type="coordinate">0 -2 0</SimulaConstant>
      </SimulaBase>
   </SimulaBase>
   <SimulaIncludeFile fileName="plantParameters/Maize/Maize/simulationControlParameters.xml"/>
   <SimulaIncludeFile fileName="templates/plantTemplateFullModel.xml"/>
   <SimulaIncludeFile fileName="environments/WageningseBovenBuurt/WageningseBovenBuurt-
maize.xml"/>
   <SimulaIncludeFile fileName="templates/plantTemplate.IncludeGeometry.xml" />
   <SimulaIncludeFile fileName="templates/plantTemplate.IncludeDryweight.xml" />
   <SimulaIncludeFile fileName="templates/configurationCarbon.xml"/>
   <SimulaIncludeFile fileName="templates/plantTemplate.IncludeStress.xml"/>
   <SimulaIncludeFile fileName="templates/plantTemplate.IncludeRootHairs.xml"/>
   <SimulaIncludeFile fileName="templates/plantTemplate.IncludeAerenchyma.xml"/>
   <SimulaIncludeFile fileName="templates/plantTemplate.IncludeRootLengthProfile.xml"/>
```

```
  <SimulaBase name="rootTypeParameters" >
    <SimulaIncludeFile fileName="plantParameters/Maize/H99/maize.xml"/>
    <SimulaIncludeFile fileName="plantParameters/Maize/W64a/maize.xml"/>
    <SimulaIncludeFile fileName="plantParameters/Maize/36H56/maize.xml"/>
    <SimulaIncludeFile fileName="plantParameters/Maize/Maize/maize.xml"/>
  </SimulaBase>
</SimulationModel>
```

- Since parameters are spread over different files, we need to locate the parameter that we want to change inside the correct file.
- We note down the plant type, simulation settings file and environment parameter file. For this file, these are **maize-aerenchyma** (line 7), **plantParameters/Maize/Maize/simulationControl-Parameters.xml** (line 12) and **environments/WageningseBovenBuurt/WageningseBovenBuurtmaize.xml** (line 14).
- If we want to change a plant parameter, we have to find the file that contains the parameters for plants of the type **maize-aerenchyma** in the **plantParameters** directory. In this case, it is the file named **plantParameters/Maize/Maize/maize.xml**. The parameter we want to change is either in this file, or one of the files referenced at the bottom of this file. The references will look like this:

```
<SimulaIncludeFile fileName="plantParameters/Maize/Maize/nitrate.xml"/>
```

  For example, parameters related to root hairs will be in the file named **plantParameters/Maize/-Variant/rootHairs.xml** that is referenced at the bottom of the file **plantParameters/Maize/-Maize/maize.xml**.

- If we want to change one of the simulation settings, such as which outputs are generated or the simulation time, we look for the relevant parameter in **plantParameters/Maize/Maize/simulationControlParameters.xml**.

- If we want to change an environment parameter, it will either be in **environments/WageningseBovenBuurt/WageningseBovenBuurt-maize.xml** or one of the files referenced in this file.
- Once we have located the parameter, we alter it. The exact change needed depends on the type of parameter. For parameters of the **SimulaConstant** type, which look like this:

```
<SimulaConstant name="density" unit="g/cm3">0.094</SimulaConstant>
```

we change the value in between the opening and closing tag (0.094) to the desired value.

- Parameters of the type **SimulaTable**, which look like this:

```
<SimulaTable name_column1="time since creation"
unit_column1="day"name_column2="growthRate" unit_column2="cm/day">0 4.5 28 4.5 38 0. 1000
0.</SimulaTable>
```

are used for parameters whose value depends on something else like time or depth, so they are written as pairs of values. In the example given, the value of the parameter is equal to 4.5 for the first 28 days, then it linearly decreases to 0 over the next 10 days and then it remains constant at zero. This is used for parameters whose value depends on something else like time or depth. Extra whitespace or newline characters are ignored so the above is equivalent to this:

```
<SimulaTable name_column1="time since creation"
unit_column1="day"name_column2="growthRate" unit_column2="cm/day">
0 4.5
28 4.5
38 0.
1000 0.
</SimulaTable>
```

- For parameters of the type **SimulaStochastic**, which look like this:

```
<SimulaStochastic name="branchingFrequency" unit="cm" distribution="uniform" minimum="0.05"
maximum="0.25" />
```

we have to provide a distribution and associated values, such as the minimum, maximum, mean and standard deviation of the distribution.

- After entering the parameter, save the file.

3.5. Advanced editing of parameters

In many cases parameter types are interchangeable, meaning that when a parameter is declared as a **SimulaConstant**, it may be replaced by a **SimulaTable** or **SimulaStochastic** (or even by a function, which we do not describe here). In rare cases when the math in the code demands it to be a constant, you will be given an error message when running the model.

One can add random noise to parameters or state variables by adding a **SimulaStochastic** tag with the name **multiplier** as a subtag to the tag in question.

