ספר פרויקט

פרטי סטודנט:

שם סטודנט	ת.ז.	מייל
רון טלר	305142812	Ron.teller.8@gmail.com
שי טלר	305142820	Shai.teller@gmail.com

פרטי מנחה:

שם	מייל
שרונה ט. לוי	stlevy@edu.haifa.ac.il

שם פרויקט:
פרויקט NetTango

מטרת הפרויקט:
בניית ממשק קידוד מבוסס בלוקים ב NetTango למודלים ב NetLogo על מנת ליצור ממשק גרפי תכנותי עבור תלמידים שיוכלו לקודד איך המודלים מבוססי סוכנים יפעלו. המטרה העיקרית היא שמשתמשים יוכלו לקודד התנהגות של "סוכנים" בתוך ה"עולם" בעזרת גרירית "לבנות" של קוד בממשק הגרפי ש NetTango מספק. ה"סוכנים" וה"עולם" מתוכנתים ב NetLogo, ה"לבנות" של קוד מיוצרות בעזרת הכלי NetTango. ייבנו שני ממשקים עיקריים:

1. מודל **הנמלים** – ממשק לתכנות מושבות נמלים שמתקשרות בעזרת פרומונים על מנת למצוא אוכל ולהחזירו לקן.
2. **חומר.רב.בתנועה** – ממשק לתכנות מערכות בכימיה ובפיזיקה.

אתר פרויקט:

קישורים להרצת פרויקט - הפרויקט ניתן להרצה בדפדפן. יש ללחוץ על הקישורים שמתחת, והפרויקט ירוץ בדפדפן, נגיש וחברותי למשתמש. עדיפות לדפדפן Chrome (אם יש בעיות טעינה).

- תת-פרויקט ***נמלים***: https://ron-teller.github.io/ants-netlogo.github.io
- תת-פרויקט **חומר.רב.בתנועה**: https://ron-teller.github.io/mmm-netlogo.github.io

קישורים למאגר קוד - ניתן לעבור על קוד הפרויקט בקישורים הבאים:

- תת-פרויקט ***נמלים***: https://github.com/Ron-Teller/ants-netlogo.github.io
- תת-פרויקט **חומר.רב.בתנועה**: https://github.com/Ron-Teller/mmm-netlogo.github.io

Table of Contents

1. Introduction

1.1. Objective

The aim of this project is to create an interactive learning environment for students to advance their learning and understanding of a given complex system or phenomena in the field of science. To achieve this goal, a web application was built that would enable users to interact with a given computational model of a complex system by use of a block based coding interface in order to study the phenomena observed arising from the complex system being simulated.

1.2. Structural Design

The application consists of two main components:

1.2.1. Agent based model

A computational model simulating the actions and interactions of agents within a complex system, for the purpose of assessing their effects on the system as a whole (e.g. formation of solar system in a system with bodies of mass that interact with each other by means of gravity).

This component was constructed using the NetLogo platform, a programmable modeling environment for simulating natural and social phenomena. The NetLogo platform is a powerful tool that comes equipped with a plethora of features including fast modeling of complex systems and construction of a user interface that enables users to visually run simulations of a modeled system under different parameters and observe it develop over time.

1.2.2. Block based programming interface

A block based visual programming platform that lets users program the functionality and behavior of a model by manipulating graphical blocks via spatial arrangement and parameter configuration.

This component was constructed using the NetTango platform, a domain-blocks-based interface for the NetLogo agent-based modeling environment. NetTango is designed to bring block based programming functionality to models programmed with the NetLogo platform.

1.3. Sub-Projects

This project consisted of 2 separate and independent sub-projects, each one modeling a different complex system. Both sub-projects were constructed using the same procedures and tools, with the same goal in mind of creating an interactive learning environment for students to advance their learning and understanding of the given complex system modeled in each sub-project.

1.3.1. Ants

The Ants project models the complex system of pheromone communication used by an ant colony working together to scavenge for food and return it to their nests. This model allows the user to observe how pheromone communication is utilized by ants to:

- Form an ant trail on the shortest path possible to a food source and back to their nest.
- Navigate through obstacles. Ants may be forced to find new paths or discover shorter ones given walls can be added or removed in real-time, changing the path an ant has to take to find food or return to their nest.

This model also features creation (and removal) of ant colonies, with each colony having its own nest and population competing for food with other colonies.

This sub-project was designed with our moderator Sharona T. Levy and Nurit Gil.

The project can be run in the browser by clicking on the following link (please use Chrome if the page does not load): https://ron-teller.github.io/ants-netlogo.github.io

1.3.2. Much.Matter.In.Motion (MMM)

This sub-project is a continuation of the already existent Much.Matter.In.Motion modeling platform (Levy, Saba, Lipov & Hel-Or, 2019). The MMM platform enables scientific modeling of computational models of varied physical and chemical complex systems by use of block based programming for the purpose of advancing students learning in a wide range of phenomena pertaining to these models.

The MMM platform's model component is programmed in the NetLogo language, with the block based programming component built with the Blockly, a client side library for JavaScript.

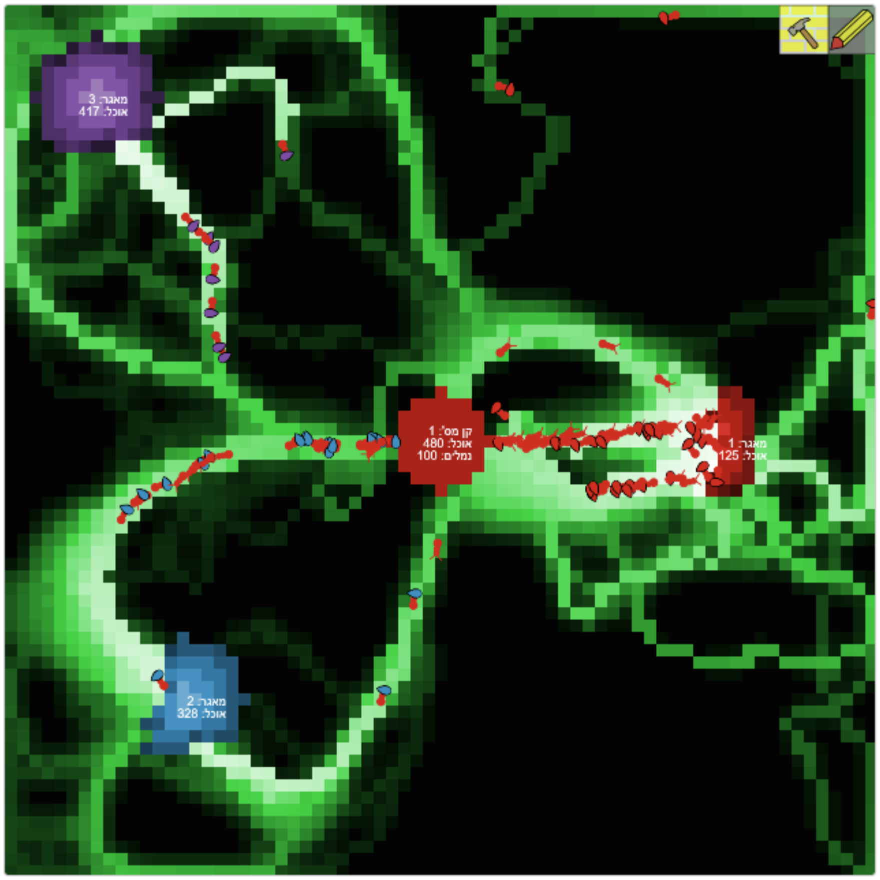
The main objectives of this sub-project were to:

- Create a block based programming interface using the NetTango platform instead of the existing Blockly platform currently used in MMM.
- Add functionality and features to existing MMM model.
- Enhance the user interface and experience.

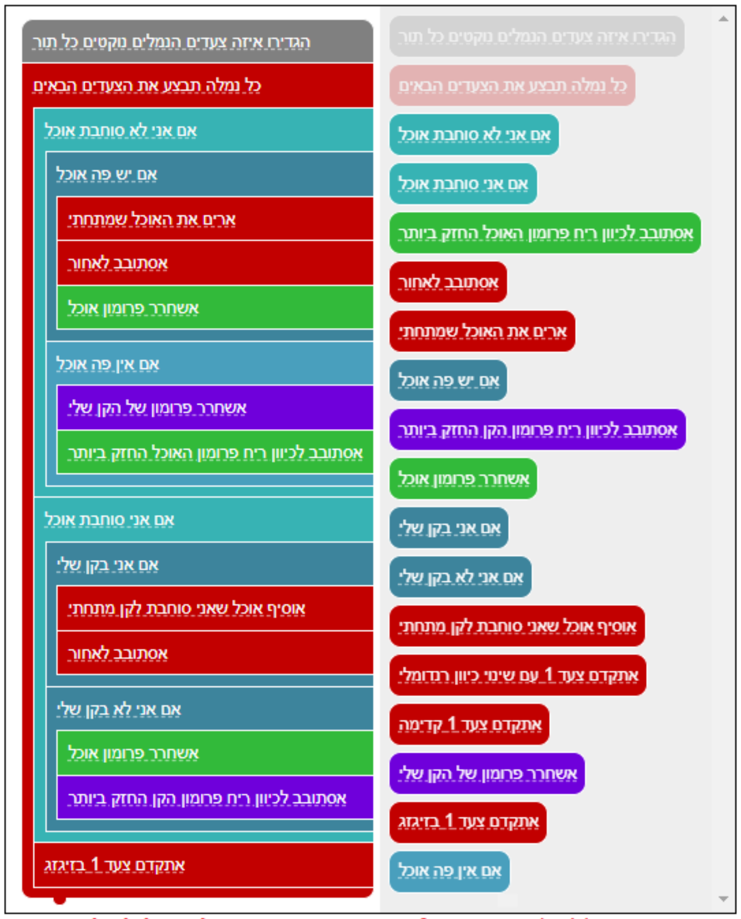
This sub-project was designed with our moderator Sharona T. Levy and Janan Saba whom is currently leading the MMM project.

The project can be run in the browser by clicking on the following link (please use Chrome if the page does not load): https://ron-teller.github.io/mmm-netlogo.github.io

** Below is an edited photo from the "Ants" sub-project showcasing the two main components - the agent based model and block based programming interface side by side. The complex system (left) is modeled in NetLogo, which contains all logic and code to simulate (and visualize) the "Ants" computational model. On the right is the block-based programming interface created in NetTango which allows the user to define an ants behavior by placing blocks in a certain order.*



Complex system simulated by an agent-based computational model created with *NetLogo*



```
הגדירו איזה צעדים הנמלים נוקטים כל תור
כל נמלה תבצע את הצעדים הבאים
  אם אני לא סוחבת אוכל
    אם יש פה אוכל
      ארים את האוכל שמתחתי
      אסתובב לאחור
      אשחרר פרומון אוכל
    אם אין פה אוכל
      אשחרר פרומון של הקן שלי
      אסתובב לכיוון ריח פרומון האוכל החזק ביותר
  אם אני סוחבת אוכל
    אם אני בקן שלי
      אוסיף אוכל שאני סוחבת לקן מתחתי
      אסתובב לאחור
    אם אני לא בקן שלי
      אשחרר פרומון אוכל
      אסתובב לכיוון ריח פרומון הקן החזק ביותר
  אתקדם צעד 1 בזיגזג
```

```
הגדירו איזה צעדים הנמלים נוקטים כל תור
כל נמלה תבצע את הצעדים הבאים
אם אני לא סוחבת אוכל
אם אני סוחבת אוכל
אסתובב לכיוון ריח פרומון האוכל החזק ביותר
אסתובב לאחור
ארים את האוכל שמתחתי
אם יש פה אוכל
אסתובב לכיוון ריח פרומון הקן החזק ביותר
אשחרר פרומון אוכל
אם אני בקן שלי
אם אני לא בקן שלי
אוסיף אוכל שאני סוחבת לקן מתחתי
אתקדם צעד 1 עם שינוי כיוון רנדומלי
אתקדם צעד 1 קדימה
אשחרר פרומון של הקן שלי
אתקדם צעד 1 בזיגזג
אם אין פה אוכל
```

Block-based programming interface created with *NetTango* that interacts with models created in *NetLogo*

2. Tools

The two core tools utilized in this project were NetLogo and NetTango.

NetLogo was used to model the complex system of each sub-project. The NetLogo environment was used to code the computational model using the NetLogo language, visualize the model with a graphical view of the model and create a user interface consisting of various widgets to interact with the model and plot data.

NetTango was used to create the block-based programming interface for models created in NetLogo.

2.1. NetLogo

NetLogo is a programmable modeling environment for simulating natural and social phenomena, and is well suited for modeling complex systems developing over time. NetLogo is used to program computational models of complex systems with an agent-based approach, allowing the modeler to give instructions to hundreds and thousands of “agents” operating independently, thereby simulating the actions and interactions of agents within a complex system, for the purpose of assessing their effects on the system as a whole. This makes it possible to explore the micro-level behavior of individuals (agents) and the macro-level patterns (phenomena) that emerge from the interaction.

NetLogo lets the user run simulations of an authored model in NetLogo under various conditions and explore their behavior.

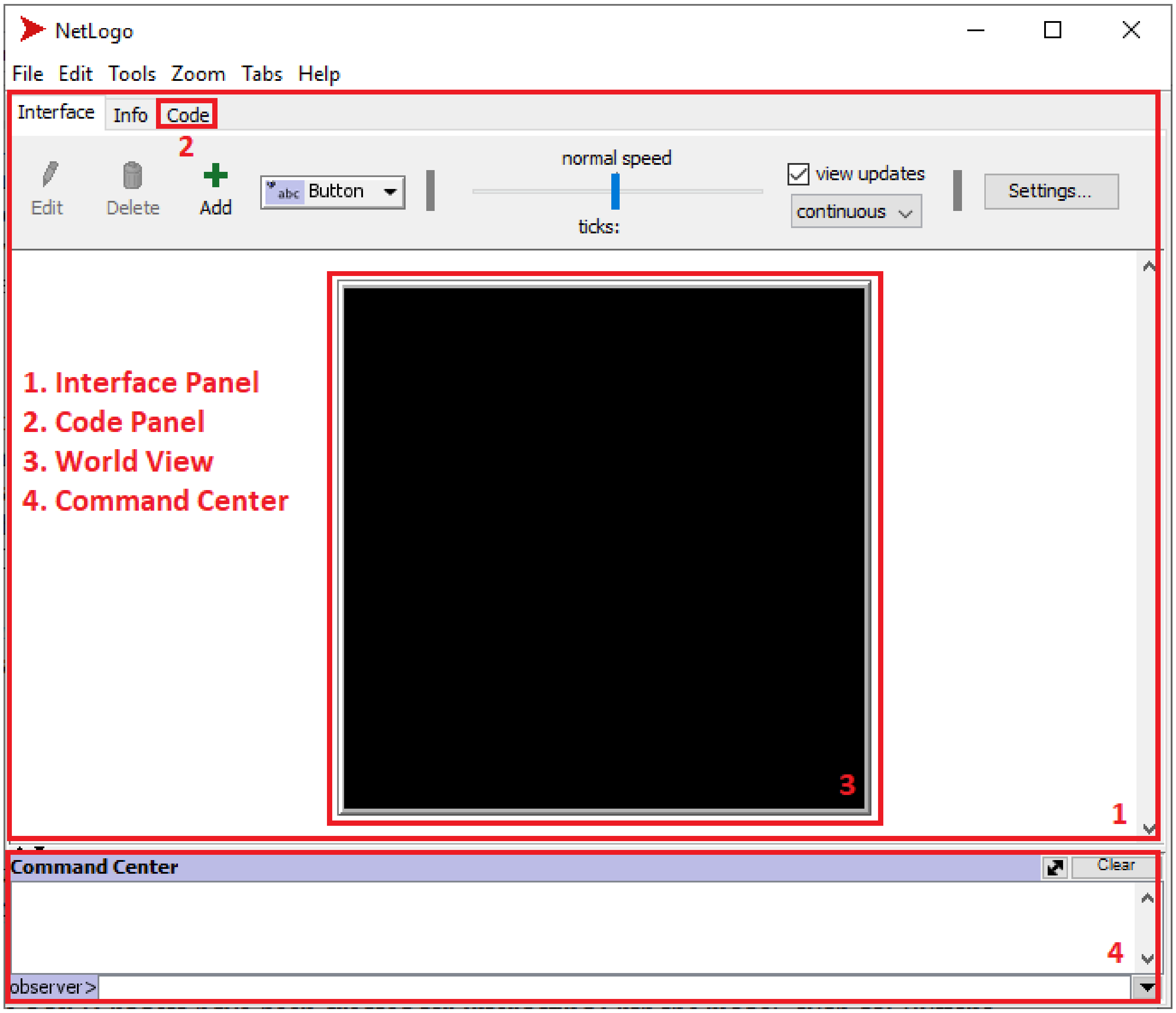
NetLogo comes equipped with the ability to create a user interface to interact with the model. This includes a number of important features, such as the creation of widgets allowing the user to select different parameter values to run the simulation under different conditions, and a world view that visually displays the model develop over time.

2.1.1.NetLogo IDE

The NetLogo platform supplies an IDE for programmers to create a computational model and interface for any given system they wish to simulate. Programming with the NetLogo IDE is done mostly by using the ***Interface*** and ***Code*** panels. The interface panel lets the programmer place and organize widgets that allow the user to interact with the model. The model itself is programmed in the code panel using the NetLogo programming language, and contains the core logic and code that runs the model. In the interface panel is a ***World View*** that visually displays the model being run.

The ***Command Center*** panel is designed mainly for developers to write commands in the NetLogo programming language and see it take affect on the model. It can be used for developing and debugging purposes.

** Below is an example of a new model set up and ready to be programmed in NetLogo IDE. The* ***interface*** *panel is currently opened, and in this panel widgets such as buttons, sliders, graphs etc. may be added. A* ***world view*** *is always present in the interface, and is currently empty since nothing has been programmed yet.*

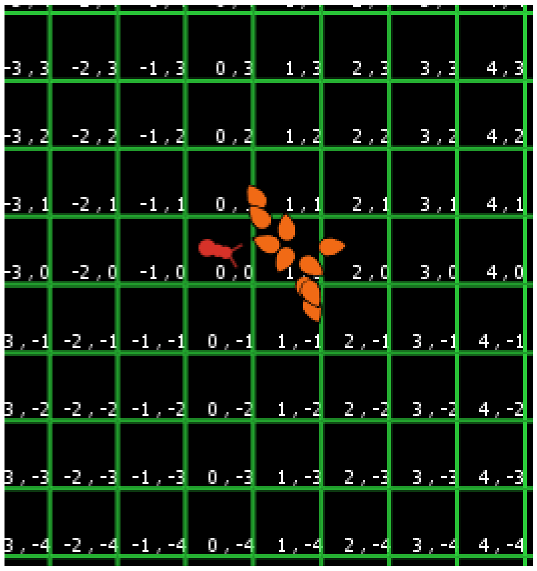


2.1.2.Model

Models are programmed using the NetLogo programming language. NetLogo is designed to create models using an agent-based approach. A typical NetLogo model is built up of “agents” which move around the world which is a 2D grid where each cell is referred to as a “patch”. The agents can move around the world and also interact with other agents and patches.

Many complex systems can be modeled using this agent-based approach since they are all inherently made up of “agent” entities that interact with each other and move around a given space, such as bodies of mass in space, cells inside an organism and ant colonies scavenging for food inside a certain area.

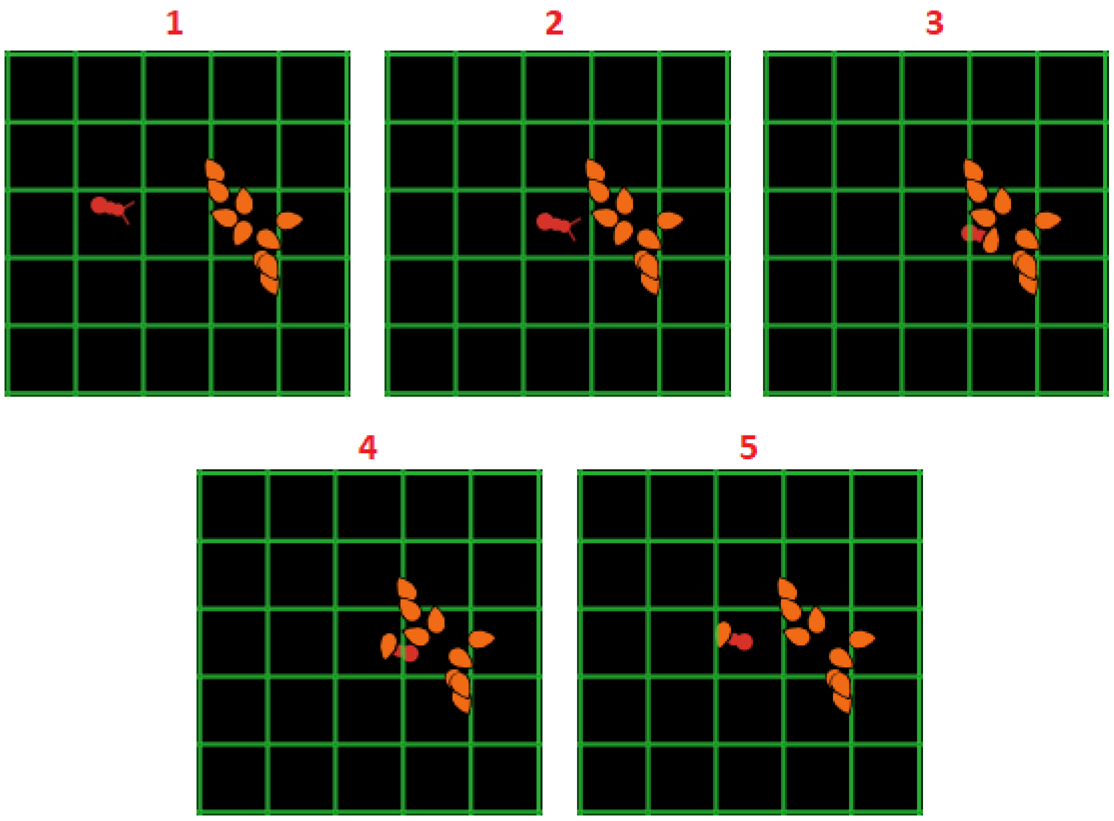
When modeling a complex system in NetLogo, it is the task of the programmer to decide how to create a computational model of the complex system, which types of “agents” to create, how they interact with each other and move around the “patches” in the world.



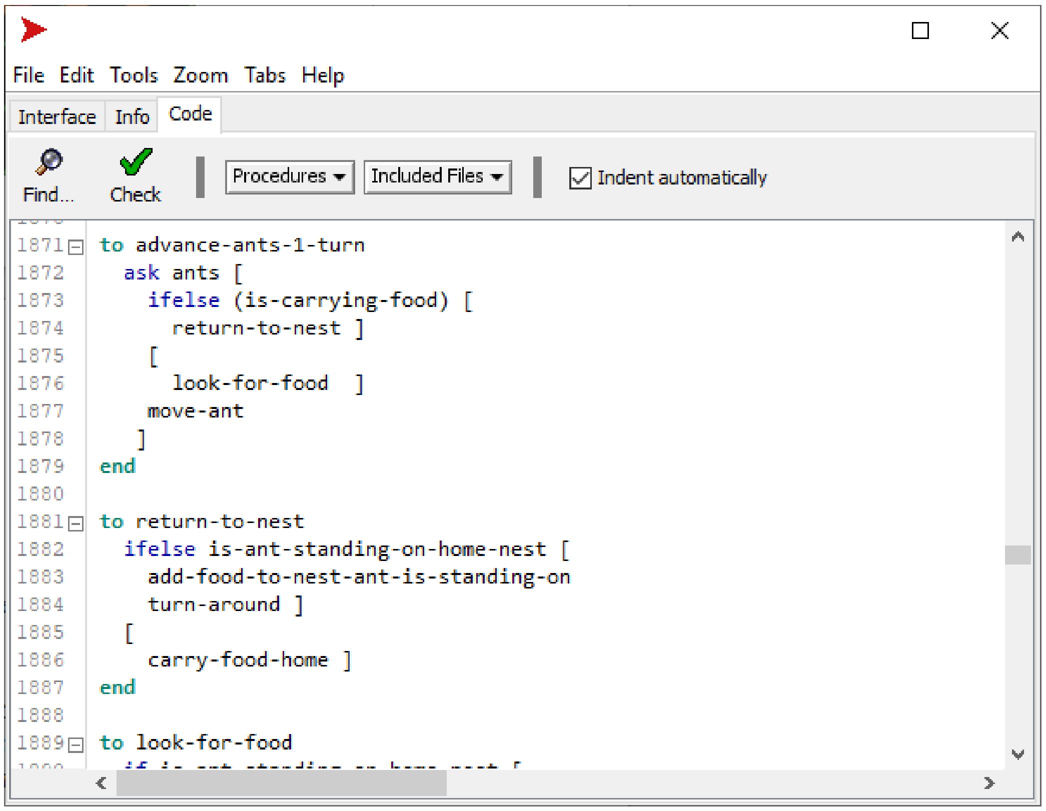
** Above is an example of a model in NetLogo where the agents are the ants and food. The ants interact with the food by picking it up and bringing it back to the nest. The world is a 2D grid where each cell is called a "patch". In the above picture, the patches have their coordinates labeled on them. The ant's location is currently in the patch with coordinates (0,0), which is the center of the world.*

Models in NetLogo are usually advanced in discrete time steps referred to as "ticks". Each tick, the world advances a single step (each model is programmed differently what is advanced, and in what manner). In a single tick it is typical for models to change the state and position of each agent, and update any other world state that needs to change accordingly.

** Below is an example of a model advancing in discrete steps ("ticks"). The ant moves forward in search for food, and when it finds some it carries it back home.*



Coding is primarily written using the **Code** panel of the IDE, written in the NetLogo programming language. The language is extensively documented and there exists many example models in the library for beginner and experienced programmers to learn from.

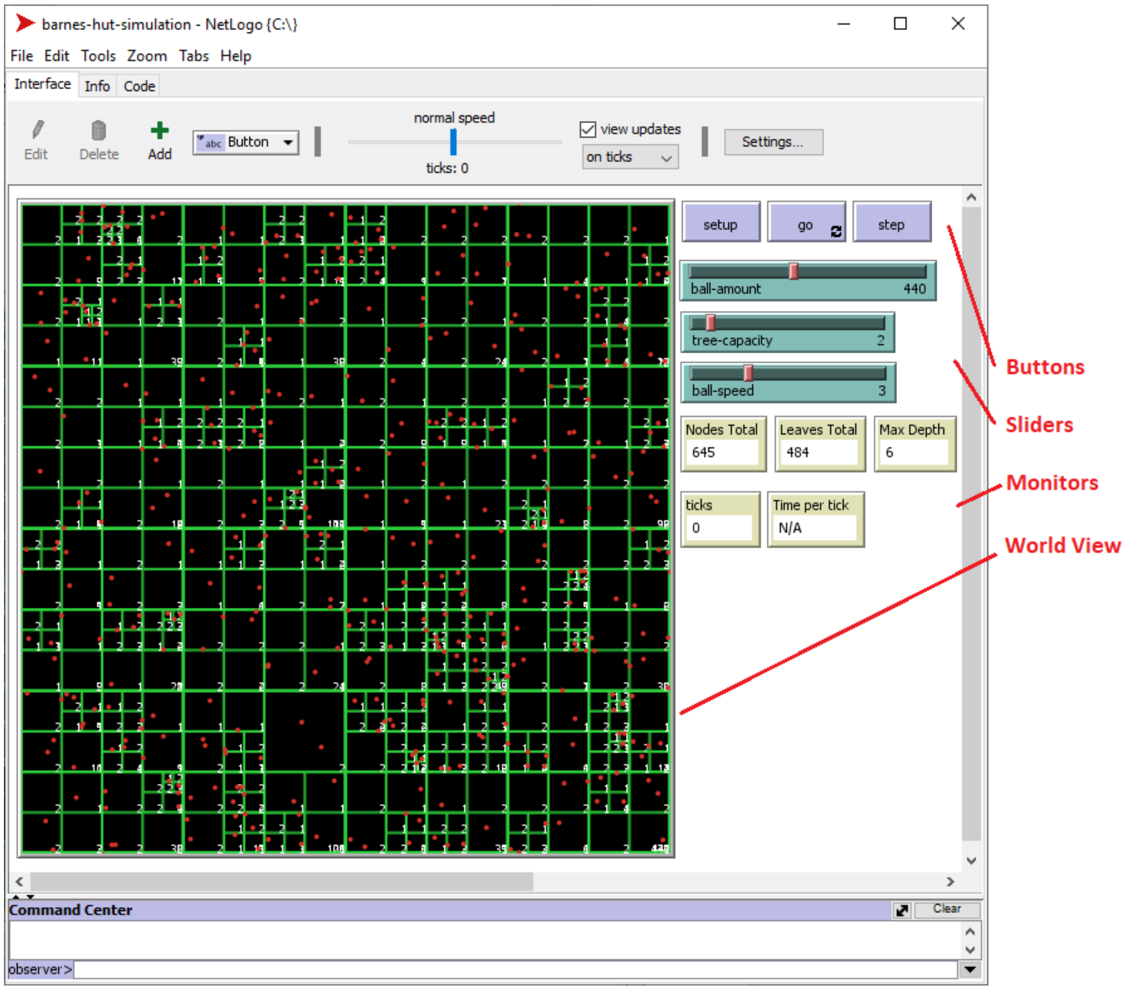


```
to advance-ants-1-turn
  ask ants [
    ifelse (is-carrying-food) [
      return-to-nest ]
    [
      look-for-food  ]
    move-ant
   ]
end

to return-to-nest
  ifelse is-ant-standing-on-home-nest [
    add-food-to-nest-ant-is-standing-on
    turn-around ]
  [
    carry-food-home ]
end

to look-for-food
```

2.1.3. Interface

The **interface** panel in the NetLogo IDE allows the programmer to create and organize widgets that interact with the model. The widgets are usually created for the purpose of running/stopping execution of the model, setting parameters so the model can be run under different conditions and monitoring data output by the model.