```
<SimulaTable name_column1="time since creation" unit_column1="day"
name_column2="growthRate" unit_column2="cm/day">0 4.5 28 4.5 38 0. 1000 0.
  <SimulaStochastic name="multiplier" distribution="normal" mean="0.2" stddev="0.1" />
</SimulaTable>
```

3.6. Creating new models for different genotypes, species and environments

Now that we have shown how to change parameters, we will show how to change which files are referenced and then we will show how to add and remove root classes. These operations are essential for creating the input files for a new model, for example when we want to simulate a new species. Let's assume we have discovered a new crop species named *unmaize* that we want to create a model for.

- Open the **OpenSimRoot/InputFiles** directory.

- Make a copy of the file **runMaize.xml** and rename it **runUnmaize.xml**. Open the latter file in your favourite text editor.
- We already saw this file in the previous section. First, we change the name. In line 7, change **maize-aerenchyma** to **unmaize**. This will let *OpenSimRoot* know that the plant we are simulating is *unmaize*, not maize.
- Right now, the file includes the environment parameters of the Wageningse bovenbuurt, which is not what we want. Change line 14 to

```
<SimulaIncludeFile fileName="environments/unmaize/unmaize.xml"/>
```

- Next, decide on which modules you wish to include in the simulation. We will keep them largely as is, but add phosphorus to the simulation. Insert the following above line 17:

```
<SimulaIncludeFile fileName="templates/plantTemplate.IncludePhosphorusBC.xml" />
```

This tells *OpenSimRoot* to load the phosphorus template, so phosphorus uptake is simulated using the Barber-Cushman model (BC).

- We are not interested in aerenchyma or the root length profile, so delete the lines referencing their templates, lines 20 and 21 in the listed file above.
- We also do not need the parameters for all the maize variants, so delete lines 23-26 and insert the following in their place:

```
<SimulaIncludeFile fileName="plantParameters/Unmaize/Unmaize/unmaize.xml"/>
```

- Your file should now look like this

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<?xml-stylesheet type="text/xsl" href="XML/treeview.xsl"?>
<SimulationModel xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="../scripts/XML/SimulaXMLSchema.xsd">
```

```
  <SimulaBase name="soil"></SimulaBase>
  <SimulaBase name="plants">
    <SimulaBase name="maize" objectGenerator="seedling">
      <SimulaConstant name="plantType" type="string">unmaize</SimulaConstant>
      <SimulaConstant name="plantingTime"  unit="day" type="time">0</SimulaConstant>
      <SimulaConstant name="plantPosition" type="coordinate">0 -2 0</SimulaConstant>
    </SimulaBase>
  </SimulaBase>
  <SimulaIncludeFile
fileName="plantParameters/Unmaize/Unmaize/simulationControlParameters.xml"/>
  <SimulaIncludeFile fileName="templates/plantTemplateFullModel.xml"/>
<SimulaIncludeFile fileName="environments/Unmaize/unmaize.xml"/>
  <SimulaIncludeFile fileName="templates/plantTemplate.IncludeGeometry.xml" />
  <SimulaIncludeFile fileName="templates/plantTemplate.IncludeDryweight.xml" />
  <SimulaIncludeFile fileName="templates/plantTemplate.IncludePhosphorusBC.xml" />
  <SimulaIncludeFile fileName="templates/configurationCarbon.xml"/>
  <SimulaIncludeFile fileName="templates/plantTemplate.IncludeStress.xml"/>
  <SimulaIncludeFile fileName="templates/plantTemplate.IncludeRootHairs.xml"/>
  <SimulaBase name="rootTypeParameters" >
    <SimulaIncludeFile fileName="plantParameters/Unmaize/Unmaize/unmaize.xml"/>
  </SimulaBase>
</SimulationModel>
```

- Now we need to make sure that these new files we are referencing actually exist. Open the environments directory and copy the folder **WageningseBovenBuurt**, renaming it **Unmaize**. Then open it. Change the name of the **WageningseBovenBuurt-maize.xml** file to

unmaize.xml. Then open it. It should look something like this, after deleting all commented-out lines:

```
<?xml version="1.0" encoding="UTF-8"?>
<SimulationModelIncludeFile xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="../../../scripts/XML/SimulaXMLSchema.xsd">
   <SimulaBase name="environment">
      <SimulaBase name="dimensions">
         <SimulaConstant name="minCorner" type="coordinate">-13 -150 -30 </SimulaConstant>
         <SimulaConstant name="maxCorner" type="coordinate">13 0 30 </SimulaConstant>
         <SimulaConstant name="resolution" type="coordinate">1 1 1 </SimulaConstant>
      </SimulaBase>
      <SimulaBase name="soil">
         <SimulaTable name_column1="depth" unit_column1="cm" name_column2="bulkDensity"
unit_column2="g/cm3">0 1.24 -5 1.24 -16 1.29 -30 1.42 -47 1.40 -65 1.51 -200 1.51 </SimulaTable>
      </SimulaBase>
   </SimulaBase>
   <SimulaIncludeFile fileName="environments/WageningseBovenBuurt/water.xml" />
   <SimulaIncludeFile fileName="environments/WageningseBovenBuurt/atmosphere.xml" />
   <SimulaIncludeFile fileName="environments/WageningseBovenBuurt/nitrate.xml" />
   <SimulaIncludeFile fileName="environments/WageningseBovenBuurt/phosphorus.xml" />
   <SimulaIncludeFile fileName="environments/WageningseBovenBuurt/potassium.xml" />
   <SimulaIncludeFile fileName="environments/WageningseBovenBuurt/organic.xml" />
</SimulationModelIncludeFile>
```