Another main component in the interface is the **world view**, which visually displays the model in its current state. It is easy to both create widgets and set/update the world view using the NetLogo IDE, which is a big advantage for anyone programming in NetLogo since these programming capabilities come integrated with the IDE.



ball-amount 440
tree-capacity 2
ball-speed 3
Nodes Total 645
Leaves Total 484
Max Depth 6
ticks 0
Time per tick N/A

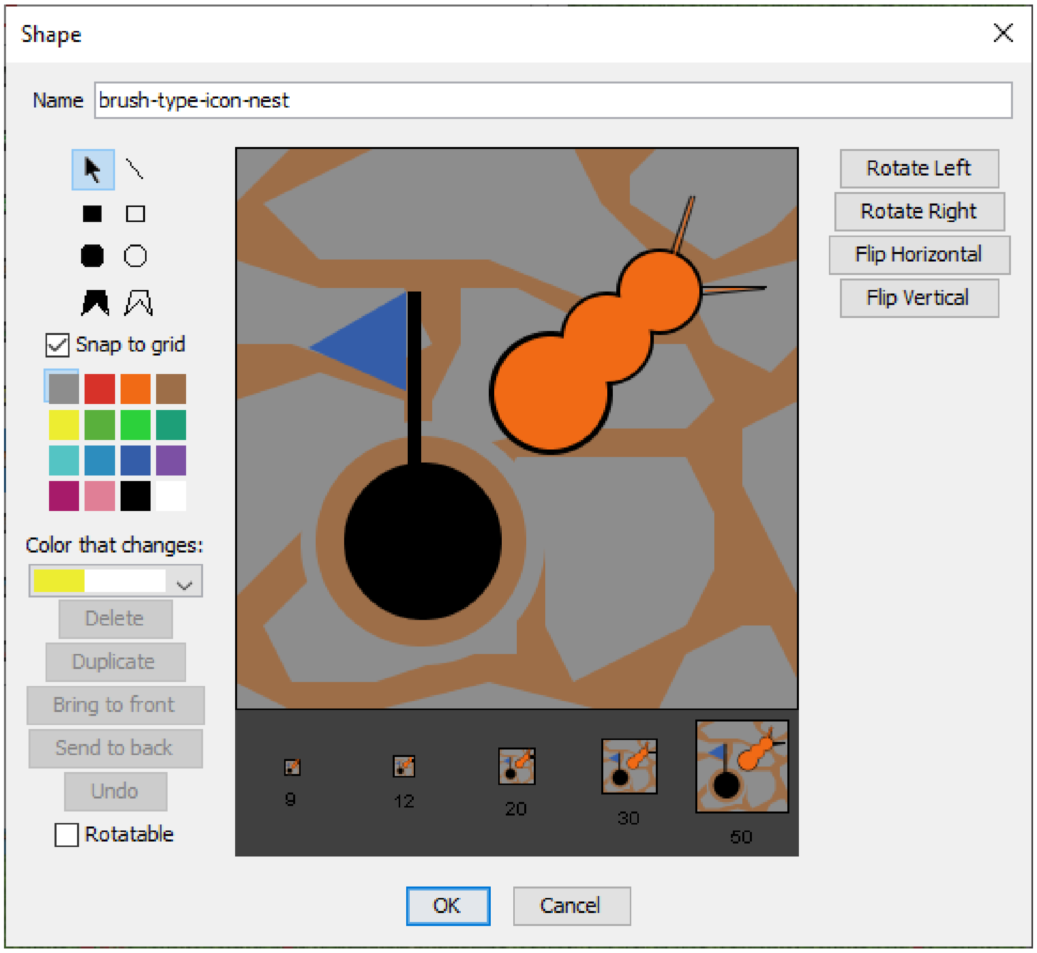
Above is example of the NetLogo IDE currently set up in the **interface** panel.

You may notice quite a few widgets have been created for interacting with the model, such as: Buttons ("setup", "go", "step") which in this case are used to run and stop the model, Sliders ("ball-amount", "tree-capacity", "ball-speed") which are used to set different values for parameters and see how the model works under different conditions, and Monitor widgets that output data ("Node Total", "Leaves Total" etc..). NetLogo supports creating 9 different types of widgets in the interface.

In this case, a mathematical model of the Barnes-Hut Simulation has been programmed in NetLogo, an approximation algorithm for performing n-body simulation. The purpose of this model was to demonstrate how the algorithm works under different parameters.

2.1.4. Graphics

The *Turtles Shapes Editor* is used in NetLogo to create shapes for agents. It already comes built-in with a variety of common shapes of various animals, plants, household items and more. It has an intuitive and simple UI for creating and configuring shapes.

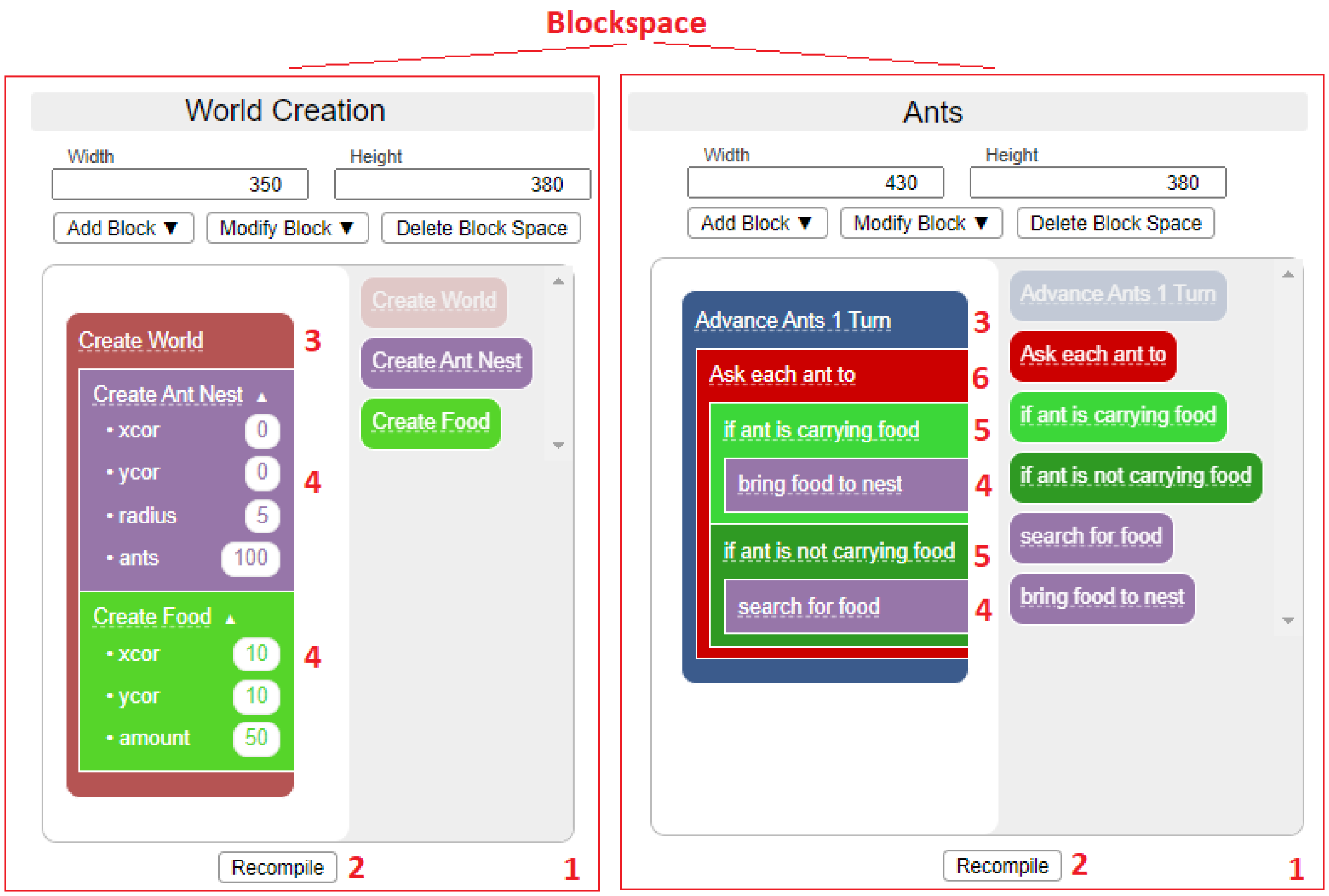


2.2. NetTango

The NetTango platform is a domain-blocks-based interface for the NetLogo agent-based modeling environment designed to bring block based programming functionality to models programmed with the NetLogo platform.

The NetTango editor is web based and intended for browser use. The editor contains two main editing sections, a NetLogo editing section which has the same functionality of the NetLogo IDE only implemented in web, and a block-based programming editing section which is the heart of NetTango and used for creating a block-based interface for NetLogo models.

* *An example of the blocks editing section on NetTango. Currently there are 2 block spaces (named "World Creation" and "Ants"), and each one has its own set of blocks inside it. NetTango supports different types of blocks that enable defining the behavior of a model functionally.*



1. Blockspace
2. Recompile
3. "Procedure" type block
4. "Command" type block
5. "if" type block
6. "ask agents" type block

2.2.1. Creating Blocks

The next sections will briefly describe in a top-down approach how blocks are created and customized in NetTango.

2.2.1.1. Block Spaces

A block space is the first thing that must be created. A block can only be created inside a block space, and a block space can contain any number of blocks. Blocks in one block space can not be used in another block space (although it is possible to create identical blocks). A block space is essentially a defined set of blocks that can be spatially arranged. A meaningful and unique name can (and should) be set for every block space. In the example above, two different block spaces were created, one for creating entities in the world, and the other for defining the behavior.

2.2.1.2. Block Types

NetTango supports creating different types of blocks that enable creating a diverse and programmable set of blocks that can define the functionality and behavior of a model. The most commonly used type of blocks are “procedure”, “if”, “command” and “ask agents”. Each type of block is set to be used in a different way, such as being shaped in a way to allow an inner set of blocks to be attached inside it. An example for this is the “if” block – if the predicate inside the “if” block is met, then all blocks inside of it are executed. In the example supplied above, an “if” block named “if ant is carrying food” is programmed to check if the ant carrying food, and if it is then it runs the block inside of it - “bring food to nest”, which it executes.

2.2.1.3. *Block Parameters/Properties*

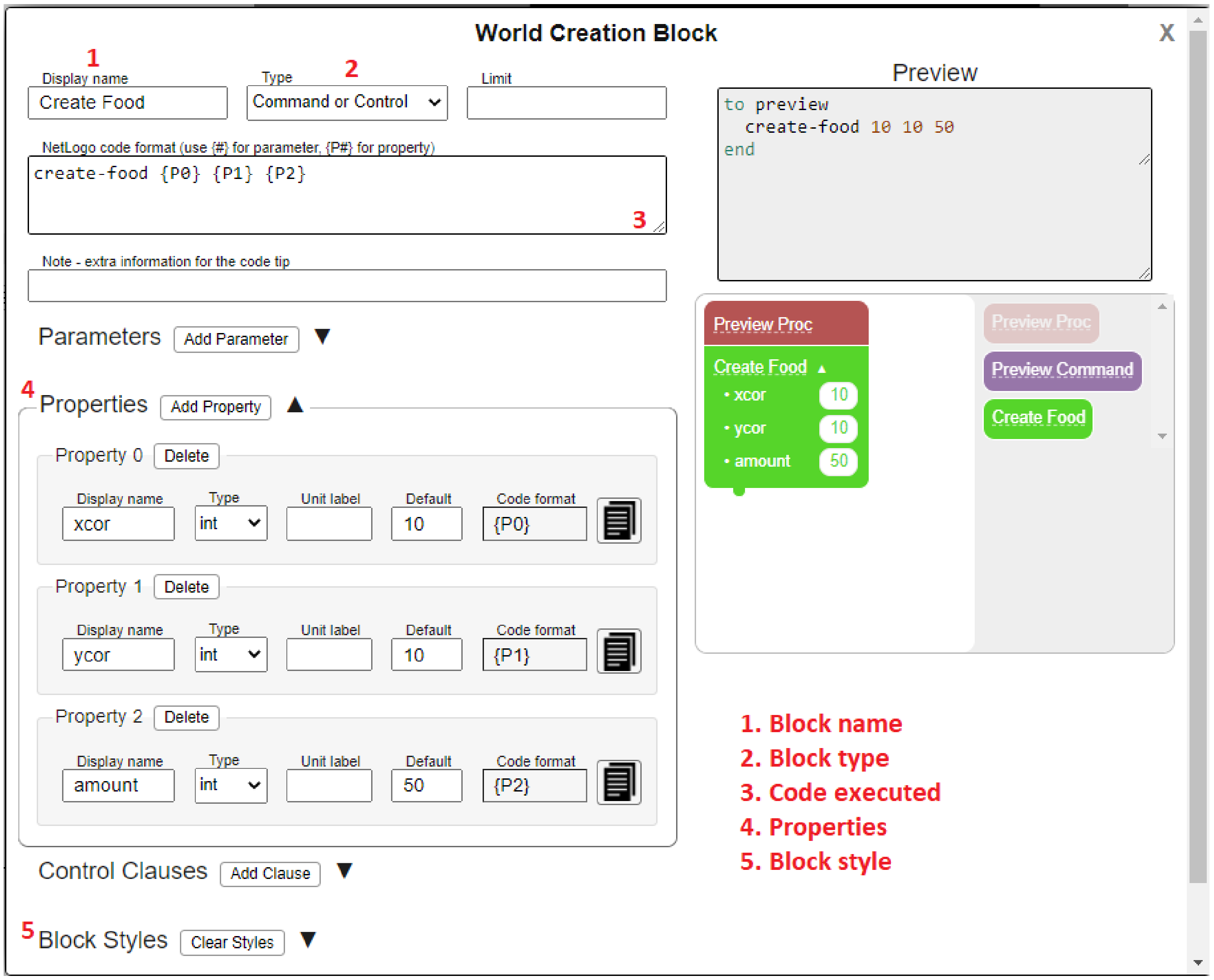
Parameters can be set for blocks by the user. The blocks can access these parameters and execute code accordingly. In the above example, the “create ant nest” block accepts 4 parameters, that tell it the coordinates of the nest to be created, its size (radius) and the amount of ants inside it.

2.2.1.4. *Block style*

A block can by styled in a certain number of ways, most notably setting the color of the block, text and border. This is an important part of the user interface since it is possible to visually group a number of blocks due to their color, and overall improve the aesthetics in the block-based interface.

2.2.1.5. *Editing a block*

Editing a block is done in the block edit window. There are numerous setting that can be set for each block to define its function, use and style.


World Creation Block

Display name: Create Food

Type: Command or Control

Limit:

NetLogo code format (use {#} for parameter, {P#} for property)

```
create-food {P0} {P1} {P2}
```

Note - extra information for the code tip:

Preview

```
to preview
  create-food 10 10 50
end
```

Property	Display name	Type	Unit label	Default	Code format
Property 0	xcor	int		10	{P0}
Property 1	ycor	int		10	{P1}
Property 2	amount	int		50	{P2}

1. Block name
2. Block type
3. Code executed
4. Properties
5. Block style

* *A block edit window with the main sections numbered.*

2.2.2. Integration with NetLogo

The NetTango editor can integrate the block-based interface created in the blocks editing section with the model currently uploaded in the NetLogo editing section, thereby enabling the blocks to affect the NetLogo model and create a “new” product of a NetLogo model that can be interacted with a block-based interface. This is achieved by editing the code executed by a block to reference a procedure within the NetLogo model’s code, or by executing a procedure type block from the NetLogo model’s code.

2.2.3. End Product

The end product used by the client is an HTML file that is a web version of the learning environment created in NetTango.

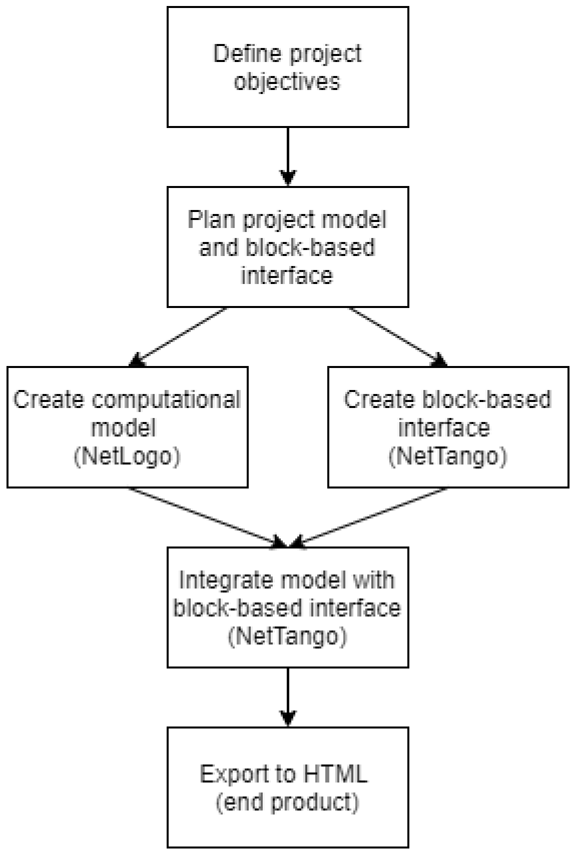
A NetTango project can be exported as HTML, enabling users to run the project in a web browser.

The HTML file is in fact the final product of this project meant for use by the client. It is the “interactive learning environment meant for use by students to interact with a computational model of a complex system by means of a block-based interface”.

3. Workflow

This section will give an overview of the methods and steps used to carry out this project.

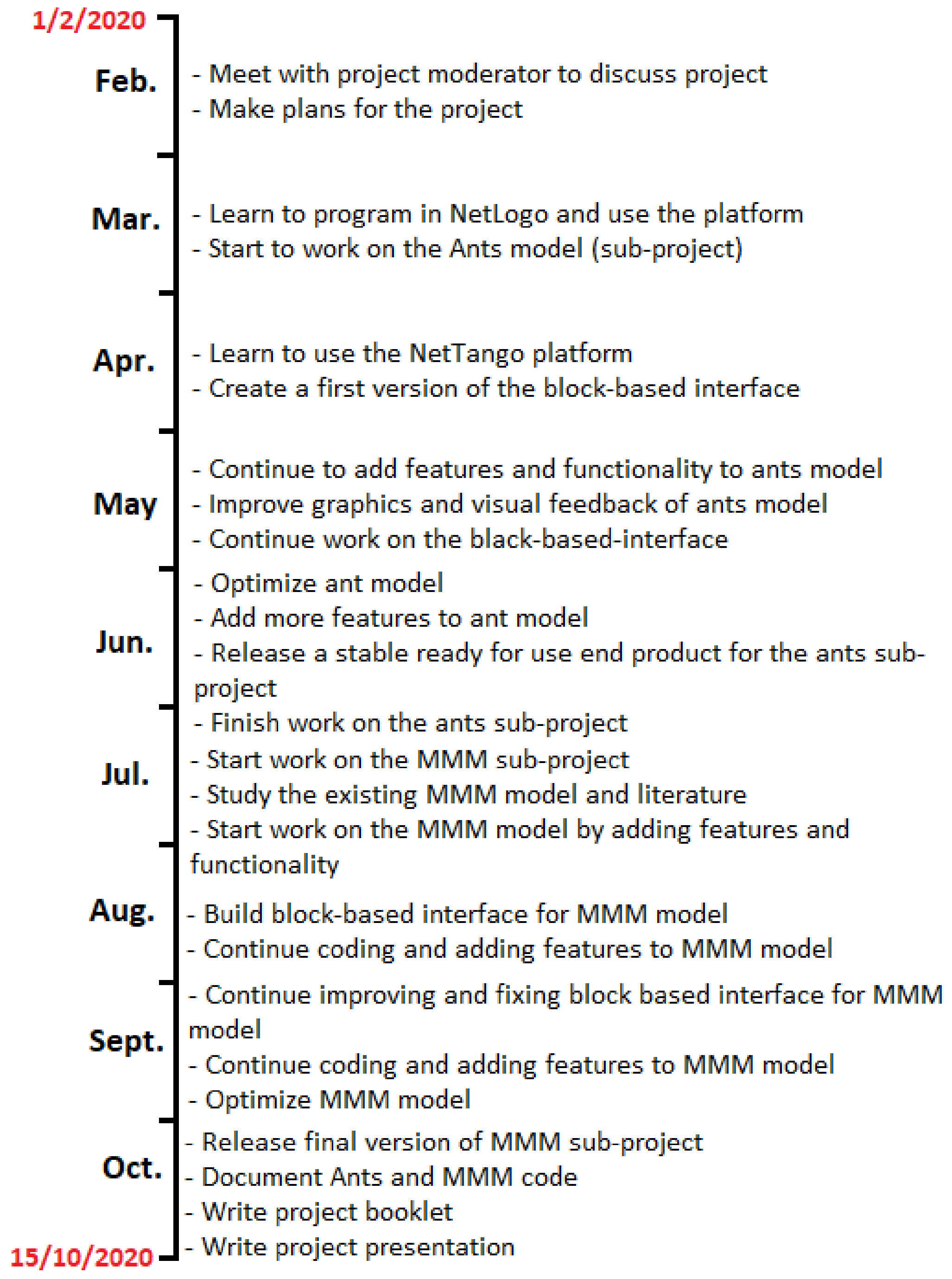
3.1. Flowchart


Define project objectives
Plan project model and block-based interface
Create computational model (NetLogo)
Create block-based interface (NetTango)
Integrate model with block-based interface (NetTango)
Export to HTML (end product)

This flowchart describes generally the process of creating an end product for each of the sub-projects.

Initially, the project objectives would be defined, and a plan made for how to implement them which was mostly a list of general features for both the model and block-based interface. The model and block-based interface can be developed independently and later integrated to work together. The end product is achieved by converting the integrated project in NetTango to an HTML file that users can run on their browser.

3.2. Schedule



1/2/2020

Feb.
- Meet with project moderator to discuss project
- Make plans for the project

Mar.
- Learn to program in NetLogo and use the platform
- Start to work on the Ants model (sub-project)

Apr.
- Learn to use the NetTango platform
- Create a first version of the block-based interface

May
- Continue to add features and functionality to ants model
- Improve graphics and visual feedback of ants model
- Continue work on the black-based-interface

Jun.
- Optimize ant model
- Add more features to ant model
- Release a stable ready for use end product for the ants sub-project

Jul.
- Finish work on the ants sub-project
- Start work on the MMM sub-project
- Study the existing MMM model and literature
- Start work on the MMM model by adding features and functionality

Aug.
- Build block-based interface for MMM model
- Continue coding and adding features to MMM model

Sept.
- Continue improving and fixing block based interface for MMM model
- Continue coding and adding features to MMM model
- Optimize MMM model

Oct.
- Release final version of MMM sub-project
- Document Ants and MMM code
- Write project booklet
- Write project presentation

15/10/2020

3.3. Environment

Hardware and software used in this project:

- Windows operating system
- NetLogo
- NetTango
- Chrome browser

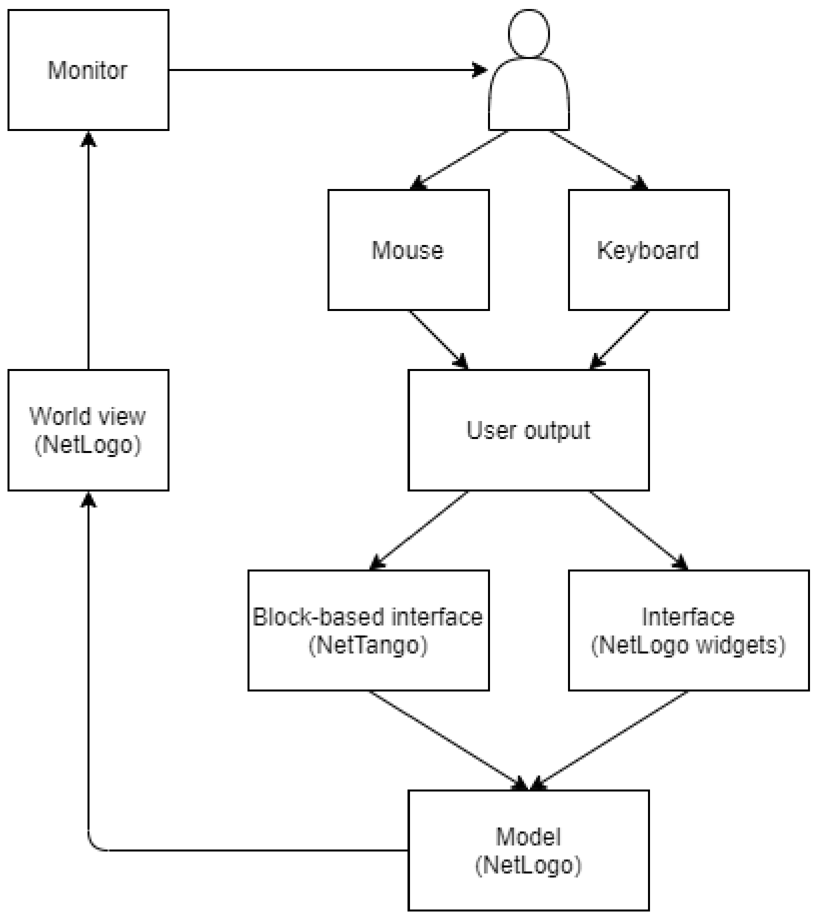
- Personal computer

3.4. Primary tasks

Listed in chronological order, though dependency is not necessarily in the same order.

1. Learn how to use the NetLogo platform and program in it.
2. Learn the scientific basis the computational model is modeled after.
3. Create a model in NetLogo
4. Learn to use the NetTango platform
5. Create a block-based interface for the modeling
6. Integrate the model with the block-based interface using NetTango
7. Document project and code.
8. Test the model.

4. User-Interaction



The users main form of interaction with the model is by interfacing with the models interface (NetLogo widgets) and by arrangement and parameter configuration of blocks in the block-based interface. By doing so the user can interact with the model and observe changes mainly through the world view displaying the current state of the model.

5. Ants

5.1. Introduction

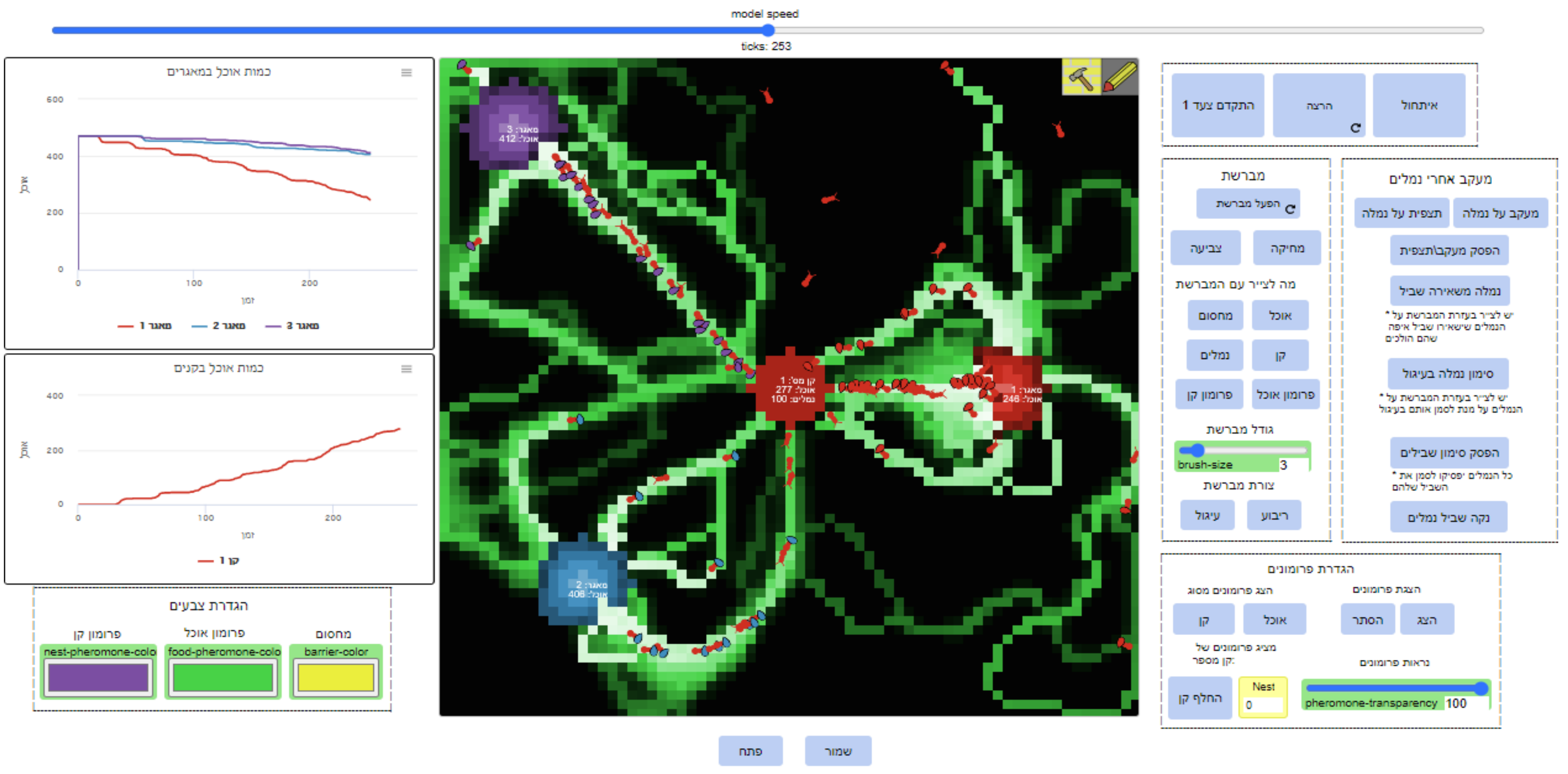
The Ants project models the complex system of pheromone communication used by an ant colony working together to scavenge for food and return it to their nests. This model allows the user to observe how pheromone communication is utilized by ants navigating through obstacles to from an ant trail to find the shortest path possible to a food source. The block-based interface enables the user to define the behavior of ants such as when to release pheromones, thus enabling the user to assess how phenomena emerged from ant colonies as a whole is correlated to the interactions of independent ants in the system.

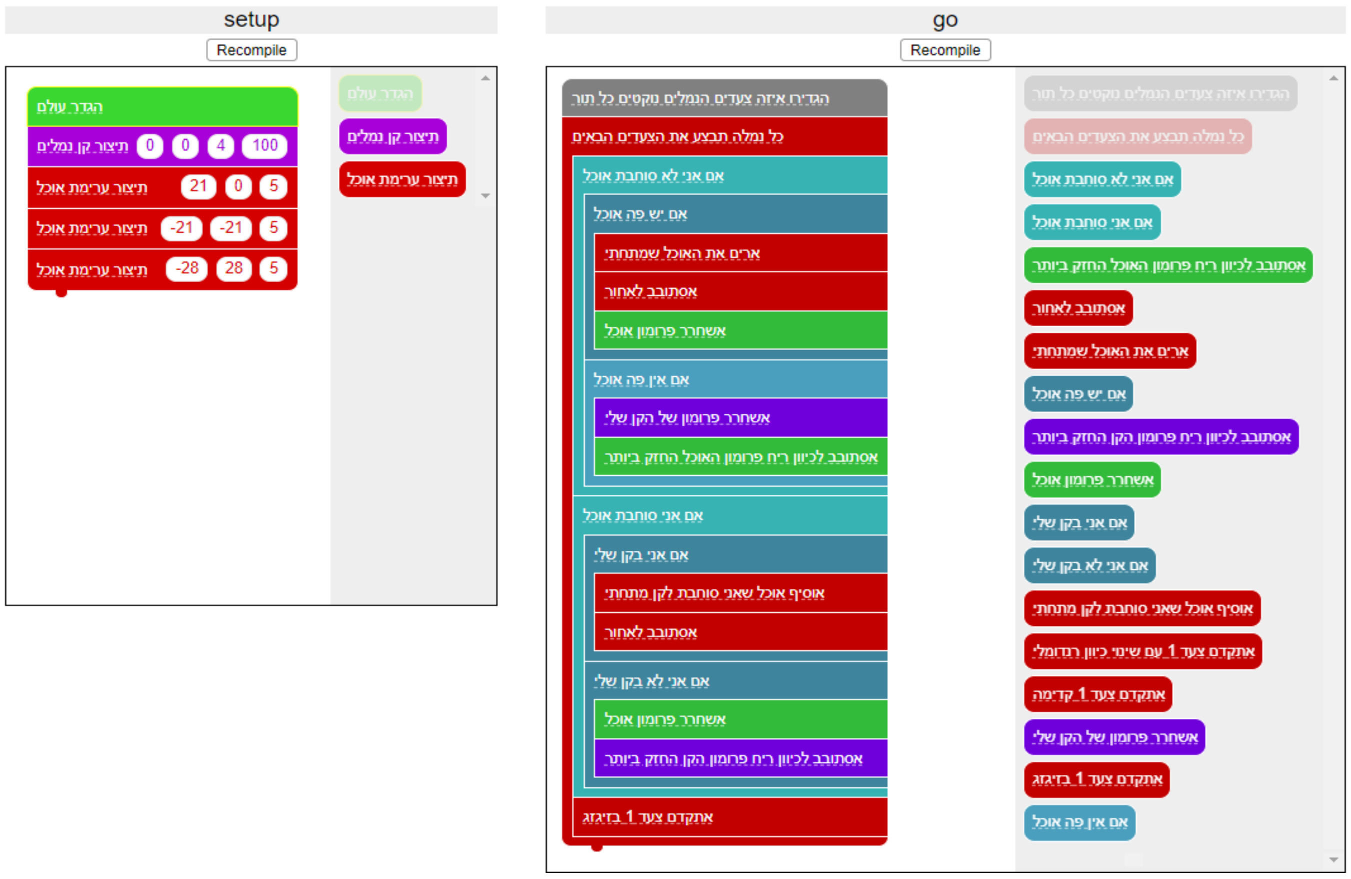
The project can be run in the browser by clicking on the following link (please use Chrome if the page does not load): https://ron-teller.github.io/ants-netlogo.github.io

5.2. Objective

The main objective is for the user to successfully configure the blocks defining ant behavior in a way that will affect the colony as a whole to successfully gather food. Doing so will let users observe how entities in a complex system interact with each other for the purpose of assessing their effects on the system as a whole. In this case, the user gets to explore how the behavior of a single ant (releasing pheromones for example) can affect the colony as a whole and observe forming patterns such as ant trails leading to a food source as a result of pheromone communication.

5.3. Overview





```
הגדר עולם
תיצור קן נמלים 0 0 4 100
תיצור ערימת אוכל 21 0 5
תיצור ערימת אוכל -21 -21 5
תיצור ערימת אוכל -28 28 5
```

```
הגדר עולם
תיצור קן נמלים
תיצור ערימת אוכל
```

```
הגדירו איזה צעדים הנמלים נוקטים כל תור
כל נמלה תבצע את הצעדים הבאים
    אם אני לא סוחבת אוכל
        אם יש פה אוכל
            ארים את האוכל שמתחתי
            אסתובב לאחור
            אשחרר פרומון אוכל
        אם אין פה אוכל
            אשחרר פרומון של הקן שלי
            אסתובב לכיוון ריח פרומון האוכל החזק ביותר
    אם אני סוחבת אוכל
        אם אני בקן שלי
            אוסיף אוכל שאני סוחבת לקן מתחתי
            אסתובב לאחור
        אם אני לא בקן שלי
            אשחרר פרומון אוכל
            אסתובב לכיוון ריח פרומון הקן החזק ביותר
    אתקדם צעד 1 בזיגזג
```

```
הגדירו איזה צעדים הנמלים נוקטים כל תור
כל נמלה תבצע את הצעדים הבאים
אם אני לא סוחבת אוכל
אם אני סוחבת אוכל
אסתובב לכיוון ריח פרומון האוכל החזק ביותר
אסתובב לאחור
ארים את האוכל שמתחתי
אם יש פה אוכל
אסתובב לכיוון ריח פרומון הקן החזק ביותר
אשחרר פרומון אוכל
אם אני בקן שלי
אם אני לא בקן שלי
אוסיף אוכל שאני סוחבת לקן מתחתי
אתקדם צעד 1 עם שינוי כיוון רנדומלי
אתקדם צעד 1 קדימה
אשחרר פרומון של הקן שלי
אתקדם צעד 1 בזיגזג
אם אין פה אוכל
```

The Ants project is comprised of two main components – the model (top) and block-based interface (bottom).

5.3.1. Model

The model’s main features are:

- Simulate a computational model of a complex system of ant colonies utilizing pheromone communication to scavenge for food and return it to their nests.
- Drawing and erasing entities in the world, such as ants, food, nests, walls (obstacles the ants can not go through) and pheromones with the use of brush.
- Create numerous ant colonies that compete for food.
- Run and stop the model, either continuously or discretely (advancing 1 tick at a time).
- Tracking ants by tracing their trail or marking them with a circle.
- Monitoring amount of food amounted in nests and food in piles, also shown in plotted graphs (left side of model)
- Customize colors of entities in the world
- Display the pheromones in the world. In the example above, the food pheromone is displayed as green colored patches. You may notice that the ants have formed ant trails on paths that have the strongest pheromone smell (stronger smell is displayed as a brighter color).
- Configurable display of pheromones in the model. The user can choose to display either the nest or food pheromone of any specific nest, and choose the visibility of it.

5.3.2. Block-based interface

The block-based interface is comprised of two block spaces named “setup” (left) and “go” (right).

- “Setup” – used for initializing world state. Every time the model is reset, the world is reset to the state configured by the blocks in this block space. In the example above, when a model is reset it will create an ant colony of 100 ants and add 3 different food sources in the given coordinates.
- “Go” – used for defining ant behavior. There is a block for each of the basic actions an ant can make or states it can be in. The user can arrange these blocks to configure what actions an ant takes every step.

The aim is for the user to configure the blocks defining the behavior of a single ant in a way that will affect the colony as a whole to successfully achieve food-gathering.

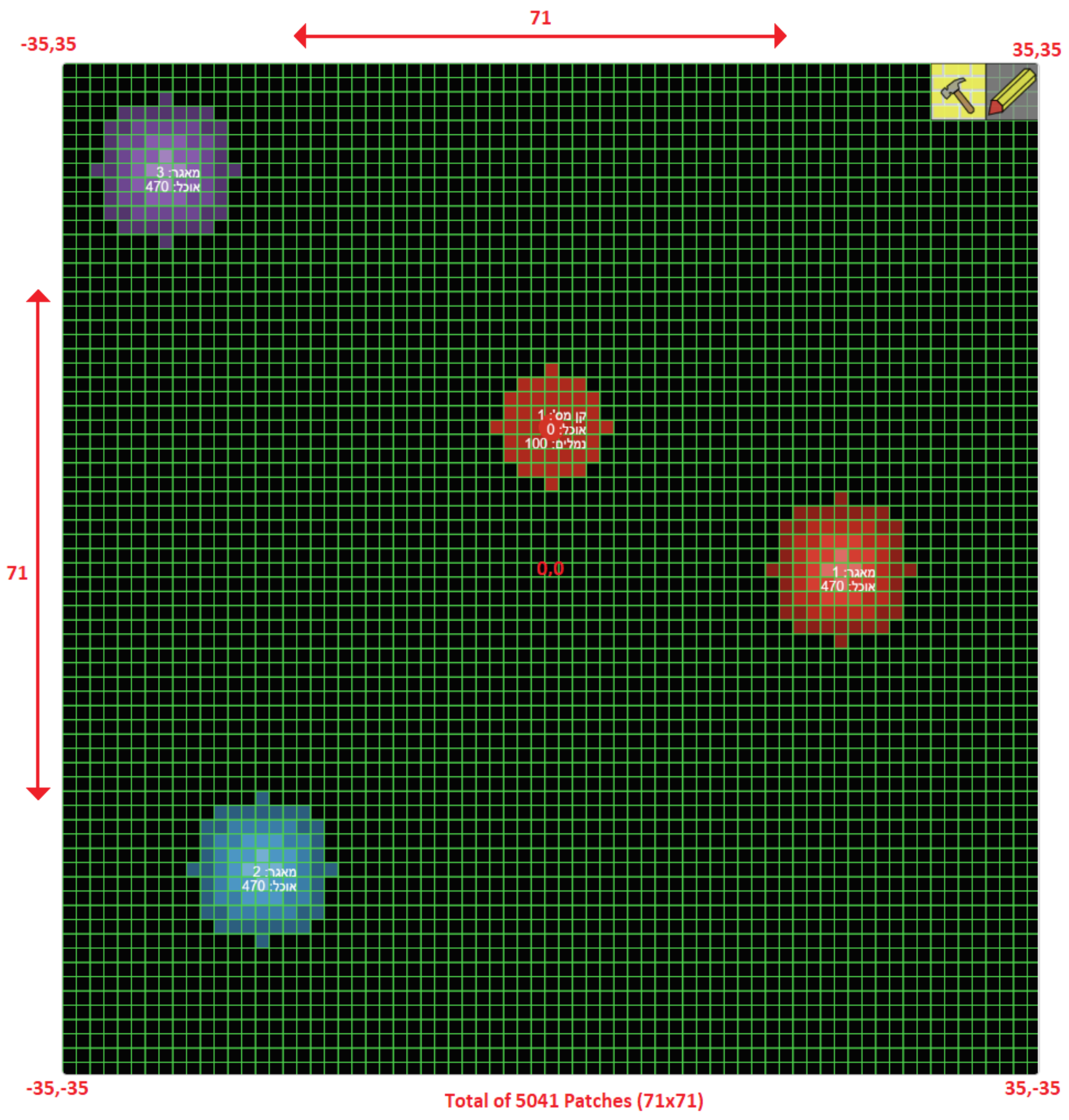
5.4. World Entities & Components

The Ants model simulates a world of several entities and components that make up the complex system. This section described all existing entities and components and how they interact with each other.

5.4.1.World Space

All entities in the ants model exist inside a bounded 2D space. Inherent to all models in NetLogo, the 2D space is sectioned into a grid, and each cell in the grid is referred to as a “patch”. All entities must move and exist within the bounded 2D space.

The size of the world is set to 71×71, which means that the world grid contains 71 rows and 71 columns. The center patch of the world is located at coordinate (0,0). The world contains a total of 5041 patches. The model can easily be configured to be smaller or bigger in the NetLogo IDE.



5.4.2. Ants

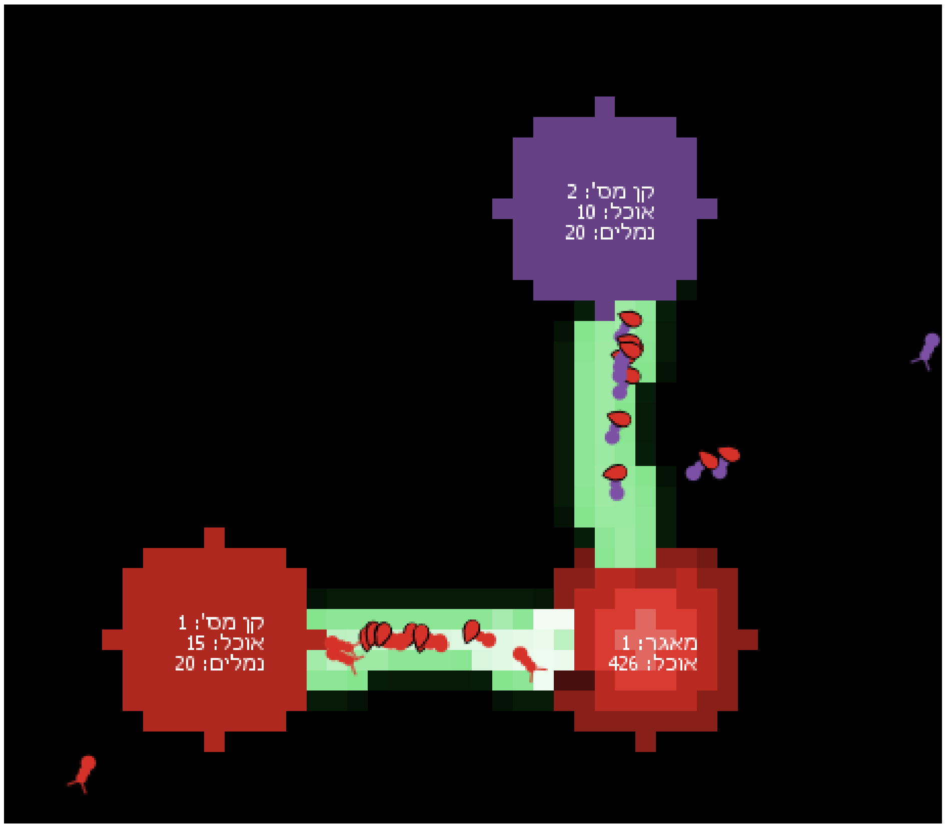
Ants are the main acting entities in this model. An ants purpose is to find food and bring it back to the nest. Ants have a defined set of properties and actions they can perform that define their possible behavior.

5.4.2.1. *Properties*

- Each ant belongs to a certain nest.
- Ants that belong to the same nest make up a colony.
- Ants can only bring food to their own nest.
- Ants can only smell pheromones emitted by other ants of the same colony.
- Ants can not smell pheromones through walls.
- Ants are blocked by walls and world boundaries.
- When blocked, the ant will change direction to closest unblocked neighboring patch.
- Ants are colored the same color of their nest.

5.4.2.2. *Actions*

- Move forward
- Pick up food
- Drop food in nest
- Release food pheromone
- Release nest pheromone
- Sniff pheromones
- Change direction to strongest pheromone smell

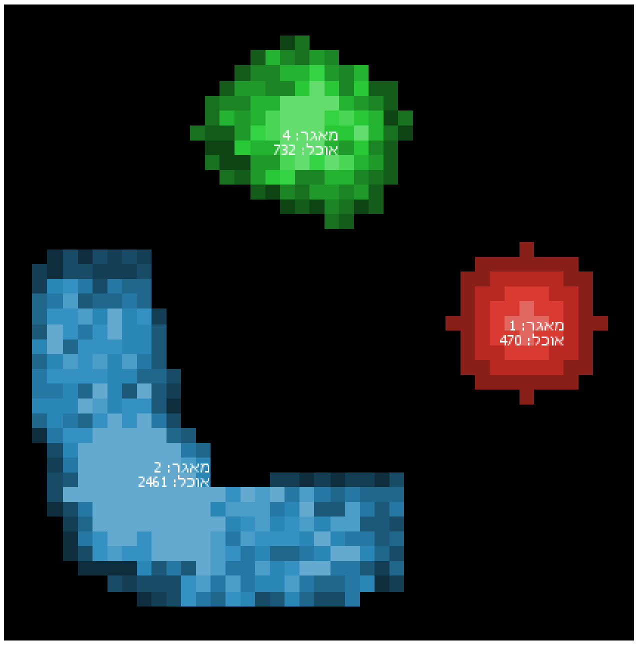


5.4.3. Nests

- Nests serve as a place for colonies to store food.
- Each colony of ants has its own nest which only they can store food in.
- Each nest has its own unique number and color to distinguish it from other nests.
- A nest occupies a whole patch, and there can not be more than 1 nest occupying a patch.
- An ant is considered to be standing on a nest if its coordinates are anywhere inside a patch occupied by a nest. A nest can spread out over as many patches possible.
- A nest can not be created in a patch that is already occupied by another nest or contains a wall and vice versa.
- Each nest displays information showing the nests unique number, the amount of ants in the colony and the amount of food amassed in the nest.
- Ants that belong to the nest are a colored a brighter shade of the nest.

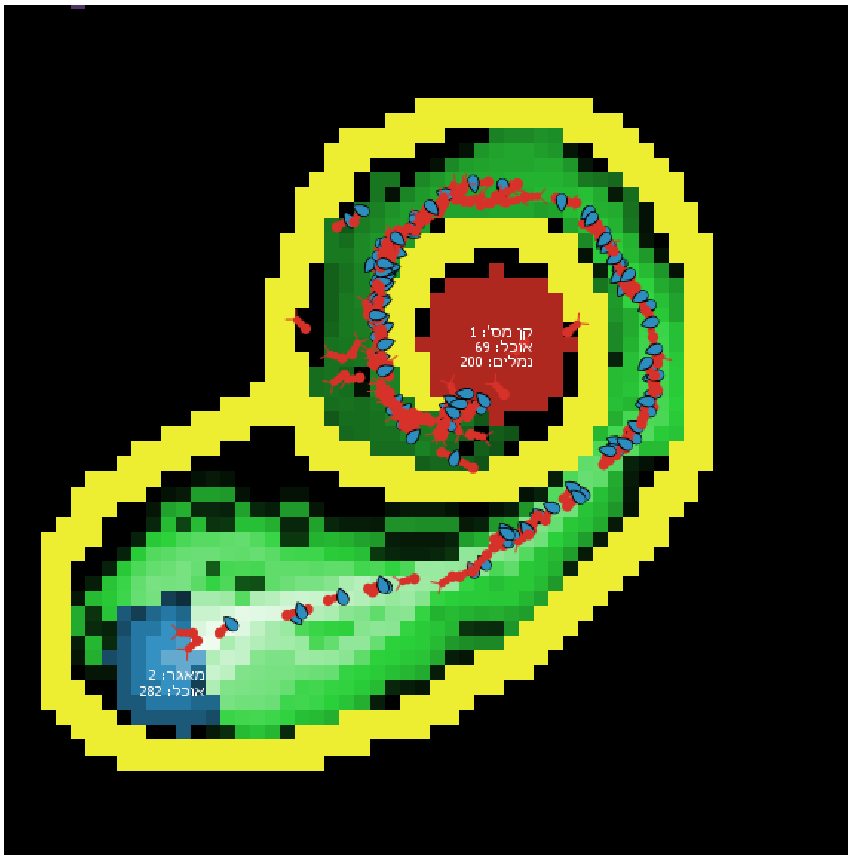
5.4.4. Food

- Food can be picked up by ants and be carried back to their nest. Ants can only pick up 1 food at a time.
- Each food belongs to a food source.
- Each food source is assigned its own distinct number and color to distinguish it from other food sources.
- Food is contained inside a patch, and there can not be more than 1 food source inside a patch.
- A patch can contain an unlimited amount of food
- An ant can pick up food if its coordinates are anywhere inside a patch containing food.
- Food can not be placed on a patch that is occupied by a wall and vice versa.
- Each food source displays information showing it’s distinct number and amount of food.
- A brighter shade of color on a food patch indicates more food in the patch.



5.4.5. Walls

- Walls obstruct an ants movement. An ant can not move through a wall, it must find a way around it.
- A wall occupies a whole patch. Ants can not pass through a patch occupied by a wall.
- Ants can not smell pheromones through walls.
- Pheromones can not pass through walls when they diffuse and spread out through the world.
- Walls can not be created in patches that are occupied by nests, contain food or are occupied by ants that can not be pushed aside to an unblocked patch.



5.4.6. Food Pheromone

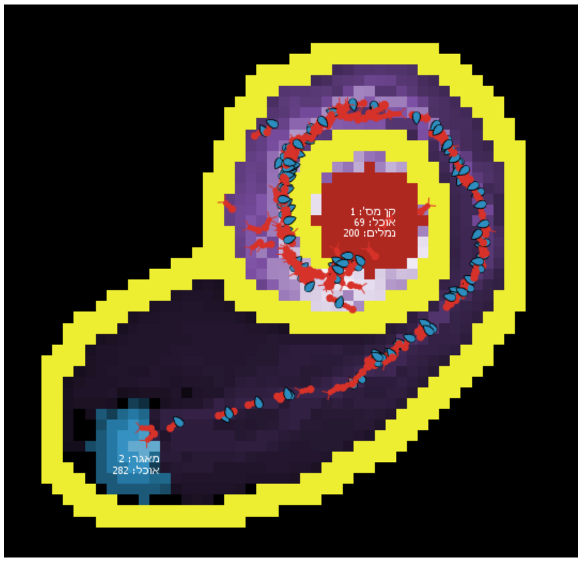
- Ants release food pheromones to guide other ants to food.
- Ants can only add pheromones to the patch they are standing on.
- Each patch contains a number value for the concentration (strength) of food pheromones present in it. A higher value means a higher concentration of pheromone.
- A patch with a higher food pheromone value should indicate it is closer to a food source than a patch with a lower pheromone value.
- Ants can only sense pheromones released by their own colony
- Ants can sniff patches in front of them and change direction to the patch with the highest food pheromone value.
- The amount of pheromone an ant releases is based on the Dijkstra algorithm:

    - When an ant encounters a food source, it will set itself to release the maximum amount of pheromone.
    - Every time an ant moves it decreases the amount of pheromone to be released. This serves as a form of measurement to how far an ant is away from the food source it found.
    - If the ant is standing on a patch that has a higher pheromone consistency than that of the amount it is set to release, it will set itself to release the higher pheromone value in the patch it is on.
    - This behavior allows the ants to map out paths to food sources. The ants follow the pheromone trails heading in the direction of the strongest pheromone they sense.
- Every turn pheromones spread out and diffuse in the world.
- Every turn pheromones evaporate slightly.

5.4.7. Nest Pheromone

* Both nest and food pheromones function in very much the same way, the only real difference being that a nest pheromone is used to guide ants to the nest and food pheromones guide ants to a food source.

- Ants release nest pheromones to find their way and guide other colony ants back to their nest.
- Ants can only add pheromones to the patch they are standing on.
- Each patch contains a number value for the concentration (strength) of nest pheromones present in it. A higher value means a higher concentration of pheromone.
- A patch with a higher nest pheromone value should indicate it is closer to a nest source than a patch with a lower pheromone value.
- Ants can only sense pheromones released by their own colony
- Ants can sniff patches in front of them and change direction to the patch with the highest nest pheromone value.
- The amount of pheromone an ant releases is based on the Dijkstra algorithm:
    - When an ant encounters a nest, it will set itself to release the maximum amount of pheromone.
    - Every time an ant moves it decreases the amount of pheromone to be released. This serves as a form of measurement to how far an ant is away from its nest.
    - If the ant is standing on a patch that has a higher pheromone consistency than that of the amount it is set to release, it will set itself to release the higher pheromone value in the patch it is on.
    - This behavior allows the ants to map out paths to their nest. The ants follow the pheromone trails heading in the direction of the strongest pheromone they sense.
- Every turn nest pheromones spread out and diffuse in the world.
- Every turn nest pheromones evaporate slightly.