- We now note something very important; this file still refers to the Wageningse bovenbuurt soil files in lines 13-18! We replace **WageningseBovenBuurt** with **Unmaize** so that the correct

files are referenced. Note also that the relative location referenced is not relative to this file, but to the base file! Our file should now look like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<SimulationModelIncludeFile xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="../../../scripts/XML/SimulaXMLSchema.xsd">
   <SimulaBase name="environment">
      <SimulaBase name="dimensions">
         <SimulaConstant name="minCorner" type="coordinate">-13 -150 -30 </SimulaConstant>
         <SimulaConstant name="maxCorner" type="coordinate">13 0 30 </SimulaConstant>
         <SimulaConstant name="resolution" type="coordinate">1 1 1 </SimulaConstant>
      </SimulaBase>
      <SimulaBase name="soil">
         <SimulaTable name_column1="depth" unit_column1="cm" name_column2="bulkDensity"
unit_column2="g/cm3">0 1.20 -20 1.20 -42 1.42 -200 1.42 </SimulaTable>
      </SimulaBase>
   </SimulaBase>
   <SimulaIncludeFile fileName="environments/Unmaize/water.xml" />
   <SimulaIncludeFile fileName="environments/Unmaize/atmosphere.xml" />
   <SimulaIncludeFile fileName="environments/Unmaize/nitrate.xml" />
   <SimulaIncludeFile fileName="environments/Unmaize/phosphorus.xml" />
   <SimulaIncludeFile fileName="environments/Unmaize/potassium.xml" />
   <SimulaIncludeFile fileName="environments/Unmaize/organic.xml" />
</SimulationModelIncludeFile>
```

- Navigate to the plantParameters directory and create a new directory named **Unmaize** there. Now open the **Maize** directory and copy it into the **Unmaize** directory, you have just created, naming the copy **Unmaize**. You should now have a directory with path

plantParameters/Unmaize/Unmaize, open this. Rename the file named **maize.xml** to **unmaize.xml** and open it.

- Near the top of the file, right after the opening **SimulationModelIncludeFile** tag, you will find the following tag:

```
<SimulaBase name="maize−aerenchyma">
```

This means that all the parameters inside this tag are for plants of the type **maize-aerenchyma**. However, our plant is called **unmaize**, so we should change this line to

```
<SimulaBase name="unmaize">
```

- The file is too big to print in its entirety here, but at the bottom, there will be references to other files, in tags that look like

```
<SimulaIncludeFile fileName="plantParameters/Maize/Maize/resources.xml" />
```

Just like in the file with environment parameters earlier, we change the references by changing **plantParameters/Maize/Maize/some-File.xml** to **plantParameters/Unmaize/Unmaize/some-File.xml** in all of these tags, which would change the above tag to

```
<SimulaIncludeFile fileName="plantParameters/Unmaize/Unmaize/resources.xml" />
```

- Now we can enter our new parameters in the relevant files as explained in the previous section.

3.7. Adding or removing root classes

Our new species, *unmaize*, might not have the same root types as maize so this means we need to know how to add and remove root classes from the simulation. First we will add a new class of

laterals, called *unlaterals*, that branch from the primary root. Then we will explain how to remove root classes.

- Open the file **plantParameters/Unmaize/Unmaize/unmaize.xml**, which contains the branching rules. First we need to add the object generator for this new root. Find the **SimulaBase** tag with name lateral, which contains a tag named **branchlist**. It should look like this (edited down for clarity):

```
<SimulaBase name="lateral" objectGenerator="copyDefaults">
  <SimulaConstant type="integer" name="rootClassID">98</SimulaConstant>
  <SimulaBase name="branchList">
    <SimulaBase name="finelateral">
      <SimulaStochastic name="branchingFrequency" unit="cm" distribution="uniform"
minimum="0.15" maximum="0.35" />
      <SimulaConstant name="lengthRootTip" unit="cm"> 4</SimulaConstant>
      <SimulaConstant name="allowBranchesToFormAboveGround"
type="bool">false</SimulaConstant>
    </SimulaBase>
  </SimulaBase>
  <SimulaConstant name="density" unit="g/cm3">0.094</SimulaConstant>
</SimulaBase>
```

- Because the **objectGenerator** has value **copyDefaults**, *OpenSimRoot* will copy any parameter that is not specified for this root class from another root class named defaults. Copying parameters from other root classes is possible, see note 8. Copy this part of the file and paste it just below it. Change **lateral** to **unlateral** in the copy and delete the content of the **branchlist**. You should have added the following lines:

```
<SimulaBase name="unlateral" objectGenerator="copyDefaults">
```

```
   <SimulaConstant type="integer" name="rootClassID">98</SimulaConstant>
   <SimulaBase name="branchList">
   </SimulaBase>
   <SimulaConstant name="density" unit="g/cm3">0.094</SimulaConstant>
</SimulaBase>
```

- We want to be able to see the difference between laterals and *unlaterals* in the VTU/VTP, so change the value for **rootClassID** to an unused value.
- Next, find the **branchList** subtag of the tag named **primaryRoot**. It is in the part of the file that (edited down for clarity) looks like this:

```
<SimulaBase name="primaryRoot" objectGenerator="copyDefaults">
   <SimulaBase name="branchList">
      <SimulaBase name="lateral">
         <SimulaStochastic   name="branchingFrequency" unit="cm" distribution="uniform"
minimum="0.25" maximum="0.45" />
         <SimulaConstant name="lengthRootTip" unit="cm">10.93</SimulaConstant>
         <SimulaConstant name="allowBranchesToFormAboveGround"
type="bool">false</SimulaConstant>
      </SimulaBase>
   </SimulaBase>
</SimulaBase>
```

- All the root classes listed in the **branchlist** are root classes that will branch from the primary root. The distance between branches is determined by the **branchingFrequency** parameter (change units to day to have branches appear at a certain temporal frequency). The **lengthRootTip** parameter is the distance of the elongation zone, no branches will appear less than this distance from the root tip. See note 7 for alternative types of branching.

- Copy the **SimulaBase** tags with name lateral and everything between (lines 3-8), paste it above line 9 and change **lateral** to **unlateral** in the copy. You should now have added the following line in between the **branchList** tags:

```
<SimulaBase name="unlateral">
<SimulaStochastic name="branchingFrequency" unit="cm" distribution="uniform" minimum="0.25" maximum="0.45" />
<SimulaConstant name="lengthRootTip" unit="cm">10.93</SimulaConstant>
<SimulaConstant name="allowBranchesToFormAboveGround" type="bool">false</SimulaConstant>
</SimulaBase>
```

- Now we need to add all the parameters for this new root class. This means, that in each file in the directory **plantParameters/Unmaize/Unmaize/**, we need to add an entry for our new root class (unless default values are defined and we want to use those, see note 8). Like we did above, the easiest is to copy the entry for the lateral root class and edit that. As an example, this is what we would add to the file rootHairs.xml

```
<SimulaDirective path="unlateral">
<SimulaTable name_column1="time since creation" unit_column1="day" name_column2="rootHairLength" unit_column2="cm">0 0 1 0 2 0.028 2000 0.028</SimulaTable>
<SimulaConstant name="rootHairDiameter" unit="cm">5e-4</SimulaConstant>
<SimulaTable name_column1="time since creation" unit_column1="day" name_column2="rootHairDensity" unit_column2="#/cm2">0 2000 1 2000 2 2000 10 2000 30 0 2000 0</SimulaTable>
</SimulaDirective>
```

Now we know how to add a root class, removing a root class is easy. Go through all the files mentioned above and remove all tags and their content that have the name of that root class as either name or path attribute. If you only want to disable (remove) the root class from the simulation, simply comment out the entries you inserted in the **branchList** of the parent root class(es).

3.8. Graphical User Interface

Because editing input files can be cumbersome and confusing, as evidenced by the previous sections, a graphical user interface (GUI) was developed that provides users with a more intuitive and straightforward way of creating input files. This GUI can be found in the repository at https://gitlab.com/rootmodels/opensimroottools. The GUI allows users to add root classes through a couple of mouse clicks, and edit parameters without having to locate them in the input files. It also automatically takes the dependencies between modules into account, which makes it less likely that an invalid input file will be created. The GUI can be run on any system that has Python3 and the TkInter package installed and has been tested to work on Windows and Linux systems. Run it from a command window or terminal with the command