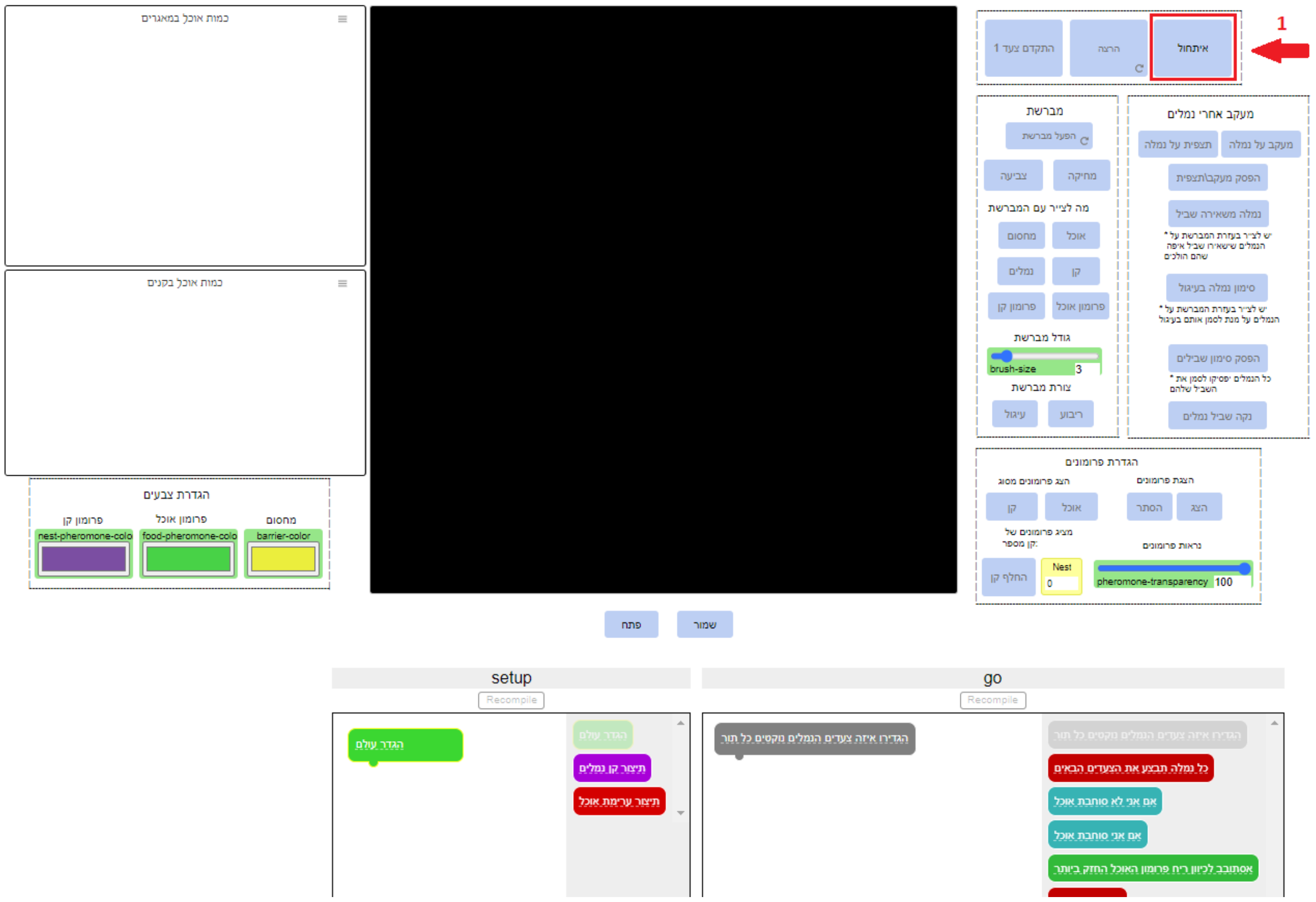


**It can be observed that the nest pheromones (purple) are weaker in patches further away from the nest, and patches closer to the nest have a stronger pheromone scent.*

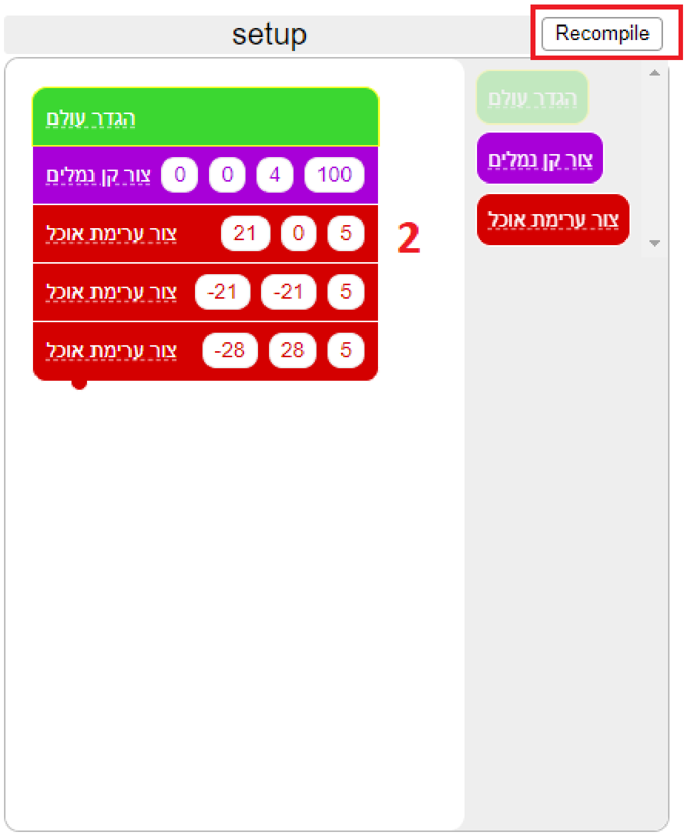
5.5. Example

This chapter showcases the use of this model from the users perspective.

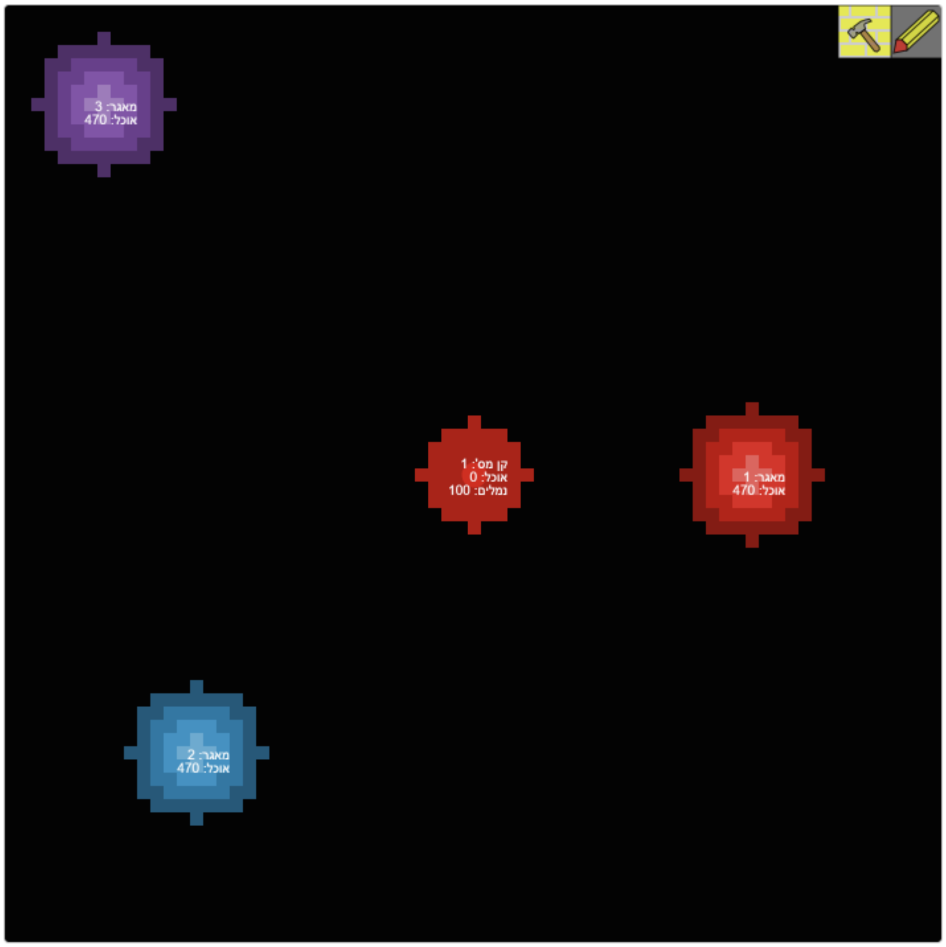
1. The project is opened up in a browser and an uninitialized model appears. The *initialize* world button is the only button that is clickable, let's click on that and see what happens!



2. Hmm, nothing seems to have happened. The world view is still empty, i don't see any ants or nests or anything. I guess i need to create the world myself. Looking at the bottom i see a *setup* block space, it seems i can create a world by arranging some of those blocks. Let's add some blocks to create an ant colony and some food sources. After i have added the blocks i see i need to press the *recompile* button on the top right for the changes to take affect so lets go right ahead and do that too.

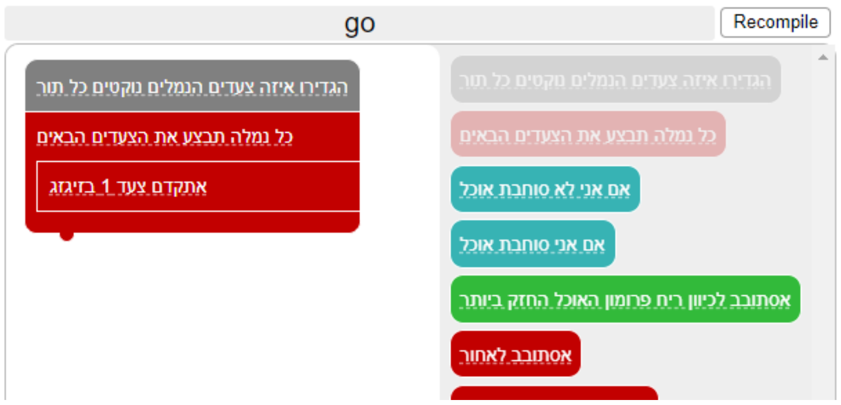


3. The world view is still empty. The blocks only defined that happens when a model is initialized, so lets press the *initialize* button again and see what happens. Viola! Our ant colony and food has been created.



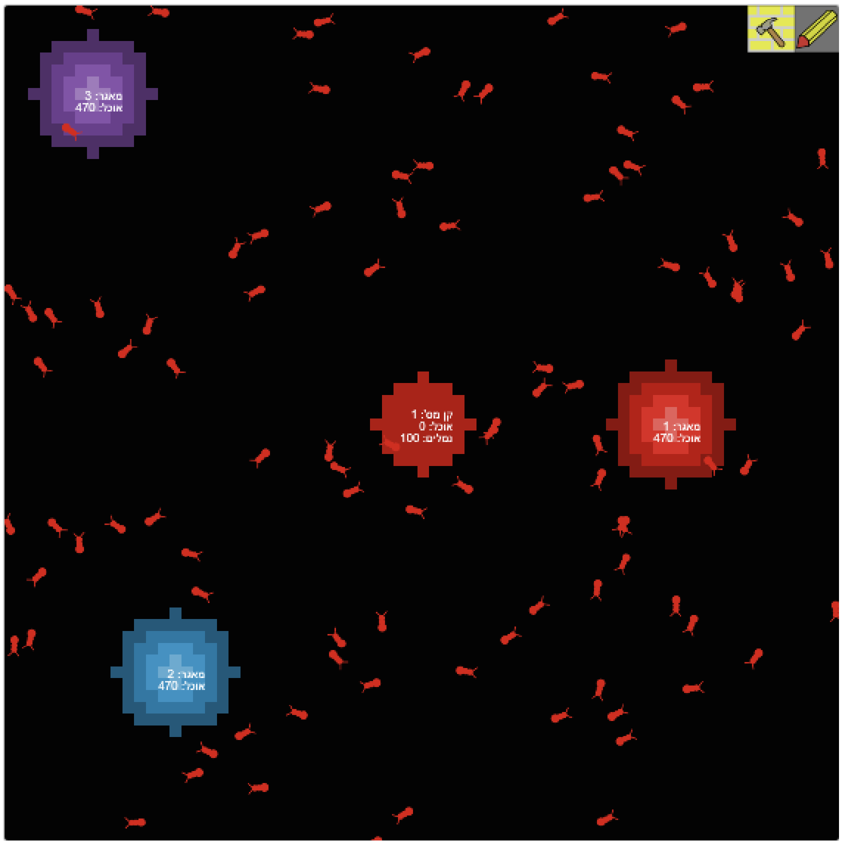
4. Now lets run the model by pressing *Play*. Nothing! the ants aren't moving, nothing is happening. Of-course nothing is happening, we haven't defined an ants behavior in the block interface.

5. Lets make the ants go forward and see if that changes anything. Lets drag some blocks to make ants move forward in a zig-zag, and recompile since we made a block change.

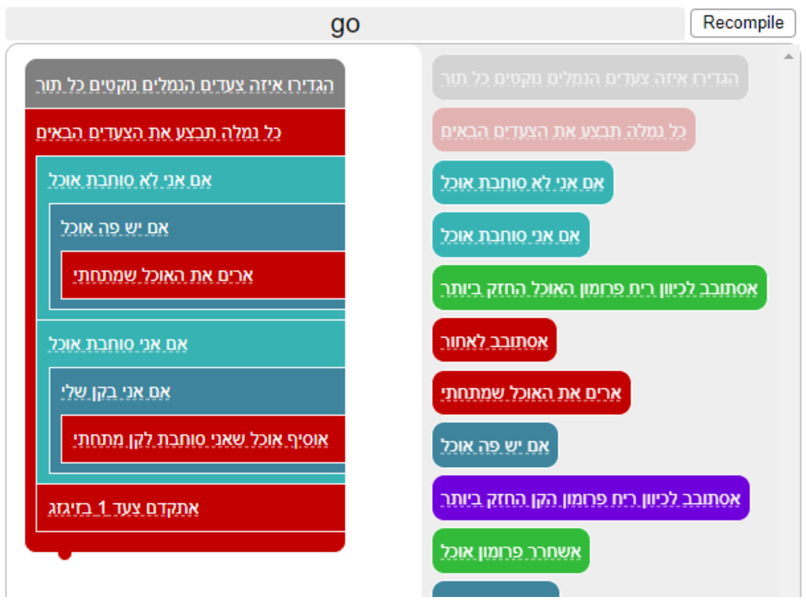

go
Recompile
הגדירו איזה צעדים הנמלים נוקטים כל תור
כל נמלה תבצע את הצעדים הבאים
אם אני לא סוחבת אוכל
אם אני סוחבת אוכל
אסתובב לכיוון ריח פרומון האוכל החזק ביותר
אסתובב לאחור

```
הגדירו איזה צעדים הנמלים נוקטים כל תור
כל נמלה תבצע את הצעדים הבאים
    אתקדם צעד 1 בזיגזג
```

6. Now let's press play and see what happens. As expected, the ants have started moving. However, they aren't doing anything BUT moving. Guess we will take care of that next.



7. Let's make the ants pick up food if it is empty, and drop food at nest if carrying food.

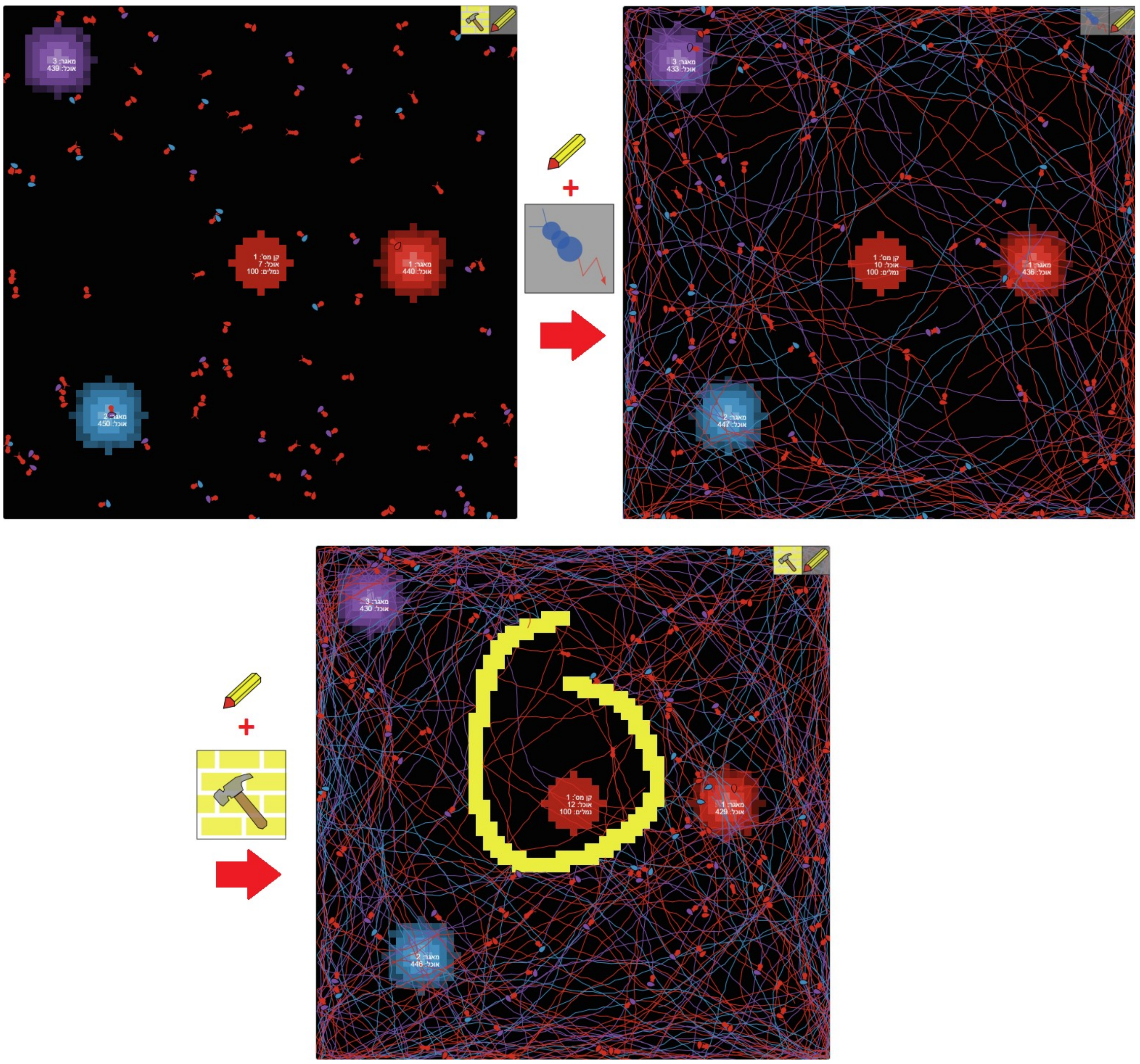


```
הגדירו איזה צעדים הנמלים נוקטים כל תור
כל נמלה תבצע את הצעדים הבאים
  אם אני לא סוחבת אוכל
    אם יש פה אוכל
      ארים את האוכל שמתחתי
  אם אני סוחבת אוכל
    אם אני בקן שלי
      אוסיף אוכל שאני סוחבת לקן מתחתי
  אתקדם צעד 1 בזיגזג
```

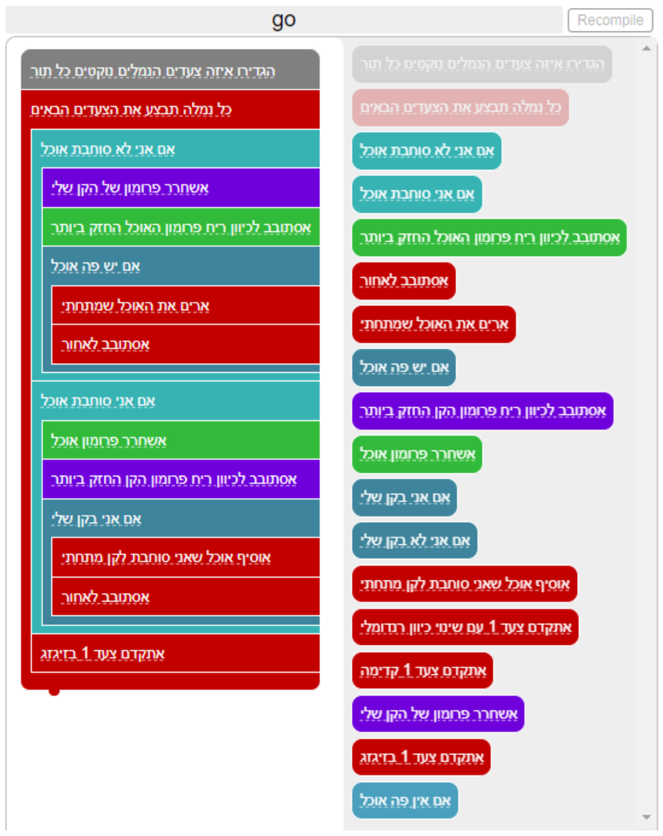
```
הגדירו איזה צעדים הנמלים נוקטים כל תור
כל נמלה תבצע את הצעדים הבאים
אם אני לא סוחבת אוכל
אם אני סוחבת אוכל
אסתובב לכיוון ריח פרומון האוכל החזק ביותר
אסתובב לאחור
ארים את האוכל שמתחתי
אם יש פה אוכל
אסתובב לכיוון ריח פרומון הקן החזק ביותר
אשחרר פרומון אוכל
```

8. Now let's play and see if the ants manage to bring the food home. Alas, that ants are a mess, there is no order! They seem to be randomly walk around the world, and if they happen to walk on food, they pick it up, and if by so chance they find their way back to the nest, they drop the food. There are no ant trails formed, no teamwork. Each ant just randomly walks around. We use the brush to select ants to trace, and we can clearly see their paths are random.

Furthermore, if we add a wall, we can clearly see that the ants can not really make it at all to the nest, they aren't any paths leading to it.



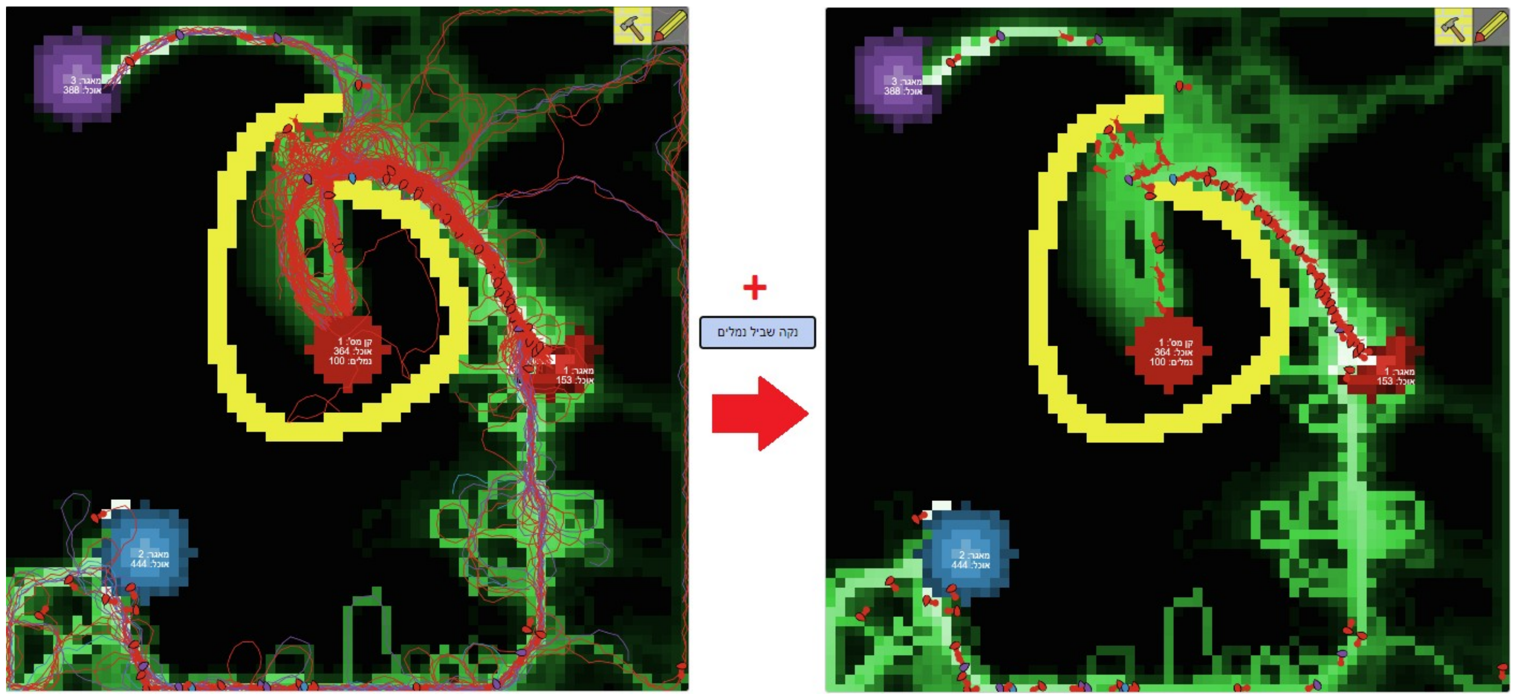
9. Pheromones must be the answer! by forming pheromone trails, the ants can lead each other to food sources and back to their nest. Let's change the ants behavior in the blocks interface to release a nest and food pheromone. Also, the ants should be programmed to also smell the pheromones and move in their direction.



```
הגדירו איזה צעדים הנמלים נוקטים כל תור
כל נמלה תבצע את הצעדים הבאים
    אם אני לא סוחבת אוכל
        אשחרר פרומון של הקן שלי
        אסתובב לכיוון ריח פרומון האוכל החזק ביותר
        אם יש פה אוכל
            ארים את האוכל שמתחתי
            אסתובב לאחור
    אם אני סוחבת אוכל
        אשחרר פרומון אוכל
        אסתובב לכיוון ריח פרומון הקן החזק ביותר
        אם אני בקן שלי
            אוסיף אוכל שאני סוחבת לקן מתחתי
            אסתובב לאחור
    אתקדם צעד 1 בזיגזג
```

```
הגדירו איזה צעדים הנמלים נוקטים כל תור
כל נמלה תבצע את הצעדים הבאים
אם אני לא סוחבת אוכל
אם אני סוחבת אוכל
אסתובב לכיוון ריח פרומון האוכל החזק ביותר
אסתובב לאחור
ארים את האוכל שמתחתי
אם יש פה אוכל
אסתובב לכיוון ריח פרומון הקן החזק ביותר
אשחרר פרומון אוכל
אם אני בקן שלי
אם אני לא בקן שלי
אוסיף אוכל שאני סוחבת לקן מתחתי
אתקדם צעד 1 עם שינוי כיוון רנדומלי
אתקדם צעד 1 קדימה
אשחרר פרומון של הקן שלי
אתקדם צעד 1 בזיגזג
אם אין פה אוכל
```

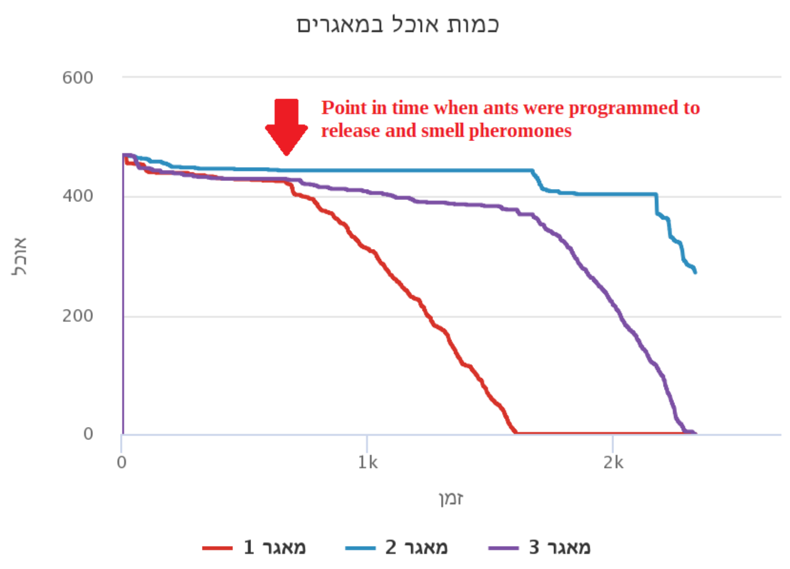
10. Now lets continue playing the model and check how the ants will behave now, will they be able to find their way to a food source and back to the nest, even with a wall around their nest?



11. Success! the ants successfully form pheromone tails to navigate their way around obstacles to a food source and back to their nest. To get a better sense of how the pheromones helped the ant colony successfully gather food, we can take a look at one of the plots that monitors the amount of food in

sources. We can see a sharp decline in food resources just when the ants were programmed to utilize their pheromones. Before that, food gathering was very slow.

This sums up the example. There are still more features not explored in this example, but are documented in the User Guide chapter.

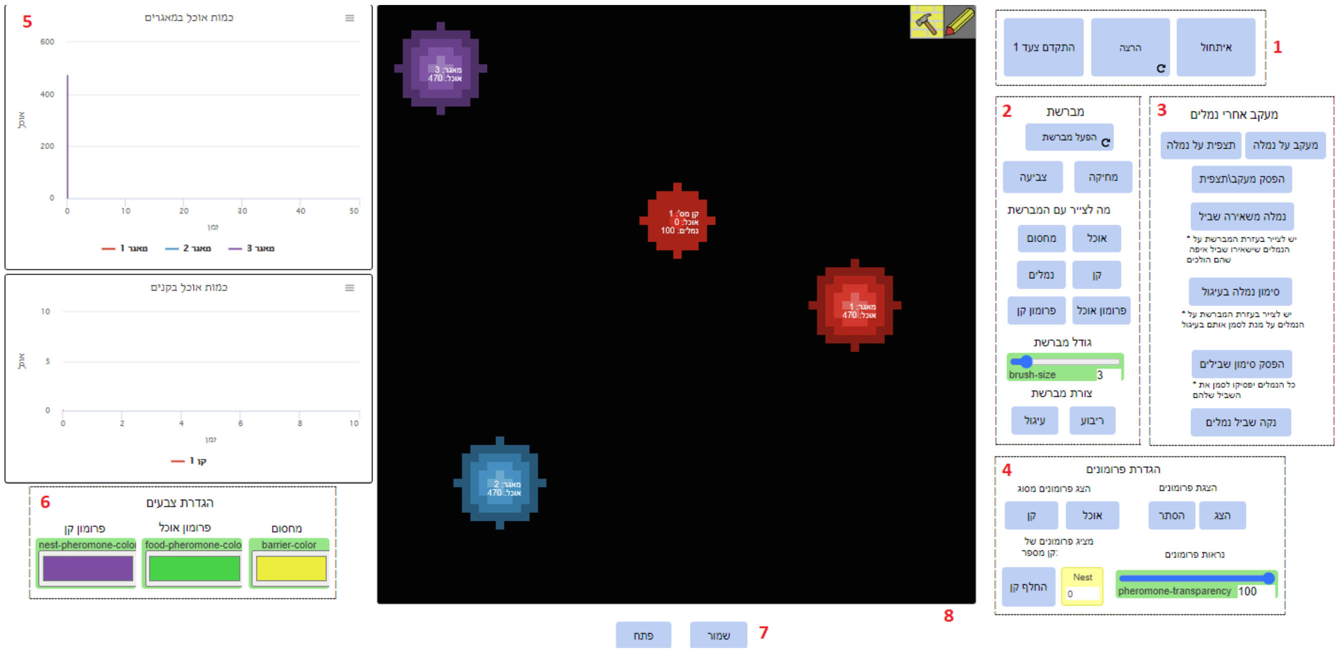


5.6. User Guide

This chapter is a guide to using all functionalities offered in this project.

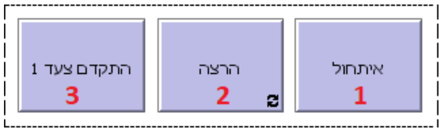
5.6.1. Model interface sections

The model interface is organized in sections that group common widgets together. The sections in the picture below are numbered.



1. Start/Stop model – buttons to control model execution
2. Brush – contains widgets that configure the brush for drawing/erasing entities in the world.
3. Tracking – numerous options that help tracking the movement of ants
4. Pheromone display configuration – configure setting regarding visual preferences of pheromones
5. Plots – contains 2 plots for monitoring food in world and food in nest
6. Color configuration – choose wall and pheromone color
7. Save/Load – save or load a world state
8. World view

5.6.2. Start/Stop model



5.6.2.1. Initialize

(1) Resets the world to an initialized predefined state configured in the block-based-interface.

5.6.2.2. Run/Pause

(2) Run/stop execution of model

5.6.2.3. Advance single turn

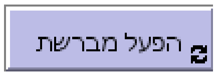
(3) Advance model a single turn

5.6.3. Brush

The brush can be used to add/remove entities in the model. It is used like a brush, by “painting” with it in the world view. The user may configure the brush’s shape and size.

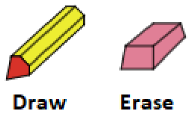
5.6.3.1. Activate brush

In order to use the brush it must be activated first. While the brush is activated its cursor will be displayed in the world view and will be usable for painting.



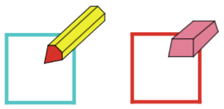
5.6.3.2. Cursor

The brush cursor is displayed in the world view as either a pencil drawing or an eraser erasing.



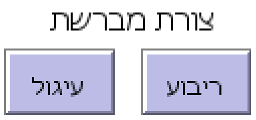
5.6.3.3. Outline

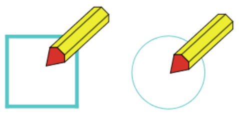
The brush outline encompasses the patches that the brush is painting on. It is colored cyan if it is drawing and red if erasing.



5.6.3.4. Shape

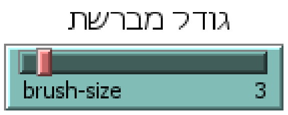
The brush shape can be configured to be either a square or circle. The brush’s shape affects which patches get painted on.

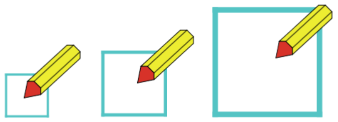




5.6.3.5. Size

The brush size can be configured to paint on a bigger or smaller area.





5.6.3.6. Draw

Set the brush to draw

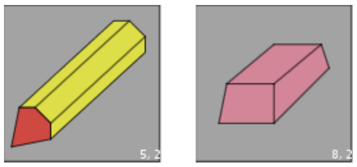
5.6.3.7. Erase

Set the brush to erase

5.6.3.8. Icons

Brush icons are displayed in the top right corner of the world view that indicate what the brush is currently painting, and if the brush is set to draw or erase.

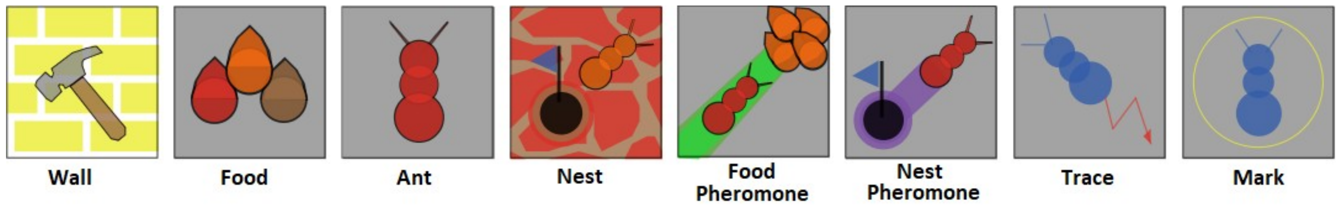
5.6.3.8.1. Draw/Erase icon



Indicates if brush is set to draw or erase. In the bottom right corner of the icon are the brush coordinates.

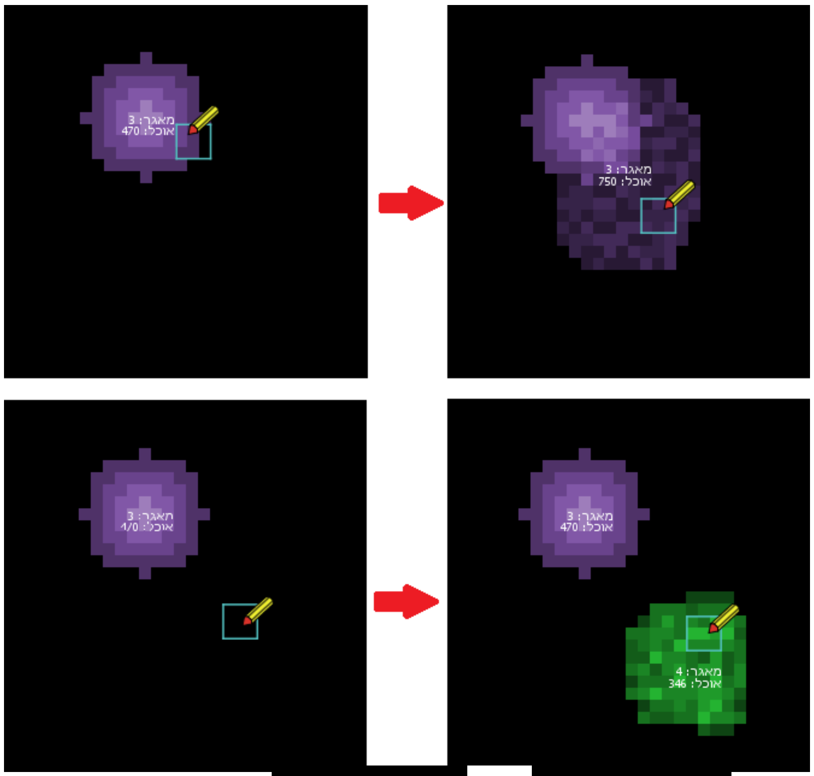
5.6.3.8.2. Brush type icon

Indicates what the brush is painting.

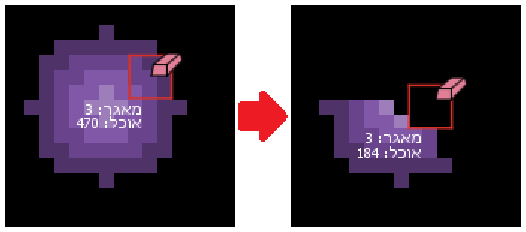


5.6.3.9. Add/Remove Food

Food can be added or removed using the brush. When adding food, if the brush was hovering over an existing food source when first clicked, it will add food belonging to that same food source. Otherwise it will create a new food source (the new food source number will be increased by 1). The amount of food added per patch is randomly chosen between 1 to 3. Erasing food removes all food the brush is painting on.



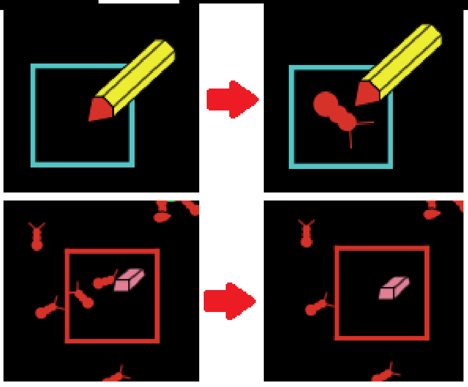
**Adding food*



Removing food

5.6.3.10. Add/Remove Ant

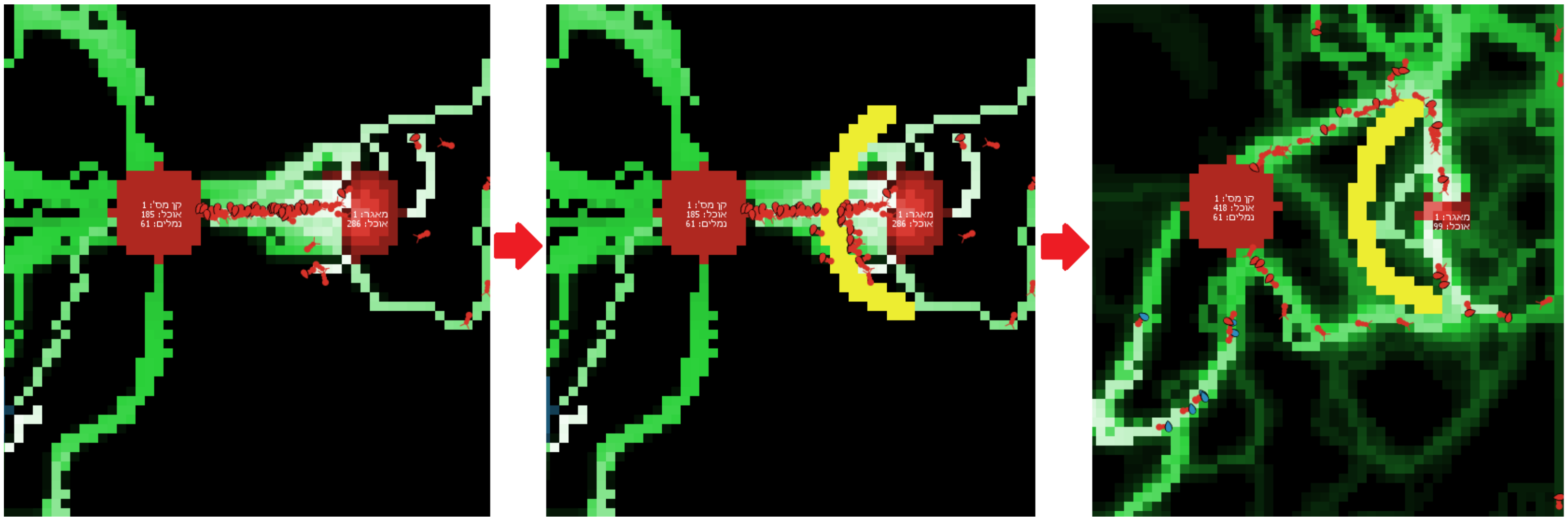
Ants can be added or removed using the brush. When adding ants, clicking anywhere inside a nest will add ants of that colony. Adding ants anywhere else in the world will default to adding ants belonging to the colony with the lowest nest number. The amount of ants added is equivalent to the brush size.



5.6.3.11. Add/Remove Wall

Walls can be created or removed using the brush. Walls enable the user to observe how ants use pheromone communication to navigate around them. An ant may be forced to find a new path if it is blocked by a newly placed wall, or discover a shorter path if a wall was removed. Users can add and remove walls dynamically as the model is running.

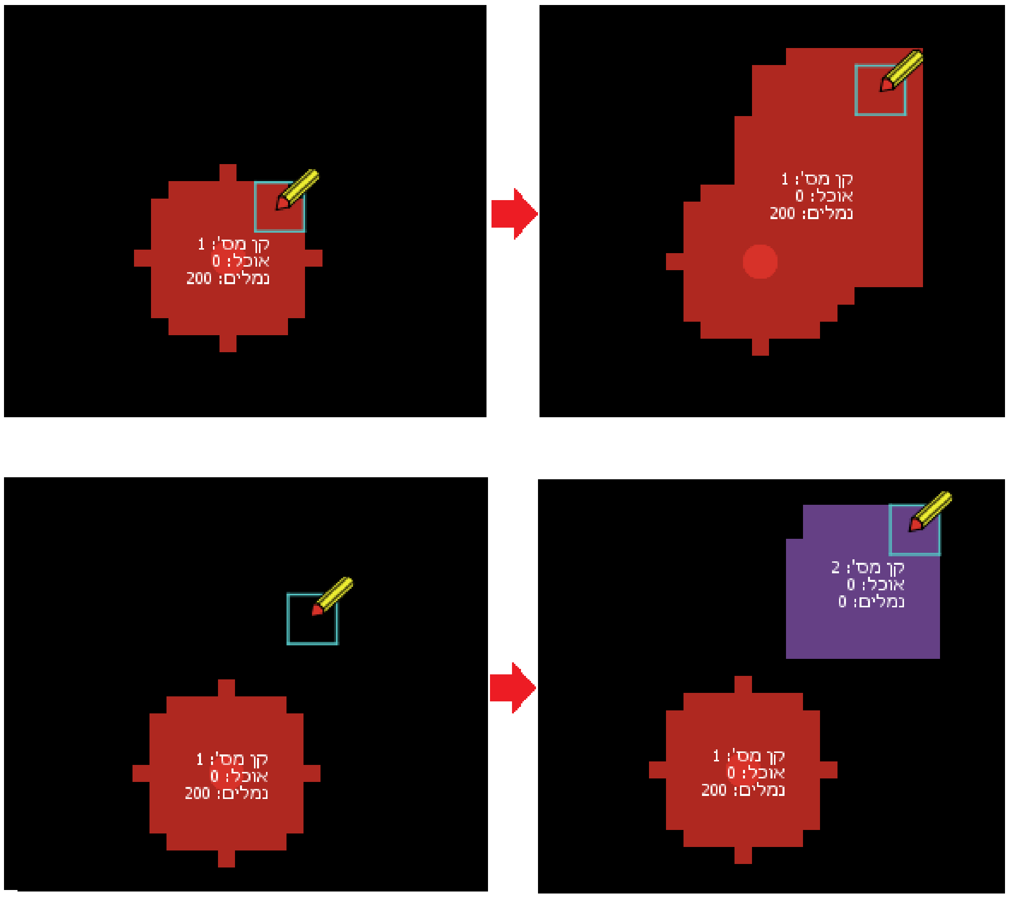
A wall can not be created in a patch that is occupied by a nest or food. If a patch is occupied by ants then a wall can be added only if ants can be pushed aside to unblocked neighboring patches.



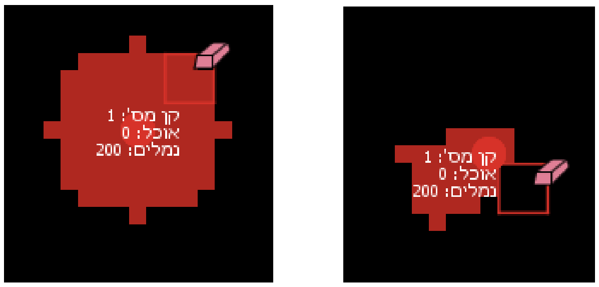
5.6.3.12. Add/Remove Nest

Nests can be added and removed using the brush. A new colony can be created by creating a new nest and adding ants to that nest. When creating nests, if the brush was hovering over an existing nest when first clicked, it will expand that nest. Otherwise it will create a new nest (the new nest number will be increased by 1).

Adding nest

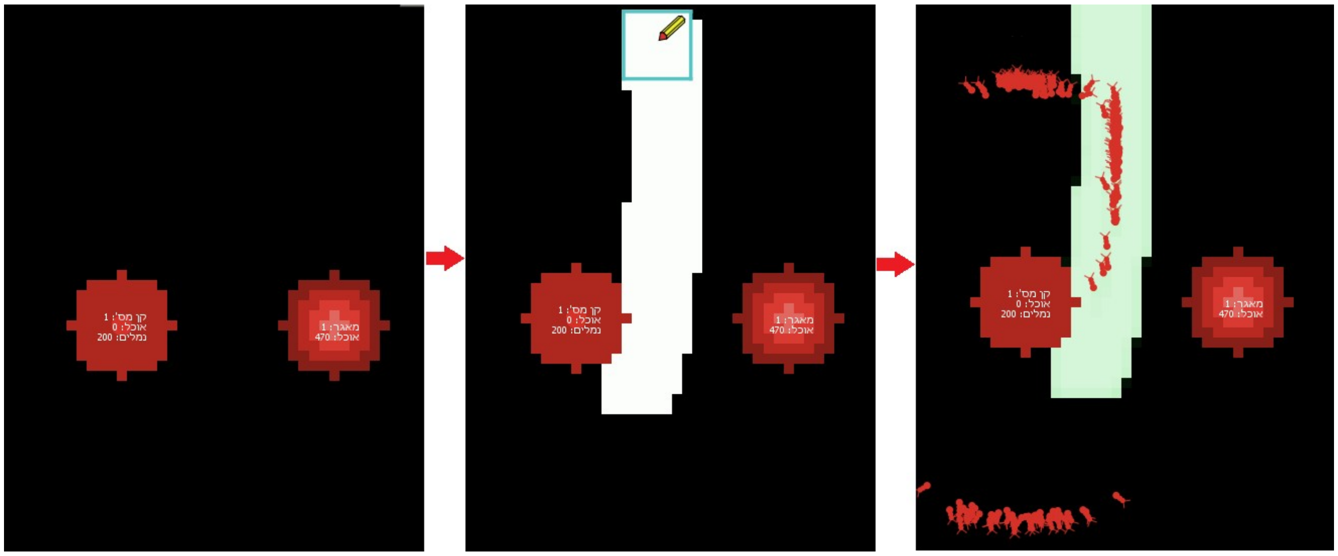


Removing nest



5.6.3.13. *Add/Remove Food pheromone*

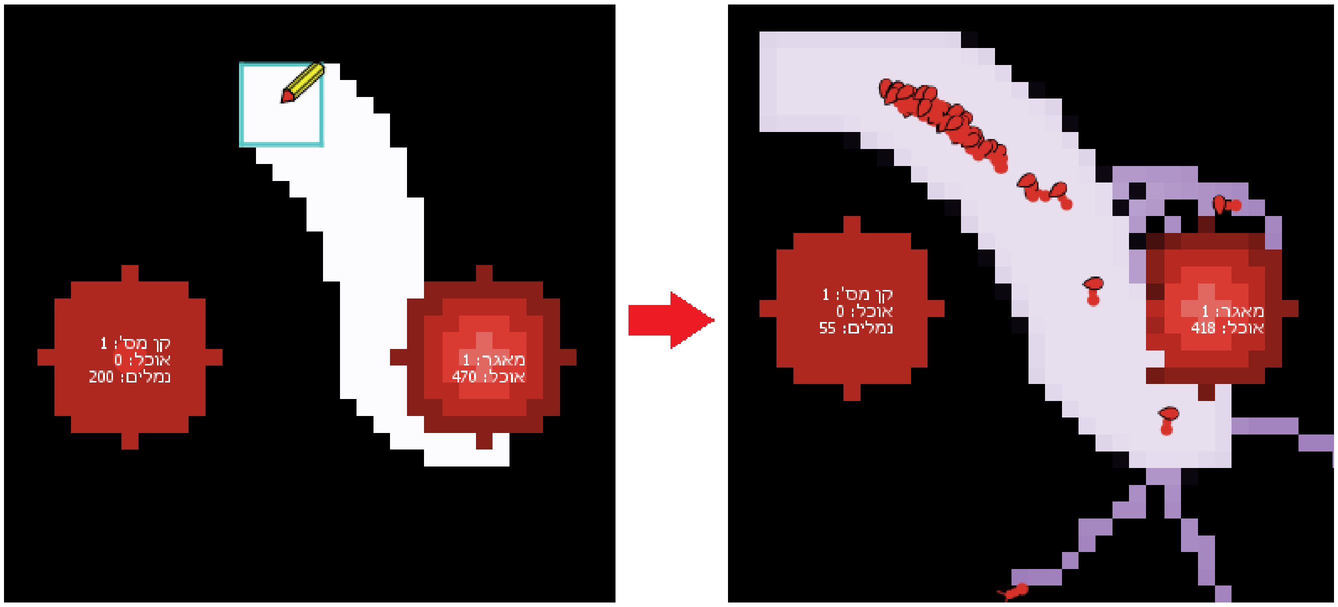
Food pheromone can be “artificially” added and removed using the brush. This lets the user experiment how ants react to the existence and lack of food pheromone, or for example what happens an existing pheromone trail is erased. All colonies can sense this pheromone.



Ants following an artificial food pheromone trail created with the brush

5.6.3.14. *Add/Remove Nest pheromone*

Nest pheromone can be “artificially” added and removed using the brush. This lets the user experiment how ants react to the existence and lack of nest pheromones, or for example what happens an existing pheromone trail is erased. All colonies can sense this pheromone.



Ants falsely following and artificially drawn nest pheromone

5.6.4. Tracking

This chapter showcases the models abilities to track an ant.

5.6.4.1. *Watch ant*

Marks an ant in a spotlight and dims rest of world.

5.6.4.2. *Ride ant*

Marks an ant in a spotlight and dims rest of world. Ant is always in the center of the world view.

מעקב על נמלה

5.6.4.3. *Stop viewing ant*

Stops Watching/Riding an ant.

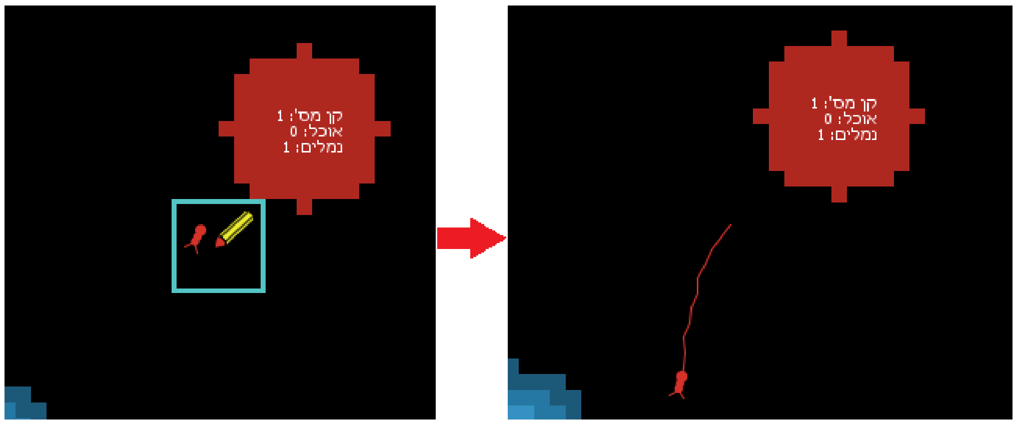
הפסק מעקב\תצפית

5.6.4.4. *Trace ant*

Paint with brush on ants to make them start/stop tracing their movement. This can be used to monitor paths an ant takes.

נמלה משאירה שביל

*יש לצייר בעזרת המברשת על הנמלים שישאירו שביל איפה שהם הולכים



5.6.4.5. *Stop tracing all ants*

All ants stop tracing.

הפסק סימון שבילים

*כל הנמלים יפסיקו לסמן את השביל שלהם

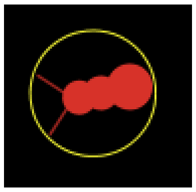
5.6.4.6. *Remove all traces*

Remove all traces drawn so far in world.

נקה שביל נמלים

5.6.4.7. *Mark ant*

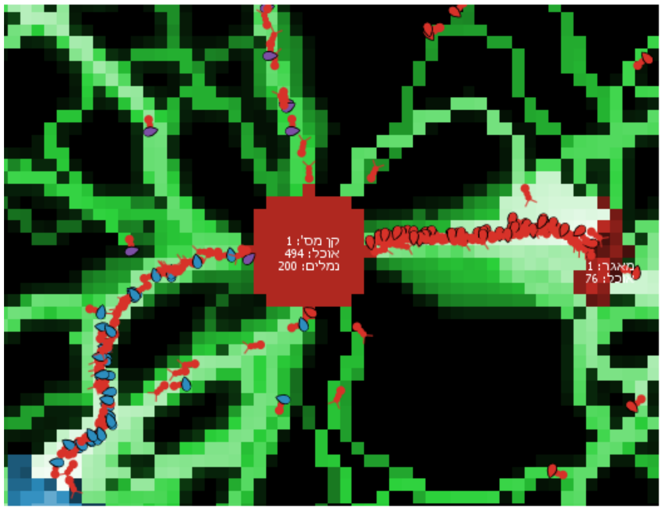
Adds a circle around ant to visually distinct it. Paint with brush on ants to add or remove circle. Can be used on multiple ants in comparison with watching/riding an ant.