```
python GUIMainWindow.py
```

or

```
python3 GUIMainWindow.py
```

See figure 2 for an example of how the GUI looks on Windows 7. It should look similar on other systems.

We will now briefly describe how to use the GUI to create your own input files.

- To create an input file based on one of the existing templates in the repository, or another file you have, use the **Import XML** button in the top left. This will load the selected file, after which you can adjust the parameters as you see fit.
- If you are starting from scratch, or wish to include a different set of modules, tick the relevant boxes in the frame on the left. Please note that you might need to scroll down to see all available modules.

- After having chosen the modules, use the frame in the bottom left to add the desired root classes. To do this, select the parent root class of the root class you wish to add, type the name of the root class in the text box in the bottom left and click either **Add** or **Add Whorl**, depending on what type of branching is required. Root classes with the same name will share the same parameters. To remove a root class, select it and press **Delete**. Note that the tiller module requires a corresponding tiller root class to function.
- After selecting modules and adding root classes, edit the parameters. By selecting a root class in the bottom left, you can access the parameters for that root class. To access parameters not associated with a single root class, such as the simulation settings, shoot and soil parameters, select **Origin** in the bottom left. You are recommended not to change parameters which you do not understand.
- After everything has been changed according to your requirements, save the model to an XML file with the **Export to XML** button. **NB:** You are advised to close the GUI after this and open a new session if you wish to create more input files.
- The **About** button opens a new screen with information similar to this section.
- The **Expert Mode** tick box switches the GUI to expert mode. In expert mode, you can alter more parameters as well as change the type of parameters. You are recommended not to use expert mode unless you know what you are doing. If you're not sure what this means, don't use expert mode.
- If you run into any bugs, issues or *OpenSimRoot* is not able to run the files created by the GUI, please let the developers know by creating an issue at https://gitlab.com/rootmodels/opensimroottools. Please be precise in your description so that the error can be reproduced and please include any relevant files.

4. Notes

1.

OpenSimRoot uses the C++14 standard so older compilers might yield an error message similar to

unrecognized command line option '-std=c++14'

If this is the case, update your compiler to the newest version. The function interpreter is based on a rather large header file, and compiling may require a bigobject option under Windows. In Visual Studio it is the additional compiler option **/bigobj** and in gcc/mingw it is **-Wa,-mbig-obj**

2.

Since *OpenSimRoot* input files are generally quite large and need to adhere to a very specific structure, it is common for new users to accidentally introduce errors when editing files. This will cause the simulation to stop with an error message that will hopefully allow you to correctly diagnose and correct the problem. The error messages always start with the class and method name that produced the error, so you can locate in the code. The following list contains some common errors but it is by no means exhaustive.

- **Unit::check: someModule wants 'someParameter' in someUnit but it has unit otherUnit**

  Solution: Change the unit of the relevant parameter.
- **Database::get: Object 'someName' requested by ??? but not listed in attribute list 'parent'. List of available names: name1, name2, name3**

  Solution: Something is missing from the XML, check in the listed location and add it.
- **Database::store: Failed to store 'someName'. An object is already listed as attribute of object 'parent'.**

  Solution: You have added two tags with the same name in the XML. Delete or rename one of them.
- **Database::setFunctionPointer: function 'someFunction' not registered.**

  Solution: You have listed a function in the XML that *OpenSimRoot* does not know about. Check if you have spelled it right, if the function exists in the source code and is registered. Running OpenSimRoot with the **-l** option will list all registered functions.
- **CarbonAllocation2Root: allocation < 0**

Solution: Carbon production is lower than maintenance costs. The plant ran out of carbon. This can be due to nutrient stress or faulty photosynthesis parameters, not enough light. Check the carbon balance, or if you never want to run out of carbon, increase the seed size and set it to never expire.

- **CarbonAllocation2Shoot: allocation < 0, carbon production lower than respiration?**

  Solution: Same as the above.

- **RelativeCarbonAllocation2ShootPotentialGrowth: Potential growth is negative**

  Solution: Same as the above. Check growth rules and make sure the growth rate does not become negative due to stochastic multipliers or stress responses.

- **WatFlow: Q[i] is NAN**

  Solution: There is either too much or too little water in a soil voxel, or there is a soil gradient which is too steep. This is usually caused by too much or too little precipitation/evaporation.

- **MichaelisMenten: Fluxdensity is not a number. Debugging info: x y z**

  Solution: Problem usually arises in the solute transport or water code. This is hard to debug, but a finer resolution of the mesh or a smaller maximum time step may help.

If you encounter an error message that is not in this list, there are a few things you can do to resolve them, most of which will require you to dig into the code.

- The error message might provide insight in how to resolve it. For example, if the following error appears: **CarbonAllocation2Shoot: allocation < 0, carbon production lower than respiration?** *OpenSimRoot* indicates that the amount of carbon produced by photosynthesis is too low. This can be due to a variety of reasons, but it will point you in the right direction.
- Another option is to search the source code to see which class is generating the error message. Most error messages will print the name of the relevant class, and by searching for (part of) the error in the codebase, you should be able to find it.
- If just reading the relevant parts of the source code does not help, you can try adding in some diagnostic messages that print relevant parameters to the terminal.

- Compiling a debug version of the source code, and then running it with a debugger can also help to understand what is going on. Run the code and backtrack from the error message to see what is causing the error.
- Finally, if you are certain that the error is not caused by mistakes in the input file, open an issue on the *OpenSimRoot* repository page. Please make sure that you provide a full description of the unintended behaviour and how to reproduce it. Bug reports are seen by all the *OpenSimRoot* developers, so this is the most expedient way to get bugs resolved.

3. Example tabled output

Here is a selection of some lines from the tabled output of a maize simulation at day 38 (slightly edited for readability).

name	time	value	rate	unit	path
"averageDailyTemperature"	38.000	16.90000000	NA	"degreesC"	"//environment/atmosphere"
"plantNutrientUptake"	38.000	22571.36085292	1504.54742033	"umol"	"//plants/maize-aerenchyma_1_1/nitrate"
"plantNutrientUptake"	38.000	546.35415547	43.10032538	"umol"	"//plants/maize-aerenchyma_1_1/phosphorus"
"leafArea"	38.000	1138.74238252	50.74930913	"cm2"	"//plants/maize-aerenchyma_1_1/plantPosition/shoot"
"plantDryWeight"	38.000	12.00440272	NA	"g"	"//plants/maize-aerenchyma_1_1"
"photosynthesis"	38.000	14.90710190	1.09622375	"g"	"//plants/maize-aerenchyma_1_1/plantPosition/shoot"
"rootDryWeight"	38.000	5.45875002	NA	"g"	"//plants/maize-aerenchyma_1_1"
"rootLength"	38.000	28915.93549064	NA	"cm"	"//plants/maize-aerenchyma_1_1"