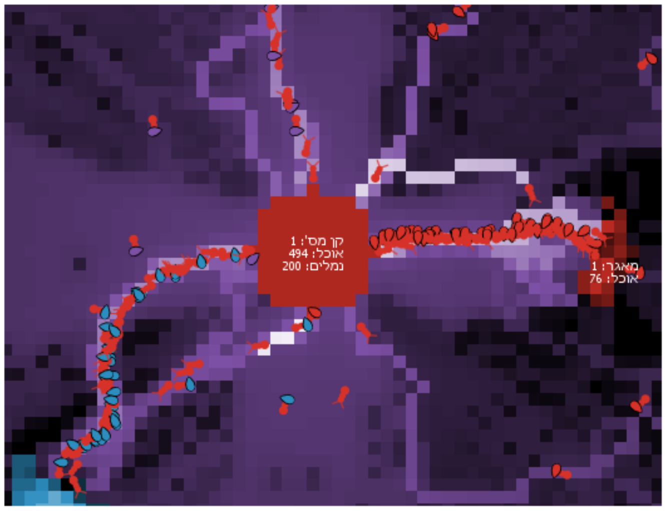
5.6.5. Pheromone display configuration

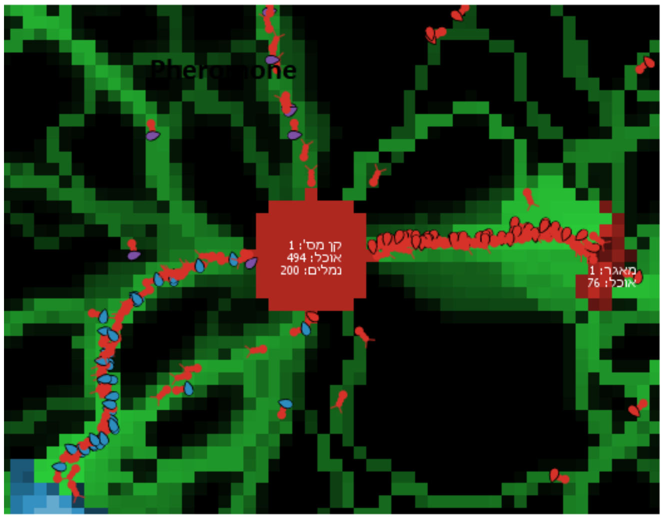
This chapter showcases the different pheromone display configuration this model offers.

Display food pheromone

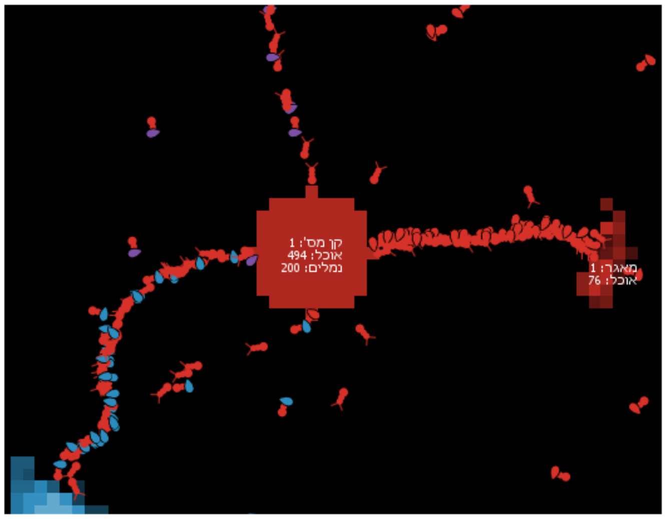


Display nest pheromone



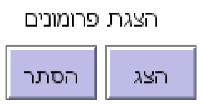


Pheromone visibility = 30%



Hide pheromone

5.6.5.1. *Show pheromone*

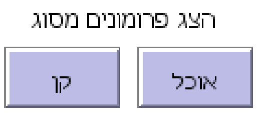


Toggles the current pheromone type to be shown.

5.6.5.2. *Hide pheromone*

Toggles the current pheromone type to be hidden.

5.6.5.3. *Display food pheromones*



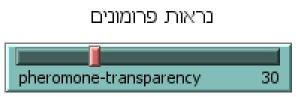
Displays food pheromone of colony currently selected.

5.6.5.4. *Display nest pheromones*

Displays nest pheromone of colony currently selected.

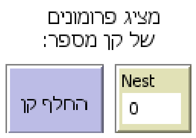
5.6.5.5. *Pheromone visibility*

Configure how visible the pheromones are.



5.6.5.6. *Select colony displaying pheromone*

Select which colony to display pheromones for.



5.6.6. Save/Load world state

5.6.6.1. *Save*

Save current world state to a file.

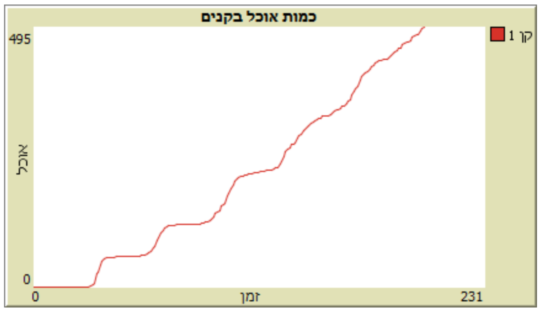
5.6.6.2. *Load*

Load world state from a file.

5.6.7. Plots

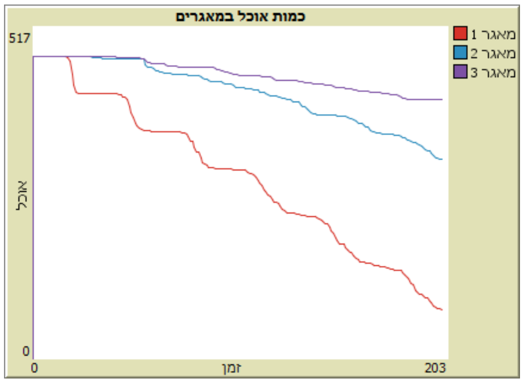
5.6.7.1. *Food in nests plot*

Plot that monitors amount of food in each nest.



5.6.7.2. *Food in sources plot*

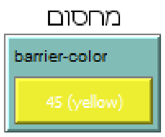
Plot that monitors amount of food in each source.



5.6.8. Color configuration

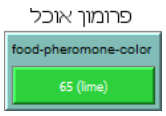
5.6.8.1. Wall color

Set color of walls.



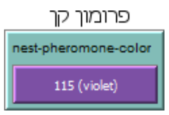
5.6.8.2. Food pheromone color

Set color of food pheromones.



5.6.8.3. Nest pheromone color

Set color of nest pheromones.

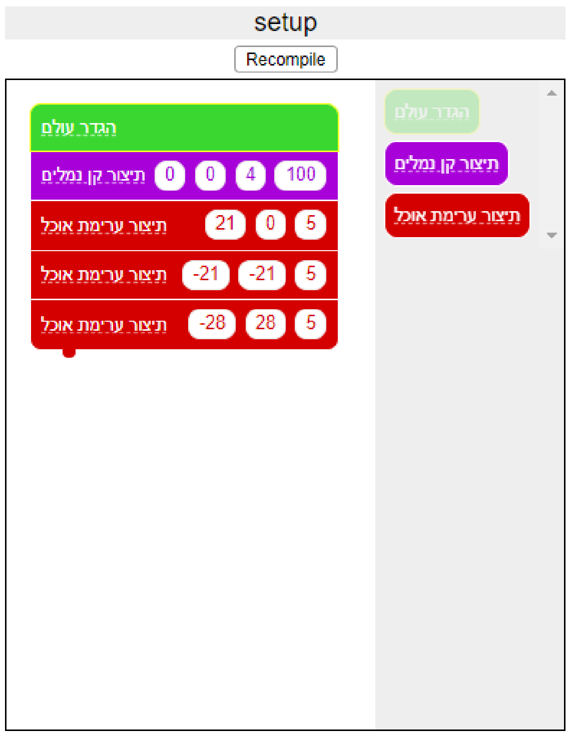


5.7. Block-based programming interface

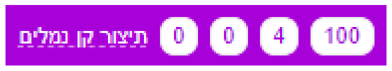
The blocks interface consists of 2 block spaces, one for world initialization and the other for defining ant behavior.

5.7.1. World initialization block space

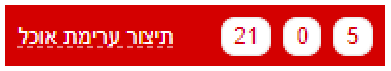
This block space is used to to configure the world state when the model is reset. The blocks can be used to create an ant colony (a nest with ants) and food sources.



Create ant colony – parameters: x-coordinate, y-coordinate, nest radius, amount of ants to create in center of nest.



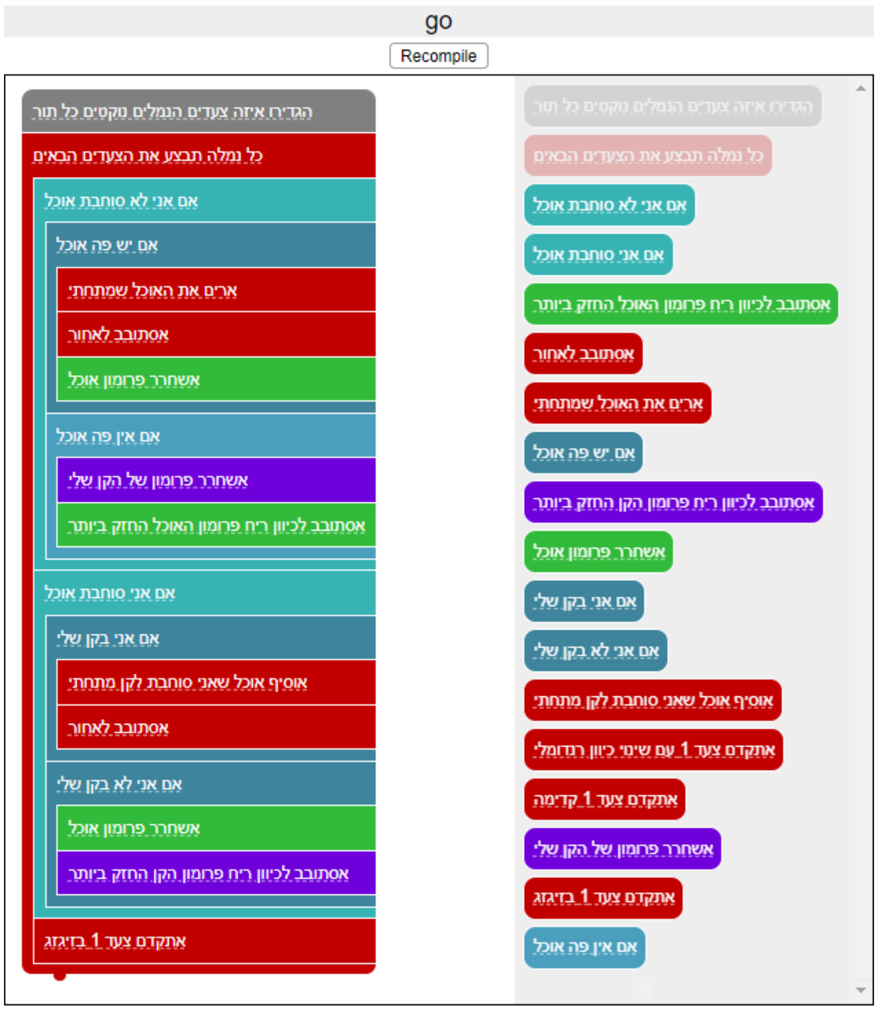
Create food source – parameters: x-coordinate, y-coordinate, radius.



5.7.2. Ant behavior block space

The main goal of this sub-project is for the user to successfully configure the blocks in the “Ant behavior block space” in a way that will affect the colony as a whole to successfully gather food. This block space consists of numerous blocks that determine the ants actions taken each step in the simulation. Below is a screenshot of a block space that is already set up correctly to enable an ant colony to gather food.

The blocks are grouped by color to help distinguish them apart by their relevance of functionality.



```
הגדירו איזה צעדים הנמלים נוקטים כל תור
כל נמלה תבצע את הצעדים הבאים
  אם אני לא סוחבת אוכל
    אם יש פה אוכל
      ארים את האוכל שמתחתי
      אסתובב לאחור
      אשחרר פרומון אוכל
    אם אין פה אוכל
      אשחרר פרומון של הקן שלי
      אסתובב לכיוון ריח פרומון האוכל החזק ביותר
  אם אני סוחבת אוכל
    אם אני בקן שלי
      אוסיף אוכל שאני סוחבת לקן מתחתי
      אסתובב לאחור
    אם אני לא בקן שלי
      אשחרר פרומון אוכל
      אסתובב לכיוון ריח פרומון הקן החזק ביותר
  אתקדם צעד 1 בזיגזג
```

5.8. Algorithms

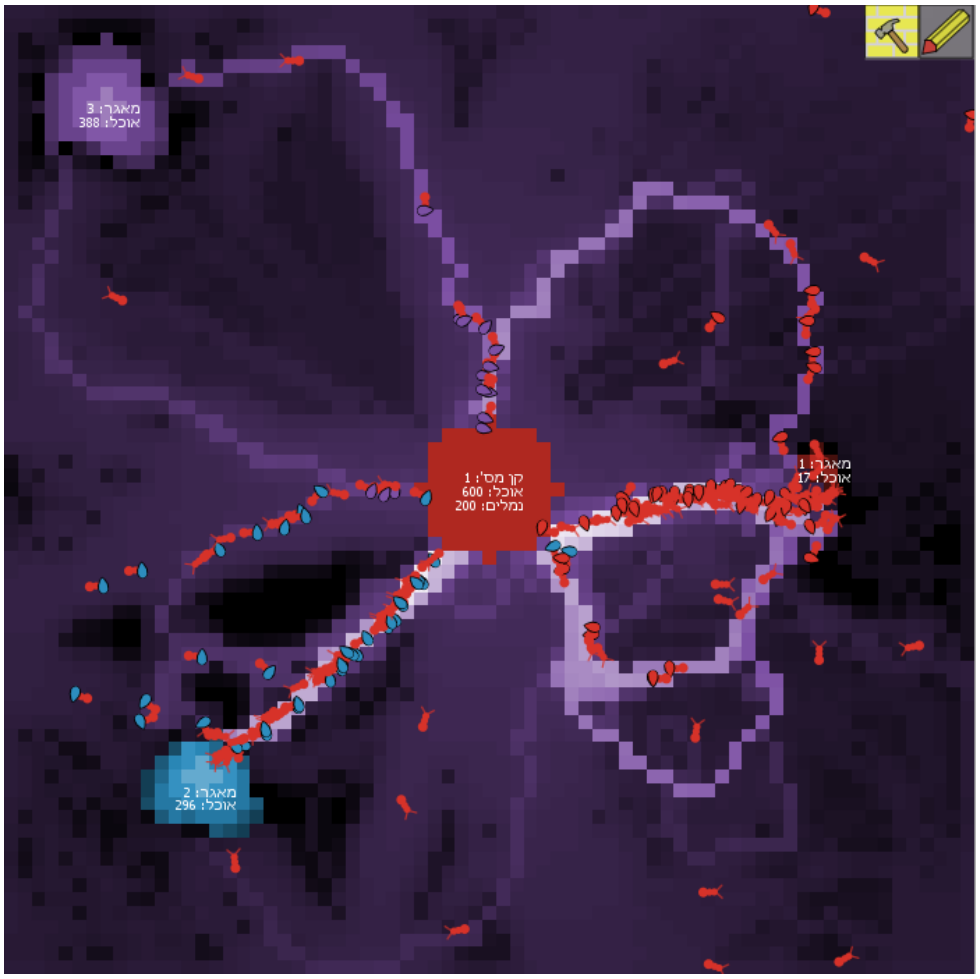
This chapter describes the main algorithms used in the computational model programmed in NetLogo.

5.8.1. Pheromone release

Ant colonies map out the world to sources they need to find by releasing pheromones in varying strength on the patch they are on to measure how close or far they are from that source. The algorithm for how much pheromone an ant releases is based on the Dijkstra algorithm, with the following rules:

- When an ant encounters a source patch to guide other ants to, it will set itself to release the maximum amount of pheromone.
- Every time an ant moves it decreases the amount of pheromone it is set to release. This serves as a form of measurement to how far an ant is away from the source patch.
- If the ant is standing on a patch that has a higher pheromone consistency than that of the amount it is set to release, it will set itself to release the higher pheromone value in the patch it is on.
- Every turn an ant releases pheromone on the patch it is standing on, in the amount it was set to release.

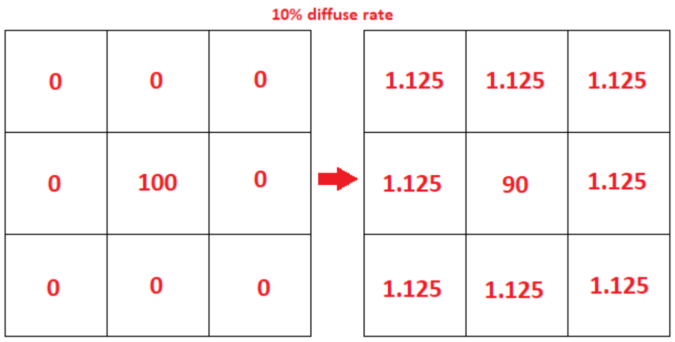
This algorithm allows the ants to map out paths to sources they need to find (such as food or nest). The ants follow the pheromone trails heading in the direction of the strongest pheromone they sense.



Ants leave a nest pheromone trail (purple) back to their nest. The trails with the brightest shade of purple have the highest pheromone consistency and preferred route of ants.

5.8.2. Pheromone diffusion and evaporation

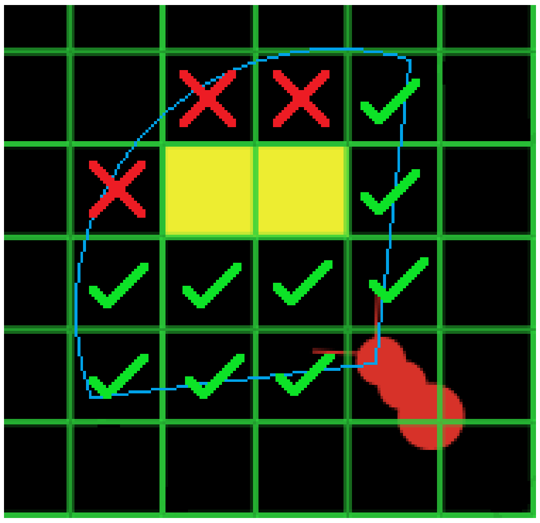
Each turn, the model spreads pheromones around the world by diffusion. After diffusion occurs, the pheromones are then evaporated. Diffusion removes a percentage of a pheromone from a patch and splits it up evenly between neighboring patches. The diffusion rate for pheromones is set at 0.4. This variable can be tweaked. Evaporation is done by lowering pheromone values by a certain percentage.


10% diffuse rate

0	0	0
0	100	0
0	0	0

1.125	1.125	1.125
1.125	90	1.125
1.125	1.125	1.125

5.8.3. Ant Smelling pheromone



Ants smell pheromones by looking for the patch with the highest (or lowest) pheromone value in a cone in front of it. The distance and angle of the cone can be tweaked. Ants can not smell pheromones through walls, so ants disregard patches in front of them that are behind them.

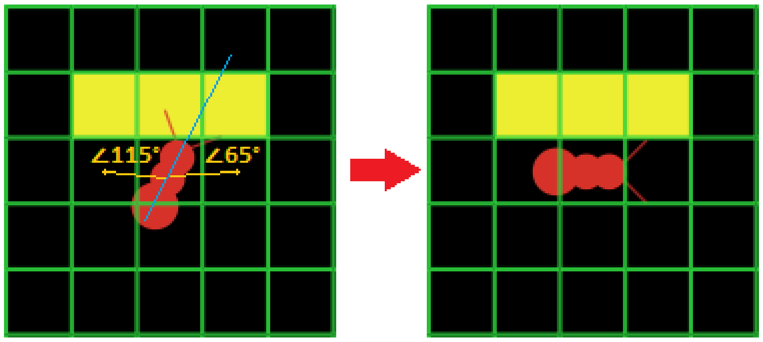
The algorithm works by passing through patches in order of closest headings to one another and assessing if the patch's center is behind the heading of a patch with a barrier and at a longer distance behind it (which means it is blocked from the ants point of view).

It is crucial for ants not be able to smell pheromones through walls otherwise they can get stuck trying to head in a direction that is blocked.

5.8.4. Scavenging for food at the start

When the model is initialized, there are no food pheromones for an ant to follow. It must start scavenging for food. In order to get the ants to look for food throughout the world, they were programmed to head in the direction of the *least* amount of nest pheromones. This way, they move away farthest from the nest. This proved to be very efficient with maze-like structures that were constructed with walls around the nest.

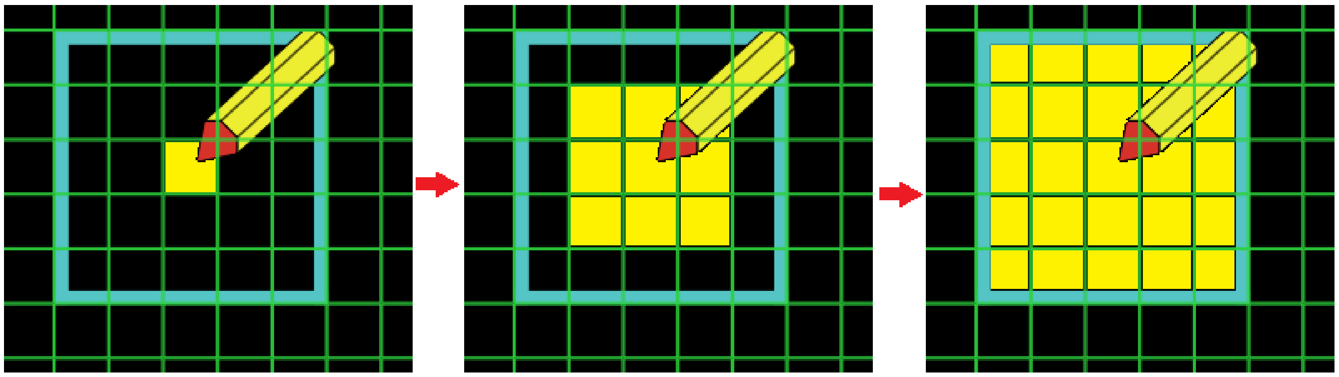
5.8.5. Ant blocked by wall



When an ant is at a very close distance to a wall it is considered blocked and will change direction to a neighboring patch that is unblocked and requires the least amount of heading change.

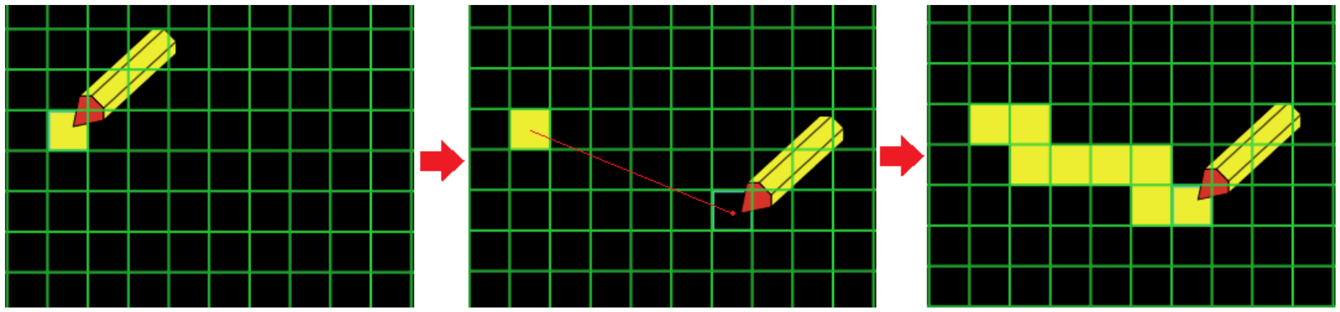
5.8.6. Creating walls and moving ants aside

When a wall is created on a patch that is occupied by ants, if the ants can be moved aside to a neighboring unblocked patch they will be relocated and a wall created in their place. In order to achieve this fluidly without trapping ants inside the newly created batch of walls, the wall are created in order of inside out from the center, with the center being the patch that was directly under the brush. It is important to create walls this way so that any ants will be pushed out without getting trapped.



5.8.7. Painting with brush without skipping patches

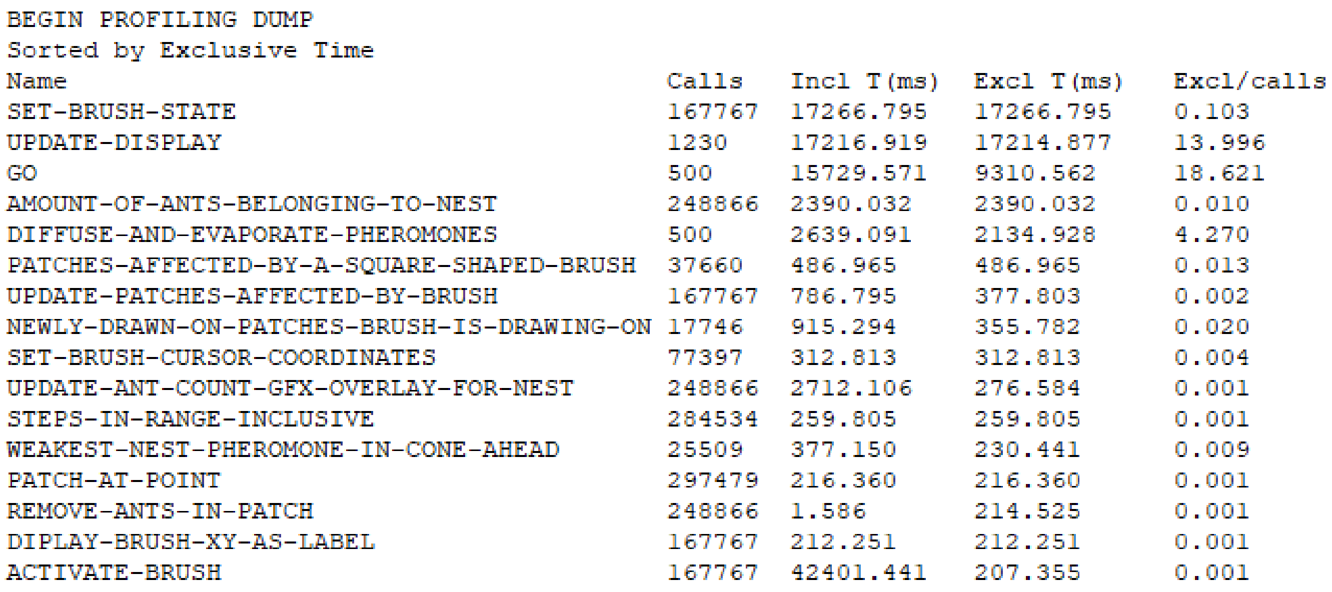
Brush position gets updated discretely (depending how fast NetLogo is updating), so it might jump ahead a few patches from its current position, so it is necessary to calculate the patches the brush skipped over and paint them as well with the brush. This is done by calculating the intersections of the world grid with line between current and last brush position.



5.9. Optimization

NetLogo comes with a built-in *profiler* extension that enables the programmer to time procedure execution on the model and assess where the bottlenecks are. The profiler is an indispensable tool and critical to have made this model work fluidly. The profiler was widely used throughout the project to make sure the model can run simulations without slowing down.

Many of the algorithms and coding was refactored a number of times with the help of the profiler to minimize slowdowns.



```
BEGIN PROFILING DUMP
Sorted by Exclusive Time
Name                                        Calls    Incl T(ms)  Excl T(ms)   Excl/calls
SET-BRUSH-STATE                             167767   17266.795   17266.795    0.103
UPDATE-DISPLAY                              1230     17216.919   17214.877    13.996
GO                                          500      15729.571   9310.562     18.621
AMOUNT-OF-ANTS-BELONGING-TO-NEST            248866   2390.032    2390.032     0.010
DIFFUSE-AND-EVAPORATE-PHEROMONES            500      2639.091    2134.928     4.270
PATCHES-AFFECTED-BY-A-SQUARE-SHAPED-BRUSH   37660    486.965     486.965      0.013
UPDATE-PATCHES-AFFECTED-BY-BRUSH            167767   786.795     377.803      0.002
NEWLY-DRAWN-ON-PATCHES-BRUSH-IS-DRAWING-ON  17746    915.294     355.782      0.020
SET-BRUSH-CURSOR-COORDINATES                77397    312.813     312.813      0.004
UPDATE-ANT-COUNT-GFX-OVERLAY-FOR-NEST       248866   2712.106    276.584      0.001
STEPS-IN-RANGE-INCLUSIVE                    284534   259.805     259.805      0.001
WEAKEST-NEST-PHEROMONE-IN-CONE-AHEAD        25509    377.150     230.441      0.009
PATCH-AT-POINT                              297479   216.360     216.360      0.001
REMOVE-ANTS-IN-PATCH                        248866   1.586       214.525      0.001
DIPLAY-BRUSH-XY-AS-LABEL                    167767   212.251     212.251      0.001
ACTIVATE-BRUSH                              167767   42401.441   207.355      0.001
```

Example of a profile dump showing execution times

Examples of import optimizations made:

- Updating display selectively

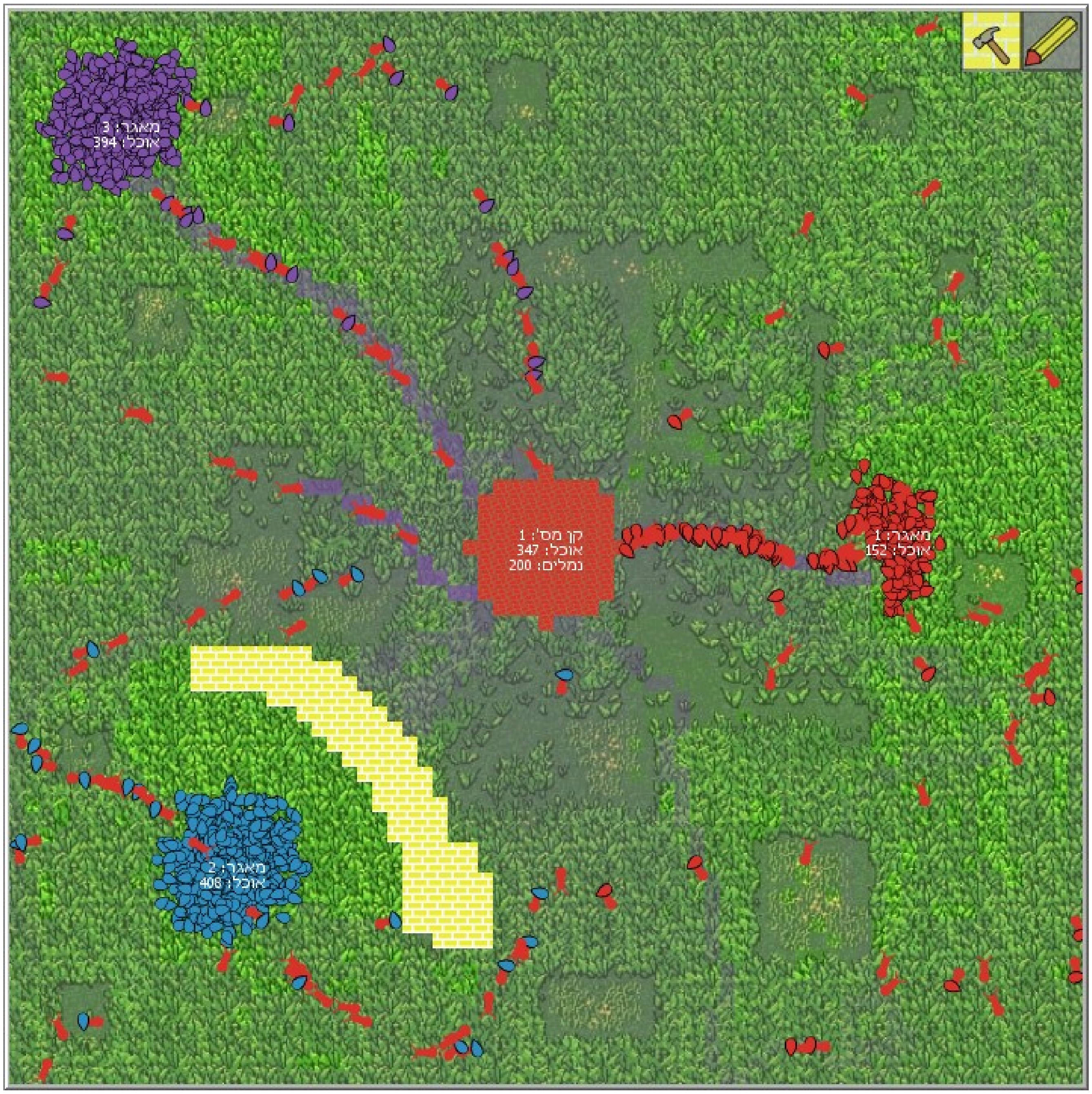
  Recoloring all patches every turn was very costly. They need to be recolored due to pheromone value changing thus the patch color changes. Instead, since it is hard to see any difference in color every turn for pheromone color change (due to evaporation and diffusion), patches were recolored only every 6 turns, which seemed to be the limit until it became noticeable that colors were being updated in time steps. Only patches with ants on them were updated every turn.

- Patches blocked by walls

  The algorithm for checking with patches are behind walls in an ants line of sight was important since it was a major slowdown to the models execution, so a new algorithm was made to speed it up.

5.10. Drawbacks

We encountered one major drawback in the project where we had to forego numerous graphical improvements to the model due to unforeseen circumstances. When we integrated the model with the block-based interface in NetTango, the model run extremely slow at about 3 frames per second. The reason for this is that NetTango runs NetLogo models in the *NetLogo web* version, which executes differently than the desktop version. Simplifying the graphics made it run smoothly again. The graphically enhanced version included a background image and nicer graphics for nests, food and walls.



5.11. Log

This chapter lists an exhaustive list of features and issues documented throughout the project.

5.11.1. Features

Feature	Labels	Status	Info
Add/Remove barrier with brush	Brush,	Done	Brush will continuously add barriers while it is held

	barrier		down, can not draw barrier in patches that already contain food or nests. If patches have ants in them, then ants will be moved aside to nearest vacant patch and then barrier will be added to patch if ants were successfully moved to another patch.
Add/Remove food with brush	Brush, food	Done	Add food to patches brush is drawing on. If brush was drawing on patches that contain an existing food source when first pressed down, then it will add that food source to the patches it is drawing on. Otherwise it will create a new food source.
Add/Remove ants with brush	Brush, ant	Done	Creates ants when brush is clicked. If brush is clicked on nest then adds ants belonging to that nest. Otherwise creates ants belonging to default nest (usually nest with lowest number).
Add/Remove nests with brush	Brush, nest	Done	Add nest to patches brush is drawing on. If brush was drawing on patches that contain an existing nest when first pressed down, then it will add that nest to the patches it is drawing on. Otherwise it will create a new nest.
Brush size	Brush	Done	Brush affects more patches the bigger size it is. Add chooser for brush size.
Brush shape square/circle	Brush	Done	Draw with brush on patches in a square or circular shape. Add chooser for brush shape.
Brush outline around patches it affects	Brush	Done	An outline drawn around patches that are under mouse pointer that will be affected by brush.
Buttons for choosing brush type instead of chooser	Brush	Done	Using buttons to choose brush type is easier for the user, and also enables using action keys to choose brush type
Brush icon to indicate which brush type is currently chosen	Brush	Done	An icon will appear in view that will show which brush type is currently chosen, i.e. “barrier” is the brush type, then a barrier icon will appear near mouse pointer. Need to take into account brush outline
Brush outlines every single patch individually it will draw on instead of single outline around all patches	Brush	Open	Not necessarily better than current outline implementation that outlines all patches
Brush outline change colors (possible for each individual patch it outlines) based on current brush type and other factors	Brush	Done	i.e. when adding barriers with brush, it can outline in red patches that barrier cannot be added on. Update: Brush outline changes colors depending if brush is in erase (red) or draw (cyan) mode.
Mark ants so they are easier to follow	Brush, mark, ant	Done	A circle will outline a marked ant and follow it as ant walks around. Marking an ant will be down with the brush, so as removing their mark
Show trail of selected ants	Brush, ant	Done	Show trail of ant as it moves around, selecting/deselecting ants to show their trail will be done using the brush
Remove all ant trails	Ant	Done	Button that will remove all trails in view
Disable all ant trails	Ant	Done	Button that will stop all ants from drawing their trail
Add/Remove food pheromone with brush	Brush, ant	Done	Need to think of how much pheromone will be added to patch brush Is drawing on, maybe for the first patch clicked it will be max value, and keep getting weaker for every patch further away from it. Update: Adds maximum value for pheromone and keeps getting stronger (by a very small value) the more patches are drawn on. This is done because if the model

			is running, then the pheromone gets evaporated, so pheromone that was last added with a brush while being held down will have the highest value due to evaporation.
Add/Remove nest pheromone with brush	Brush, ant	Done	Same as food pheromone
Plot food for each source	Plot, food	Done	X is ticks, y is amount of food currently available from specific source in world
Plot food in each nest	Plot, nest	Done	Plot showing how much food acquired in each nest, x axis is ticks, y axis is amount of food.
Seed shape for food	Gfx, food	Done	Food will be its own agent breed instead of patch variable, this way we can give it a seed shape as well.
Ants carrying food with color of food they picked up	Gfx, ant	Done	Ants carrying food will look like they have piece of food on the front part, the color of the food they picked up
Tile gfx for barriers	Gfx, barrier	Done	Barriers will be their own agent breed instead of patch variable, this way we can give it a tile shape as well. Update: Due to netlogo web running slow with many turtles, instead of gfx turtles for barriers the patch is colored accordingly
Tile gfx for nests	Gfx, nest	Done	Nests will be it's their agent breed instead of patch variable, this way we can give it a tile shape as well. Update: Due to netlogo web running slow with many turtles, instead of gfx turtles for barriers the patch is colored accordingly
Background image (of grass or anything that can look nice)	Gfx	Done	This can be done by a few ways: 1 Coloring the patches to a certain color as background. This won't look the best 2 Adding a gfx grass tile for each patch. Create a new turtle breed for these graphic tiles. 3 Adding a background image of grass There are a few possible ways to add pheromone color for each option. Patches are hidden under background image (option 3) so there is the possibility of creating gfx pheromone tiles for each patch. Update: Background image was added as background image, and transparent turtles that are square shaped and represent the gfx for pheromone color the patch so the pheromone is visible. Update: Deprecated. Netlogo web seems to run very slowly with a lot of gfx pheromone turtles, so the background image is not a viable option currently.
User can set background image by choosing an image	Gfx	Done	User clicks designated button to select background image, a file explorer pops up and user picks image. Update: Deprecated
Alert user if background image chosen is not supported	Interface	Done	User gets a message that file chosen could not be used as background image. Update: Deprecated
Remove background image	Interface, gfx	Done	User clicks designated button to remove current background image. Update: Deprecated
Default background image	Gfx	Done	Supply a default background image. This can be done by converting an image to base64 and storing that in the code. A script can be written to receive a base64 string and break it up into smaller chunks of

			variables so it can be coded into the netlogo file.
Switch visually between showing nest and food pheromone	Gfx, food-pheromone, nest-pheromone	Done	Button/chooser combo to change between showing either food pheromone or nest pheromone in view on patches.
Ant color	Gfx, ant	Done	Currently, the changeable color in ant shape is the food it is carrying, but there can be only 1 color that changes in shape. In order to make ants color change too (and not only the food it is carrying) we will need to make a different shape for each ant color with a different name
Plot amount of ants carrying food	Ant	Open	
Plot amount of ants not carrying food	Ant	Open	
Ant changes direction when blocked by barrier	Ant, barrier	Done	Ant will face in direction of neighboring patch that is unblocked and requires the minimum amount of heading change
Display patches that the ant is sniffing	Ant	Open	Select ants with brush to display which patches they sniff, maybe even which patch they choose to go to.
Ants return to nest if they haven’t found food in certain amount of ticks	Ant	Pending	Is this behavior we want?
Ants die if they haven’t eaten food in certain amount of ticks	Ant	Pending	Is this behavior we want?
Nests create new ants if there is enough food	Nest, ant	Pending	Is this behavior we want?
Unique nests, each nest created separately is a new nest	Nest	Done	When creating nests with brush, automatically determine if it’s a new nest depending on if the brush is creating a nest that is adjacent to an existing nest
Ants drop nest pheromone when searching for food so they can find their way back	Ant, nest-pheromone	Done	Implement a Dijkstra type of algorithm so that the further an ant moves the smaller the amount of pheromone it releases, and if it is on a patch with a higher amount of pheromone than it is currently releasing, then from this point on release that amount of pheromone.
Ants drop food pheromone when carrying food home so other ants can find food source	Ant, food-pheromone	Done	Implement a Dijkstra type of algorithm so that the further an ant moves the smaller the amount of pheromone it releases, and if it is on a patch with a higher amount of pheromone than it is currently releasing, then from this point on release that amount of pheromone.
Pheromone can’t pass through barriers	Pheromone	Done	At the end of every turn, all patches that have barrier on them have food and nest pheromone set to 0, that way pheromones can’t be diffused through barriers
Ant drops food if it dies	Ant, food	Pending	Is this behavior we want?
Ant drops food on floor if given command “drop food” in NetTango.	Ant, food, nettango	Pending	Is this behavior we want? Currently if ant is not standing on a nest and is asked to drop food, then nothing happens. This could lead to unusual behavior since it will realize it is nest to a food the next turn, and will start dropping food pheromone at max strength since it thinks it has just found a new food source, but it was the food it dropped last turn.
Brush adds food on each patch only once since it was first held down	Brush, food	Done	Otherwise the food gets added way too quickly. Another possibility is to add food continuously but at a slower pace.
Barriers added on patches with food will move food aside to closest	Barrier, food	Pending	Is this behavior we want?

unblocked patch			
Barriers added on patches with ants will move ants aside to closest unblocked patch	Barrier, ant	Done	Ants are moved to center of closest unblocked patch
Continuous fluid brush drawing that does not skip patches	Brush	Done	If moving mouse too quick then the brush will skip patches mouse moved over. Calculate the patches that the mouse moved over between current and last brush position.
Slider to determine how much food to add with brush	Brush, food	Pending	Is this behavior we want? Currently adds a random amount of food, maybe change behavior to add specified amount of food equally to patches under brush
Show “+1” animation when ant adds food to nest	Food	Open	Make “+1” it go up like in MMO for a second then disappear.
Create barrier tunnels with brush	Brush, barrier	Open	Easily create a “tunnel” with brush that ants can pass through, instead of having to draw 2 parallel lines.
Create circles with brush	Brush	Open	Instead of drawing a circle free hand, have the option for brush to draw a perfect hollow circle, can be useful for creating a barrier maze
Create squares with brush	Brush	Open	Instead of drawing a square free hand, have the option for brush to draw a perfect hollow square, can be useful for creating a barrier maze
Save world	Interface	Done	Click on “Save” button to save current world state to a file so it may be loaded later on.
Load world	Interface	Done	Click on “Load” button bring up a file explorer to choose a world state to load.
Reload world	Interface	Done	Reload that last world state that was loaded Update: Currently this functionality is deprecated
Ants can detect food from certain distance	Ant	Done	Instead of only going towards strongest food pheromone, if ant is close enough to food then it will go towards the food.
Slider for: food/nest pheromone sniff sensitivity, evaporation rate, diffusion rate, drop rate, pheromone released decrease rate	Interface	Done	Update: sliders removed to reduce complexity of interface.
Ants search for food in direction of least amount of nest pheromone if they cannot sense food pheromone	Ant	Done	This is done so ants spread out furthest away from nest. This has improved the speed at which ants find food, especially at the start and is noticeable when maze like structure is built around the nest
Undo/Redo brush	Brush	Open	
Ants can only sniff patches that are not blocked in line of sight by a barrier	Ant, pheromone, barrier	Done	This is important since ants can act weirdly or get kind of stuck if they can sniff a patch behind a barrier and go towards it.
Action key to toggle between all brush types	Brush	Open	
Add exact amount of food in given radius	Brush, food	Open	
Control ant with mouse	Brush, ant	Open	Ant goes towards mouse
Visual feedback when ant collides with barrier	Ant, barrier, gfx	Pending	Possible to make the barrier flash for a certain time, or vibrate maybe.
Ants belong to a single nest	Ant, nest	Done	Ants can only return food to their own nest they belong to. This means that each nest will have its unique nest pheromone that only ants belonging to that nest can release.

Unique pheromones for each nest	Pheromone, nest	Done	Each nest of ants has its own food and nest pheromones that only ants belonging to that nest can smell.
Creation of multiple (different) nests	Brush, nest	Done	Create different nests with brush. If drawing on patches that do not contain an existing nest, create a new nest. If part of the brush if drawing on an existing nest, then extend that nest.
Create ants belonging to specific nest with brush	Brush, ant	Done	Clicking on a nest while brush is selected to create ants will create ants of that specific nest. Clicking anywhere else in the world will create ants of default nest (nest with smallest source number)
Show amount of food in nest	Nest, gfx	Done	The amount of food in a nest is visually shown on the nest and centered.
Show amount of ants in nest	Nest, gfx	Done	The amount of ants in a nest is visually shown on the nest and centered.
Show nest number of nest	Nest, gfx	Done	The nest number is visually shown on the nest and centered.
Different color for ants belonging to different nests	Ant, gfx	Done	Ants will have a slightly brighter shade of the color of their nests.
Flash turtle or patch	Gfx, code	Open	Ability to flash a turtle or patch for a given time. Add option to choose flash color, frequency. A way to achieve this is by creating a turtle on existing turtle or patch with the same shape so it covers it, and have that new turtle flash colors. Tie them together with a link so they have same position and heading.
Shake turtle or patch	Gfx, code	Open	Ability to visually shake an object or patch for a given amount of time and frequency. A way to achieve this is by creating a turtle on existing turtle or patch with the same shape so it covers it, and have that new turtle shake back and forth. Tie them together with a link so they have same position and heading.
Script for fast edit of shapes instead of using turtle shapes editor	Code	Open	Ants of different nests have same shape but different color, so instead of creating them manually, use a script to do it, so any future changes to the basic ant shape can be easily copied to create the same shape for different colors. Need to reverse engineer how turtle shapes are saved in netlogo file and to change them. Update: The netlogo files were reverse engineered and how shapes are saved with their parameters is known, it is fast enough to edit them in a simple text editor to change their colors.
Show amount of food in food pile	Food, gfx	Done	The amount of food in a pile is visually shown on the pile and centered.
Show food pile number	Food, gfx	Done	The food pile number is visually shown on the pile and centered.
Choose pheromone brightness	Interface, pheromone	Done	Slider to select maximum brightness for pheromones displayed in world view
Hide/Show pheromone	Interface, pheromone	Done	Button to select if pheromones will be displayed.
Highlight nest that pheromones are displayed for	Gfx, pheromone, nest	Open	When selecting which nest to display pheromones for, highlight it temporarily (maybe flash) to indicate which visually which nest is chosen.
World is bounded	World, ants	Done	Bound the world so ants can not return cyclically around world. Ants treat world bound as a barrier

Count food in each pile efficiently	Efficiency, code	Done	In order to improve execution time of plots, food count in each pile is monitored, when it is added, removed or picked up by ant and updated to a list, instead of counting all patches with food.
Thicker brush outline	Brush	Done	Make brush outline thicker so it is more visible. A turtle shape with thick outline can be created for this, a slight problem is that for bigger brush sizes it seems a bit too thick. Update: there is an experimental primitive "__set-line-thickness" however it is currently not supported in netlogo web. This seems like the best solution.
Set brush to add or erase with buttons instead of switch	Brush	Done	Added two buttons "Add" and "Erase"
Set brush shape with buttons instead of chooser	Brush	Done	Added two buttons "Circle" and "Square"
Select pheromone type to display (food or nest) with buttons	Brush	Done	Added two buttons "Food" and "Nest"
Translate interface to Hebrew	Interface	Done	Translate both netlogo and nettango component to Hebrew.
Update plot immediately when food is added or removed from world	Plot, food	Done	Plot needs to update immediately after amount of food has changed in world, instead of only updating when world has advanced a turn. This makes the model feel more interactive.
Update plot immediately when a nest is added to world	Plot, nest	Done	Plot needs to update immediately after nest added to world, instead of only updating when world has advanced a turn. This makes the model feel more interactive.
Color selection for barrier	Interface, barrier	Done	Add color selection widget to choose barrier color
Color selection for food pheromone	Interface, food-pheromone	Done	Add color selection widget to choose food pheromone color
Color selection for nest pheromone	Interface, nest-pheromone	Done	Add color selection widget to choose nest pheromone color
Show coordinates of brush	Brush, gfx	Done	Brush icon shows coordinates of brush
Remodel interface and group widgets by relevance	Interface	Done	Remodel interface so that widgets are grouped by relevance and have border added around them

5.11.2. Issues

Issue	**Label**	**Status**	**Info**
Square brush outline is not centered around patches it affects when brush size is even number	Brush	Done	Brush outline for square brush needed to be offset from where mouse was pointing, depending on how patches under brush are calculated
Brush slows down model by a lot when adding anything (barrier/nest/food etc)	Brush	Done	Removed the redundant use of NetLogo primitive "display". It was being used every time a barrier/nest/food/ant was created, slowing down the model substantially.
Brush outline when using ant brush type should not get bigger for bigger brush	Brush	Open	Possibly add icon with how many ants are being added

size			
Ants can go through corner of adjacent diagonal barriers	Barrier, ant	Done	Changed algorithm to check if ant is unblocked by barrier
Brush does not always draw on patches it is on	Brush	Done	Caused due to checking mouse position at different times. The solution was to save the current mouse data to global variables at the start when brush is activated and only refer to them.
Brush draws slowly after adding feature to not skip any patches	Brush, efficiency	Done	Changed algorithm to find intersection of line with patches, runs much faster now
Square shaped brush draws slowly	Brush, efficiency	Done	Changed algorithm to get patches affected by square shaped brush
Erase ant currently works with click, need to change so brush erases ants all the time while brush is held down	Brush	Done	Fixed
Ants bounce around too much between barriers that are too close together causing them to slow down on their way to destination	Ant	Done	This was caused by 2 things mainly, which were that ants could sniff patches behind barriers and that when an ant was close to a barrier it would do a 180 degree turn. I.e. If an ant sniffed a strong nest pheromone while carrying food back home through a barrier then it would go towards that patch, but when it got too close to barrier then it would turn 180. This would cause the ant to bounce around between 2 parallel barriers for a bit (not always but it was unnatural and slowed the ant down). Once the ant could not smell through barriers anymore it could not smell a stronger nest pheromone on a path closer to the nest, and also by changing the behavior of ant that is in front of a barrier to face unblocked patch that requires least heading change made the ants movement much better.
Algorithm for ants sniffing patches that are not blocked in line of sight of barrier is slowing down the model	Efficiency, ant, pheromone	Done	Changed to a faster algorithm that uses angles and distances instead of points of intersection.
Ants can miss food when they are close to it	Ant, food-pheromone	Done	Fixed by making ant drop when it finds food, and not in the next turn.
Errors when world is not a torus (wraps horizontally and vertically)	Code	Open	Numerous places in code that probably need to be changed because they assume that a patch always exists, but with a non-torus world this is not the case since it has boundaries, so more checking needs to be done.
Brush outline is not visible when “Go” button is not pressed.	Brush	Done	Caused because the display was being updated every tick, but when “Go” wasn’t running then nothing was causing the view to update. Solved by updating the view every 0.04 seconds if ticks have not advanced in that time frame.
Save/Load/Reload world does not work in NetLogo web	Code	Done	Need to use extensions “import-a” and “fetch” to handle importing and exporting world state in web version of netlogo.
Brown nests tiles are not aesthetic since the background is also brown, so they appear as brown squares	Nest, gfx	Open	A possible solution is to skip over brown colored nests
Diffusing pheromones is one of the main bottlenecks, multiple nests means diffusing more pheromones.	Efficiency, code	Open	The main reason for the slowdown is because each patch holds a list of pheromones values the length of the amount of nests in the world (each element is the

			pheromone value on this patch for a given nest), and the only effective way of diffusing (time-wise) is by using the “diffuse” primitive. This primitive can only be used on a number primitive, not lists, so in order to diffuse the list of pheromones that each patch holds, there is a need to set a temporarily variable in the patch to contain the value pheromone for a specific nest number (do this at once for all patches for a given nest number), use the “diffuse” primitive on it, then change the value in the pheromone list to contain this new calculated pheromone value. Currently, the way this is done is by looping through each nest number and running the procedure mentioned above. Need to think of a way to speed this up. Solution: Limit amount of nests that can be created, and have a multiple patch variables that hold pheromone value for each nest. This way, it is possible to only use the diffuse primitive on variable without the need to switch between
Turtles shapes in netlogo seem to keep data of shapes that have no dimensions, such as circle with 0 radius, might slow down rendering time.	Efficiency, gfx	Done	Remove redundant data of shapes that are not needed.
Square brush outline not positioned correctly on patches it is drawing on	Brush	Done	Fixed.
Switching between which nest to display pheromone for should only switch between nests that currently exist. If nest gets erased then it is confusing for the user to see pheromones for a nest that is not in the world	Gfx, nest, phermone	Done	Fixed.
Label color of nest information might not be very visible due to nest color (white label on yellow nest isnt very visible)	Gfx, nest	Done	Nest will not be colored colors such as yellow that reduce visibility of white fonts.
Background image is removed when removing ant trails	Gfx	Done	Since both the background image and ant trails are part of the same “drawing” layer, erasing the ant trails erases the background image as well. The solution to this is to reload the background image after erasing ant trails.
Creating nests with arbitrary source number	Code	Open	Can not create nests with arbitrary source number. Such a change would need to change in the code how pheromone values are stored. Would have been solved to begin with if netlogo supported hashes, but since it doesn't then nest source numbers are automatically assigned in ascending order, it is not possible to specify arbitrary source number. This might be problematic for blocks programming, creating ants and deciding what nest they belong to.
Recoloring patches is slowing down execution	Efficiency	Done	Recoloring all the patches every tick slows down execution. It is possible to instead only recolor patches that ants are on, or food patches that ants have taken food from, and recolor the rest of the patches (to update pheromone color) every number of ticks, since the evaporation rate and diffuse rate are low that they do not make a visible difference hence there is no need to

			recolor pheromones in patches each tick. Possible to set how often patch gets recolored depending on evaporation/diffuse rate.
Displayed nest number should start from 1 instead of 0	Gfx, nest	Done	Updated code to show an incremented (by 1) nest number (since nest numbering starts from 0 and up).
Food color at max brightness should not be white but a bright shade of its color, distinguishable from white	Gfx, food	Done	Shade of food is capped at a value that will enable user to distinguish its actual color, so it wont seem white.
Ants get stuck in area searching for food	Ant, pheromone	Open	Ants can get stuck when looking for food since they try to get furthest away from nest pheromone if they can not sense food pheromone, maybe change algorithm to find patch with least amount of ants and least heading change
Marks around ants are not removed immediately after has is removed or moved aside by barrier	Ant	Open	The solution to this is to tie both the ant and mark with a link so that when an ant moves the mark moves with it.

6. Much.Matter.In.Motion (MMM)

6.1. Introduction

This sub-project is a continuation of the already existent Much.Matter.In.Motion modeling platform (Levy, Saba, Lipov & Hel-Or, 2019). The MMM platform seeks to advance the learning of students in the field of science through modeling physical and chemical models via an interactive modeling and block-based programming interface by which they will be exposed to a wide range of phenomena pertaining to these models. User can "draw" the entities in the model they wish to simulate, and define the entities physical properties with the block-based interface.

The MMM platform's model component is programmed with NetLogo, while the block based programming component is built with the Blockly, a client side library for JavaScript.