```
"rootSurfaceArea"          38.000  3777.82413565      NA              "cm2"
        "//plants/maize-aerenchyma_1_1"
"rootVolume"               38.000  58.07180872        NA              "cm3"
        "//plants/maize-aerenchyma_1_1"
"rootWaterUptake"          38.000  1489.05929962      109.57170509    "cm3"
        "//plants/maize-aerenchyma_1_1"
"shootDryWeight"           38.000  6.54565270         NA              "g"
        "//plants/maize-aerenchyma_1_1"
"totalWaterInColumn"       38.000  72480.62036434     NA              "cm3"   "//soil/water"
```

4. XML tags

OpenSimRoot input files are written in the XML markup language format. XML is a structured format that is both human-readable and machine-readable. XML files contain markup and content, the markup is used to structure the content of the file. The building blocks that make up XML files are called tags, similar to in HTML. There are three types of tags:

- Opening tags:

```
<SimulaBase>
```

- Closing tags:

```
</SimulaBase>
```

- Empty tags:

```
<SimulaBase/>
```

Everything between two tags of the same type is the content of that particular set of tags, which can include other tags. Tags can also have attributes, for example, this is a tag with the attribute "**name**", which has the value "**example**". It has "**Example content**" as content.

```
<SimulaBase name="example">Example content</SimulaBase>
```

5. *OpenSimRoot* XML tags

The tags used in *OpenSimRoot* input files are always of one of the following types. The type of the tag depends on what it is used for.

- **SimulationModel:** This acts as the top-level tag. The rest of the input file should be a subtag of a tag of this type.
- **SimulaBase:** This tag is used as a structural element, its only function is to contain other tags. Tags of this type should always have the "**name**" attribute, which is used by the code to navigate around the extensible tree structure, and might also have an **objectGenerator** attribute, which specifies how object of this type should be generated.
- **SimulaIncludeFile:** This tag is used to split input files over multiple files, which allows for separation of different sets of parameters into different files (e. g. the atmospheric parameters, soil parameters and plant parameters). The only attribute this tag should have is "**fileName**", which is a relative path to the relevant file. Everything between the **SimulationModelIncludeFile** tags in the referenced file is inserted in the file in place of this tag.
- **SimulationModelIncludeFile:** This tag is used as top-level tag in a file that is included through using a **SimulaIncludeFile** tag, see above.
- **SimulaDirective:** This tag is used to refer to a different location in the XML file. It should only have a "**path**" attribute. For all intents and purposes, everything in the content of this tag behaves as if it is part of the content of the tag referenced in the path.
- **SimulaConstant:** This tag is used to store constants of different types. It should always have the "**name**" attribute. Other common attributes are "**unit**" and "**type**". The constant stored in the tag is given in the content. A tag of this type can store a numerical value, a string or a coordinate.
- **SimulaTable:** This tag is used to save a "table" of information, a set of values, stored in the second column, that depend on the values in the first column, often representing time or

distance. It should always have the attribute "**name_column2**", which functions as the name attribute of other types of tags. Other common attributes are "**name_column1**", "**unit_column1**", "**unit_column2**", "**interpolationMethod**" and "**function**". Tags of this type are often used to store numerical values that change over time, like growth rates. The values are given in the content of the tag, in the order you get when reading the table from left to right, top to bottom.

- **SimulaStochastic:** This tag is used to let *OpenSimRoot* know that the value of this object should be pulled from some random distribution. It should always have the attributes "**name**" and "**distribution**". Depending on the distribution chosen, attributes like "**min**", "**max**", "**mean**" and "**stdev**" should also be given. It also often has the attribute "**unit**".
- **SimulaPoint:** This tag is used for points whose location changes over time, like the root tips. It should have the "**name**" and "**function**" attributes.
- **SimulaDerivative:** This tag is used for state variables that change dynamically during the simulation. It should always have the "**name**" and "**function**" attributes and often has a "**unit**" attribute. The "**function**" attribute tells *OpenSimRoot* which model to use for this state variable.
- **SimulaVariable:** This tag is used for state variables that represented something that is integrated over time. It should always have the "**name**" and "**function**" attributes and often has a "**unit**" and "**integrationFunction**" attribute. This tag is used similarly to the SimulaDerivative tag, with the difference that the value will be integrated over time.
- **SimulaLink:** This tag is used to link to a different part of the model, most often to the parameters. It should always have the "**name**" attribute and often has the "**unit**" and "**linksName**" attributes. It is often used for easy access to certain parameters or making sure parameters are listed in the input file.

There is an XSD Schema provided in the repositories and online under http://rootmodels.gitlab.io/OpenSimRoot/SimulaXMLSchema.xsd. The Schema describes the XML

rules that are allowed in *OpenSimRoot* and allows an XML editor to have auto-completion that conforms to the rules.

6. Modules

It is often most convenient to think of *OpenSimRoot* as a set of modules, each of which simulates a certain process or feature of the root system. If you want to simulate a root system and include water uptake in the simulation, this means you need to include a water module. There can be multiple modules for the same process, each of them will model the relevant process differently and every simulation will require different modules to be included. There are some dependencies between modules; for example, you cannot simulate the nitrate flows through the soil without including the water module. Every module generally corresponds to a small set of files in the repository, most of which fall into the following categories:

- A template file that contains all the dynamic state variables with their associated plugins required for the module to work.
- A file with plant-related parameters that the module requires.
- A file with environment-related parameters that the module requires.

We will now list some of the more important modules and their dependencies, if applicable.

- **Minimal model:** A minimal set of tags and input parameter needed to run a simulation that includes a plant. Examples of relevant parameters are the simulation length and the time interval between outputs. All other modules require this module to be present.
- **Geometry:** Contains plugins that keep track of geometric traits of roots, root segments and the root system as a whole. The traits are root length, surface area and root volume. Most modules require this module to be present.
- **Dry weight:** Contains plugins that calculate the dry weights of roots and shoots based on volume and density. Relevant parameters are the densities or specific weights of different plant organs. Most modules require this module to be present and you should always include this and the above two in your simulations.

- **Carbon:** Simulates carbon production, allocation and limitations in the crop. Relevant parameters include light use efficiency, seed size, maximum allocation rates and irradiation rates.
- **Respiration:** Simulates the carbon costs of respiration, both in roots as well as shoots. Relevant parameters are the respiration rates for different root classes and plant organs. Requires the carbon module to be present.
- **Exudation:** Simulates the carbon costs of root exudation. Relevant parameters are the exudation rates for each root class. Requires the carbon module to be present.
- **Root hair:** Simulates the growth of root hairs and the effect it has on nutrient uptake. Relevant parameters are the root hair density, root hair length and root hair diameter.
- **Phosphorus:** Simulates phosphorus uptake rates. A Barber-Cushman version and a version which couples to the soil water module are available, but since phosphorus is immobile, it is recommended you use the Barber-Cushman version. Typical parameters include seed phosphorus content, uptake parameters (Imax,Cmin,Km), soil phosphorus content, diffusion rates and the soil buffer power.
- **Water:** Simulates soil water flow and crop water uptake. Several water uptake models are available; if you have reasonable estimates of the root hydraulic conductivities it is recommended you use the Doussan version, otherwise you can use the Hopmans version (not recommended if you want to simulate a realistic water uptake distribution). Relevant parameters include soil hydraulic parameters and initial water content, explicit or relative transpiration rates (relative transpiration rates depend linearly on either leaf area or photosynthesis rates), root hydraulic conductivities and various atmosphere-related parameters. Requires either evaporation parameters or the evapotranspiration module to be present, as well as the carbon module (if transpiration is related to photosynthesis rates).
- **Evapotranspiration:** Simulates various atmosphere-related processes based on the Penman-Monteith equations. Calculates moisture evaporation from the topsoil and crop transpiration rates. Requires the water module to be present.

- **Nitrate:** Simulates soil nitrate flow and uptake as well as nitrate mineralisation in the soil. A Barber-Cushman version and a version which couples to the soil water module are available, but since nitrate is mobile, it is recommended you use the version that couples soil nitrate flow to the soil water flow. Typical parameters include initial soil nitrate content, seed nitrate content, mineralisation parameters, uptake parameters (Imax,Cmin,Km), nitrogen fixation rates (for legumes) and diffusion parameters.
- **Potassium:** Simulates soil potassium flow and uptake. A Barber-Cushman version and a version which couples to the soil water module are available. Typical parameters include initial soil potassium content, seed potassium content, uptake parameters (Imax,Cmin,Km) and diffusion parameters.
- **Root cortical aerenchyma:** Simulates the formation and effect of root cortical aerenchyma. Typical parameters include aerenchyma formation rates and parameters related to the effects of aerenchyma, such as reduction in respiration rates and remobilisation of phosphorus.
- **Root cortical senescence:** Simulates root cortical senescence and its effects. Typical parameters include the cortical senescence rates and the effects of it on radial hydraulic conductivity.
- **Nutrient stress:** Simulates the effects of nutrient deficiency on plant growth and functioning. Relevant parameters include the minimum and optimal nutrient contents for each plant organ and parameters relating the nutrient stress levels to modifiers that modulate plant functioning. Requires at least one nutrient module to be present.
- **Local nutrient responses:** Simulates root responses to local nutrient concentrations. Typical parameters include transfer functions that relate local nutrient concentrations to branching rates, root elongation rates and gravitropism. Requires at least one nutrient module to be present.
- **Tiller:** Simulates the formation of tillers and roots emerging from these tillers. Typical parameters include number of tillers and tiller roots, emergence times of tillers, tiller root types (with associated parameters) and parameters for the tiller emergence pattern.

7. Including whorl roots

If the branching parameters are given as in the example in the text, new branches will emerge along the root as it grows, at the intervals specified. However, some crops have roots that do not emerge like this, but in a "whorl" of roots that all emerge at roughly the same time. Examples are the nodal roots in maize. For this branching pattern, add the following lines to the branchlist of the parent root:

```
<SimulaBase name="nodalroots">
   <SimulaConstant type="integer" name="numberOfBranches/whorl" unit="#">3</SimulaConstant>
   <SimulaConstant type="integer" name="maxNumberOfBranches" unit="#">3</SimulaConstant>
   <SimulaConstant type="time" name="branchingTimeOffset" unit="day">9.</SimulaConstant>
   <SimulaConstant name="branchingSpatialOffset" unit="cm">1.5</SimulaConstant>
</SimulaBase>
```

Here the first two parameters are the number of branches in the root whorl, the third parameter is the emergence time of the roots and the final parameter is the distance from the base of the parent root where the roots will appear.

8. Defining and using classes of default parameters

Any root class that is declared in the main parameter file (which was called **unmaize.xml** in our example) with **copyDefaults** as "**objectGenerator"** attribute will use the default parameters if none are provided. It is possible to define different classes of default parameters, which is useful if you want all the major roots to use one set and all the laterals to use another. In the root class declaration, you can tell it to use parameters from a certain default group with the following tag

```
<SimulaConstant type="string" name="copyDefaultsFrom">../defaultsMajorAxis</SimulaConstant>
```

To declare the default parameters for this group, which is named **defaultsMajorAxis**, simply add parameters like you would for any root class but use **defaultsMajorAxis** as the root name. Like this:

```
<SimulaDirective path="defaultsMajorAxis">
   <SimulaTable name_column1="time since creation" unit_column1="day"
name_column2="rootHairLength" unit_column2="cm">0 0 1 0 2 0.028 2000 0.028</SimulaTable>
```

```
  <SimulaConstant name="rootHairDiameter" unit="cm">5e-4</SimulaConstant>
  <SimulaTable name_column1="time since creation" unit_column1="day"
name_column2="rootHairDensity" unit_column2="#/cm2">0 2000 1 2000 2 2000 10 2000 30 0 2000
0</SimulaTable>
</SimulaDirective>
```

References

(1) Dunbabin Vanessa M., Postma Johannes A., Schnepf Andrea, et al. (2013) Modelling root-soil interactions using three-dimensional models of root growth, architecture and function. *Plant and Soil* **372(1-2)**, 93–124

(2) Dunbabin Vanessa M., Diggle Art J., Rengel Zdenko, et al. (2002) Modelling the interactions between water and nutrient uptake and root growth. *Plant and Soil* **239(1)**, 19–38

(3) Gérard Frédéric, Blitz-Frayret Céline, Hinsinger Philippe, et al. (2017) Modelling the interactions between root system architecture, root functions and reactive transport processes in soil. *Plant and Soil* **413(1-2)**, 161–180

(4) Javaux Mathieu, Schröder Tom, Vanderborght Jan, et al. (2008) Use of a threedimensional detailed modeling approach for predicting root water uptake. *Vadose Zone Journal* **7(3)**, 1079

(5) Leitner Daniel, Klepsch Sabine, Bodner Gernot, et al. (2010) A dynamic root system growth model based on l-systems. *Plant and Soil* **332(1-2)**, 177–192

(6) Lobet Guillaume, Pagès Loïc, Draye Xavier (2014) A modeling approach to determine the importance of dynamic regulation of plant hydraulic conductivities on the water uptake dynamics in the soil-plant-atmosphere system. *Ecological Modelling*, **290**, 65–75

(7) Pagès Loïc, Vercambre Gilles, Drouet Jean-Louis, et al. (2004) Root typ: a generic model to depict and analyse the root system architecture. *Plant and soil* **258(1)**, 103–119

(8) Pierret Alain, Doussan Claude, Capowiez Yvan, et al. (2007) Root functional architecture: A framework for modeling the interplay between roots and soil. *Vadose Zone Journal* **6(2)**, 269

(9) Wu L., McGechan M. B., McRoberts Neil, et al (2007) Spacsys: integration of a 3d root architecture component to carbon, nitrogen and water cycling—model description. *Ecological Modelling* **200(3-4)**, 343–359

(10) Bingham Ian J., Lianhai Wu (2011) Simulation of wheat growth using the 3D root architecture model SPACSYS: Validation and sensitivity analysis. *European journal of agronomy* **34(3)**, 181-189

(11) Berntson G. M. (1994) Modelling root architecture: are there tradeoffs between efficiency and potential of resource acquisition? *New Phytologist* **127(3)**, 483-493

(12) Postma Johannes A., Lynch Jonathan P. (2012) Complementarity in root architecture for nutrient uptake in ancient maize/bean and maize/bean/squash polycultures. *Annals of botany* **110(2)**, 521-534

(13) Postma Johannes A., Dathe Annette, Lynch Jonathan P. (2014) The optimal lateral root branching density for maize depends on nitrogen and phosphorus availability. *Plant physiology* **166(2)**, 590-602

(14) Postma Johannes A., Kuppe Christian, Owen Markus R., et al (2017) Opensimroot: widening the scope and application of root architectural models. *New Phytologist* **215(3)**, 1274-1286

(15) Schnepf Andrea, Black Christopher K., Couvreur Valentin, et al (2020) Call for participation: Collaborative benchmarking of functional-structural root architecture models. The case of root water uptake. *Frontiers in Plant Science* **11**

Figure captions

1. A maize root system simulated by *OpenSimRoot* and visualised by ParaView. The different colours indicate the age of the root segment, with blue segments being the oldest and red segments the youngest.

2. The OpenSimRoot input file generation GUI as it looks on Windows 7.