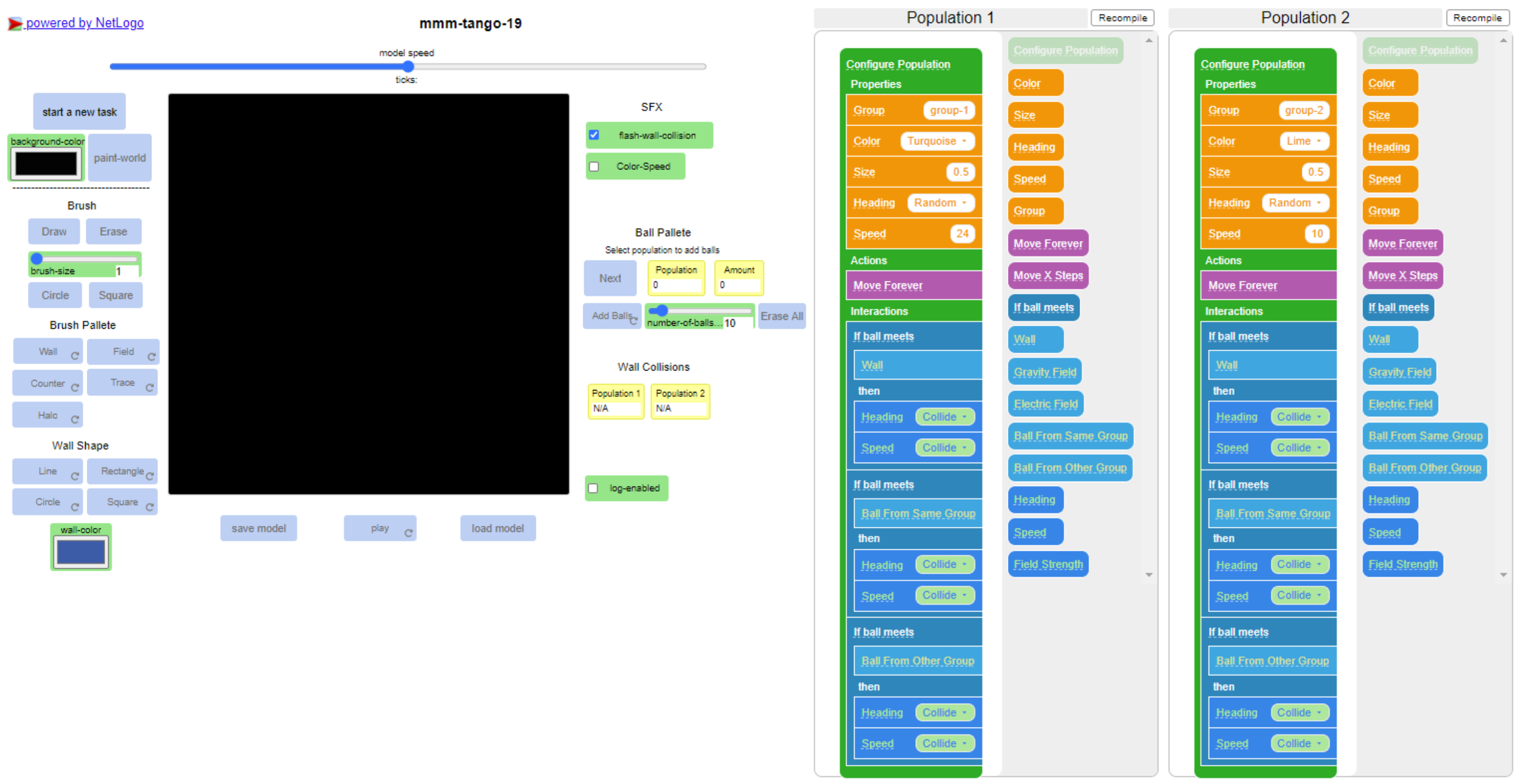
The project can be run in the browser by clicking on the following link (please use Chrome if the page does not load): https://ron-teller.github.io/mmm-netlogo.github.io

6.2. Objectives

The main objectives of this sub-project were to:

- Create a block based programming interface using the NetTango platform instead of the existing Blockly platform currently used in MMM.
- Add functionality and features to existing MMM model.
- Enhance the user interface and experience.

6.3. Overview



The MMM project is comprised of two main components: the model (left) and the block-based interface (right).

6.3.1. Model

The NetLogo model serves as an abstract model for creating physical and chemical systems that can be simulated with a common set of rules and entities. The main acting entities in the model are "balls" which, given how their physical properties are set up ("rules" defined in the block-based interface), can be used to model a great deal of systems in physics and chemistry. The entities (balls) belong to different populations, and each population can be set to have its distinct set of physical properties.

The physical properties of the entities (balls) are represented as three different categories:

1. **Properties** - color, size, initial heading, initial speed
2. **Actions** - move forward
3. **Interactions** - change of heading and speed when ball meets (interacts with): ball of same population, ball of other population, wall.

Examples of systems that can be simulated in the model include:

- Formation of a solar system. The balls act as bodies of mass, which interact with each other by means of gravity.
- Pressure of gas inside a container.
- Electric field

The model's main features are:

- Simulate many physical and chemical systems containing entities and a small set of "rules" common to these systems.
- Use a brush to interactively "paint" and model the physical/chemical system.
- Observe the complex system as it develops over time.

6.3.2. Block-based interface

The block-based interface defines the physical properties of the entities (ball populations) simulated in the scientific computational model, how they interact between themselves and other entities. The aim is for the user to model a system by means of defining the physical properties for each ball population by arranging the blocks and configuring the parameters.

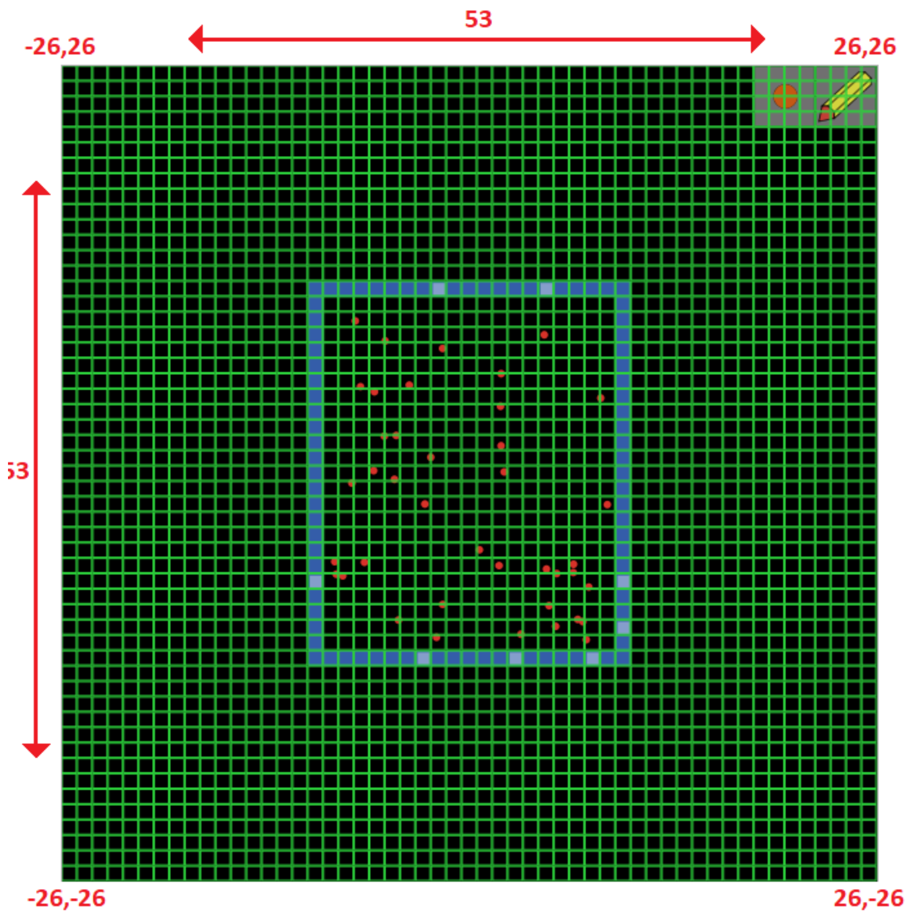
6.4. World Entities & Components

The MMM model simulates a world of several entities and components that make up the complex system. This section described all existing entities and components and how they interact with each other.

6.4.1. World space

All entities in the MMM model exist inside a wrapped 2D space. Inherent to all models in NetLogo, the 2D space is sectioned into a grid, and each cell in the grid is referred to as a “patch”. All entities must move and exist within the 2D space. When a ball moves out of bounds it wraps to the other side of the world.

The size of the world is set to 53×53, which means that the world grid contains 53 rows and 53 columns. The center patch of the world is located at coordinate (0,0). The world contains a total of 2809 patches. The model can easily be configured to be smaller or bigger in the NetLogo IDE.



6.4.2. Balls

Balls are the main acting entities in the model. Balls can represent different entities depending on which physical or chemical system is being modeled, such as atoms, electrons, bodies of mass and more. Balls belong to a certain population, and each population has its own set of defined physical properties that specifies the balls behavior, and are represented as 3 categories of properties, actions and interactions.

6.4.2.1. Properties

- **Color** - Color of the ball.
- **Size** - Size of ball, can represent mass as well.
- **Initial heading** - Heading of ball when it is first created
- **Initial speed** - Speed of ball when first created.

6.4.2.2. Actions

- **Move Forward** - Ball moves forward each turn.
- **Move Forward X Steps** - Ball moves forward for X number of turns then stops.
- **Do not move** - Ball does not move. This option is represented as a lack of the the former actions.

6.4.2.3. Interactions

The heart of the system modeling lies here, since how the balls are defined to interact with other entities in the system essentially constructs a specific physical/chemical model.

The result of an interaction changes the speed and direction of a ball. The type of interaction sets the manner at which the heading and speed change.

A ball interaction is constructed of both cause and affect components:

- Which entity is the ball interacting with
- How does this interaction affect the balls heading and speed

Example:

If **ball** meets **wall** then

heading = *collide*

speed = *collide*

The entity the ball is interacting with is the wall. The interactions affects the heading by way of colliding, meaning that the new heading of the ball is to be calculated using collision physics. The speed is also set to change the same way.

6.4.2.3.1. Interacting entities

The different entities that the ball can be set to interact with in the model are:

- *Balls from the same population*
- *Balls from another population*
- *Walls*
- *Gravity field*
- *Electric field*

6.4.2.3.2. Heading change

The types of heading change that can occur as a result of an interaction with balls and walls:

- *No change* - Nothing changes
- *Collide* - Collision physics
- *Turn left* - Turn 90 degrees left
- *Turn right* - Turn 90 degrees right
- *Repel* - Repulsion forces act on ball to change its heading

- *Attract* - Attraction forces act on ball to change its heading (such as gravity)
- *Repel and attract* - Such as protons that attract and repel one another.

6.4.2.3.3. Speed change

The types of speed changes that can occur as a result of an interaction with balls and walls:

- *No change* - Speed does not change
- *Collide* - Collision physics
- *Zero* - Speed set to 0
- *Increase* - Increase speed
- *Decrease* - Decrease speed
- *Repel* - Repulsion forces act on ball to change its speed
- *Attract* - Attraction forces act on ball to change its speed
- *Repel and attract* - Such as protons that attract and repel one another.

6.4.2.3.4. Gravity field

Interaction with the gravity field for a ball is configured with a number. This is the acceleration relevant to the Y axis.

Example:

If **ball** meets **gravity** then

field-strength = 9.81

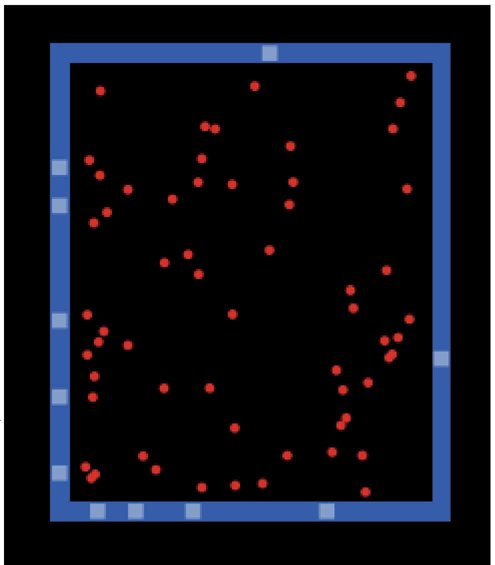
6.4.2.3.5. Electric Field

Interaction with an electric field for a ball is configured with a number. This is the acceleration relevant to the speed the electric field was configured.

Example:

If **ball** meets **electric-field** then

field-strength = 10

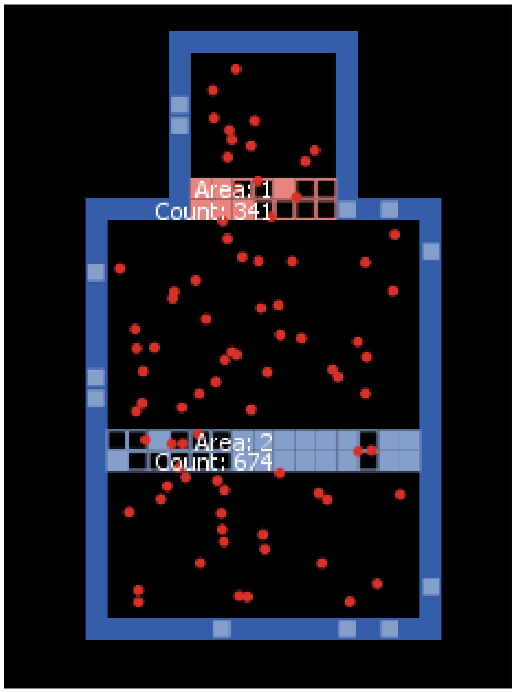


6.4.3. Walls

Walls are stationary entities that balls can interact with. Walls occupy an entire patch. Walls can be used to bound balls inside an area. This can be used, for example, to model air particles inside a container. The model enables flashing walls when a ball collides with them, this can be helpful to assess how many wall collisions there are. In the example above, this can be used to approximate the pressure inside the container - more flashes means more particles hitting the container which means higher pressure.

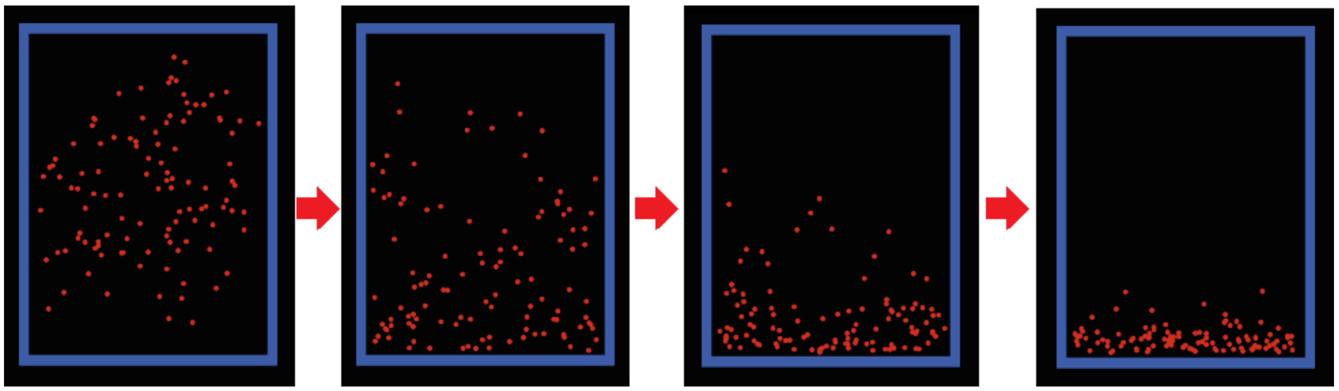
6.4.4. Counters

Counters are tools for measuring the amount of balls passing over a certain area. The counters can be added or removed using the brush. Balls passing over a counter increase its ball count. Counters have different numbers and colors to differentiate between them.



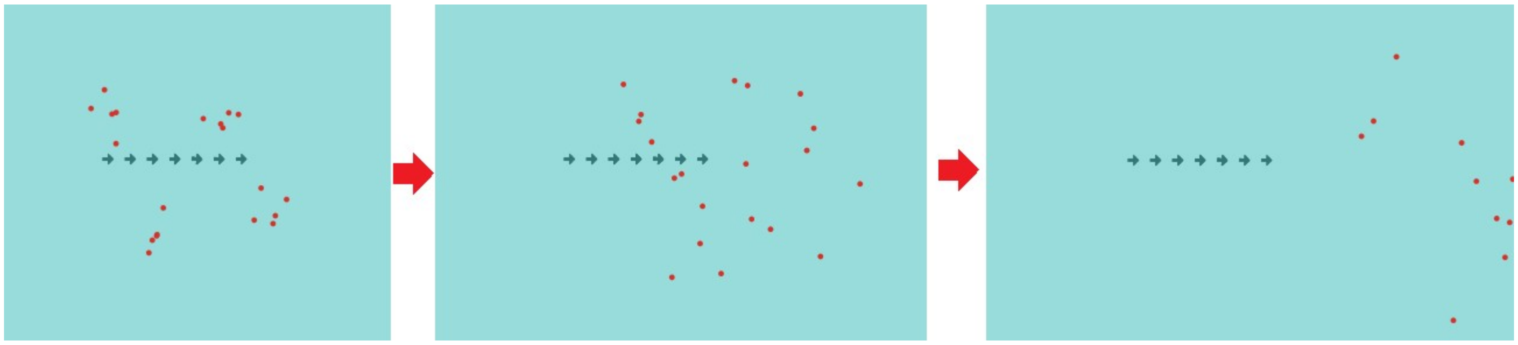
6.4.5. Gravity field

The gravity field is universal in the model in the sense that it exists on the Y axis for all entities. Each ball population can set how much they are affected by the gravity field. A ball population with a gravity interaction set to 0 will not be affected by gravity at all.



6.4.6. Electric Field

The electric field accelerates balls in the direction it was configured. Each ball population can set how much they are affected by an electric field. A ball population with an electric field interaction set to 0 will not be affected by an electric field at all.

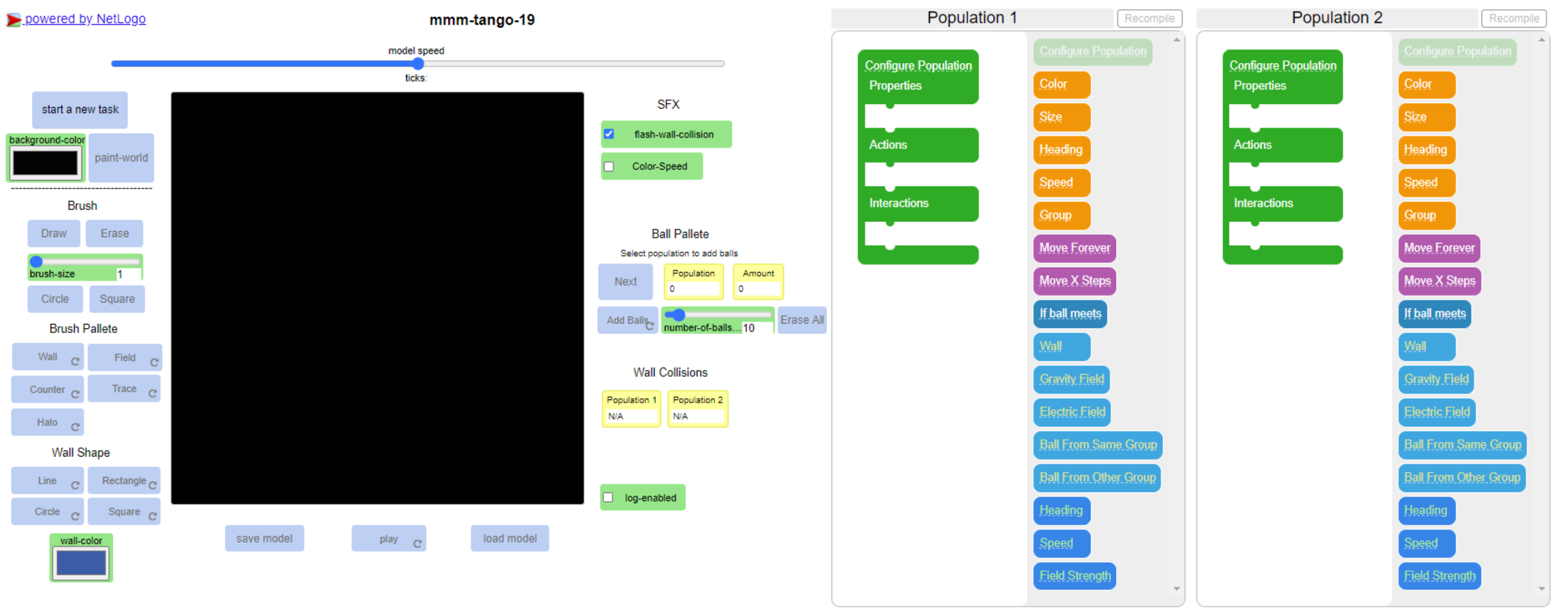


6.5. Example

This chapter showcases the use of this model from the users perspective.

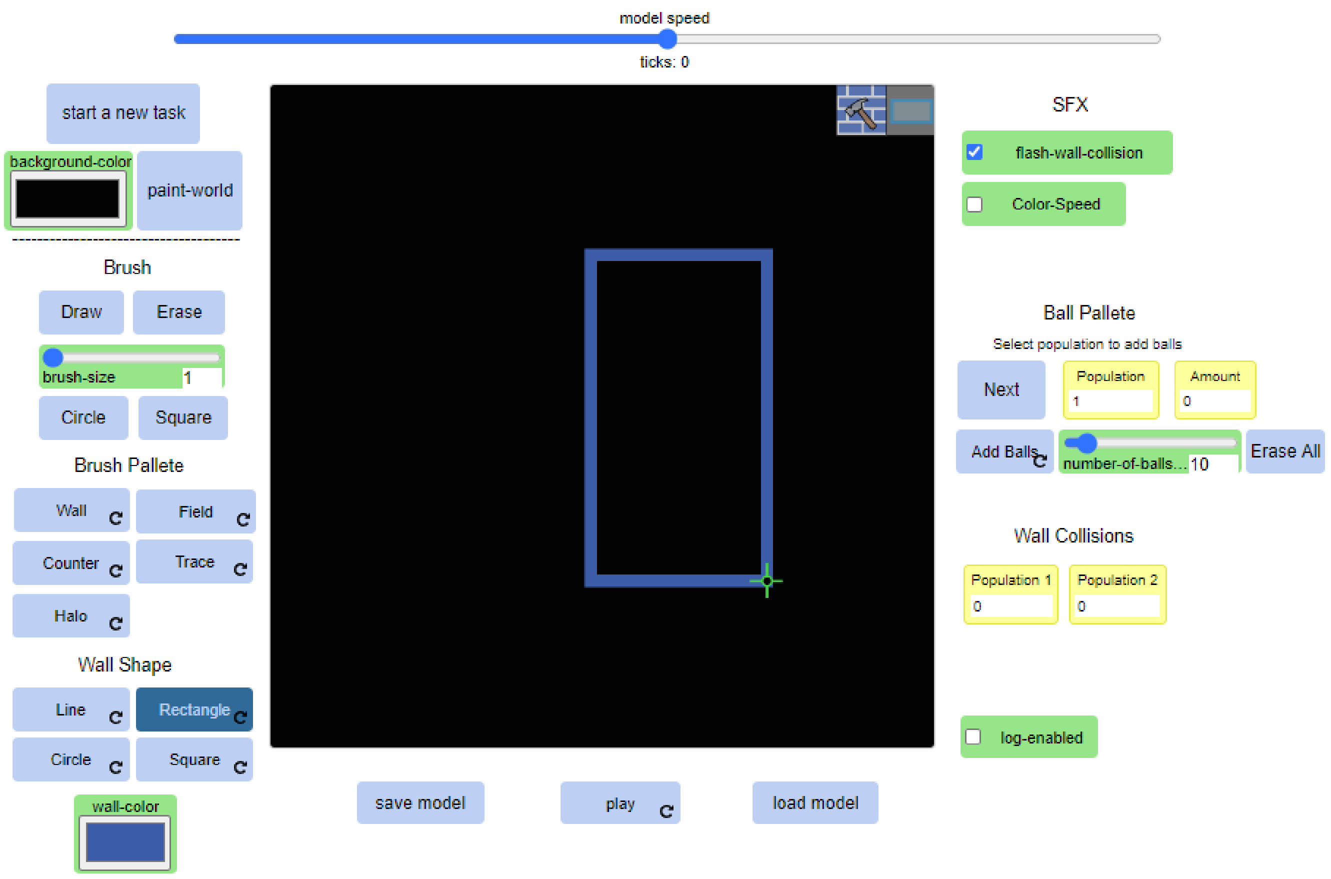
6.5.1. Pressure

1. The project is opened up in a browser and an initialized model appears. The *initialize* world button is the only button that is clickable, let's click on that and see what happens!

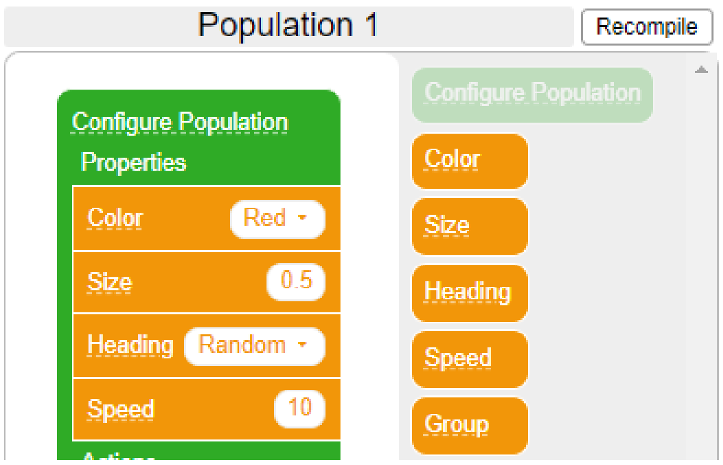


2. The model is empty, with no entities created nor any rules defined to govern them. In this example we plan to model a physical system of particles inside a container with the aim of understanding how pressure is related to the density of particles.

In order to construct this model, we will fist create the container that will have the air particles trapped inside it. We will do so by drawing "walls" with the brush in a rectangular shape.

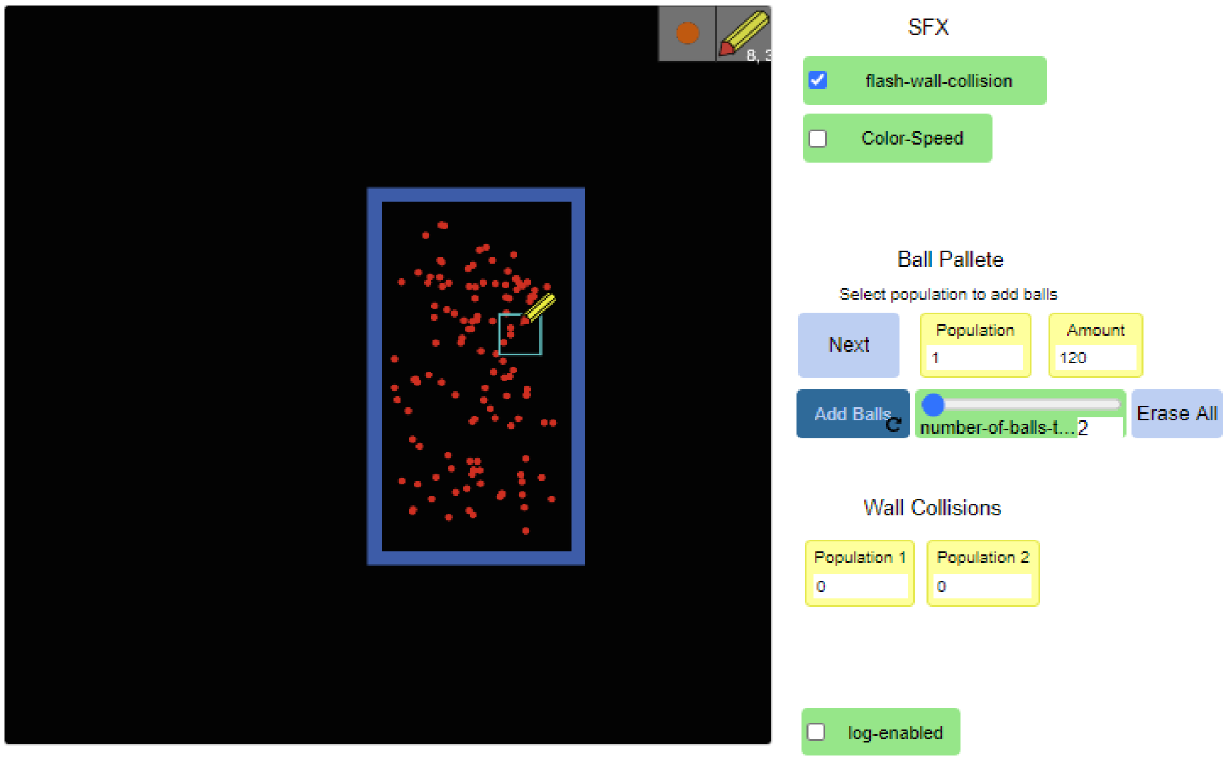


3. Now to simulate the gas particles inside the container, we will need to create balls inside the container which will represent the air particles. But before we create them, we will first define their properties. In this example we will only be using a single population for the simulation, thus we shall define the properties of the first population in the block interface.



Configure Population
Properties
Color Red
Size 0.5
Heading Random
Speed 10

4. Now we will create the balls inside the container using the brush.


SFX

☑ flash-wall-collision

☐ Color-Speed

Ball Pallete

Select population to add balls

Next	Population	Amount
	1	120

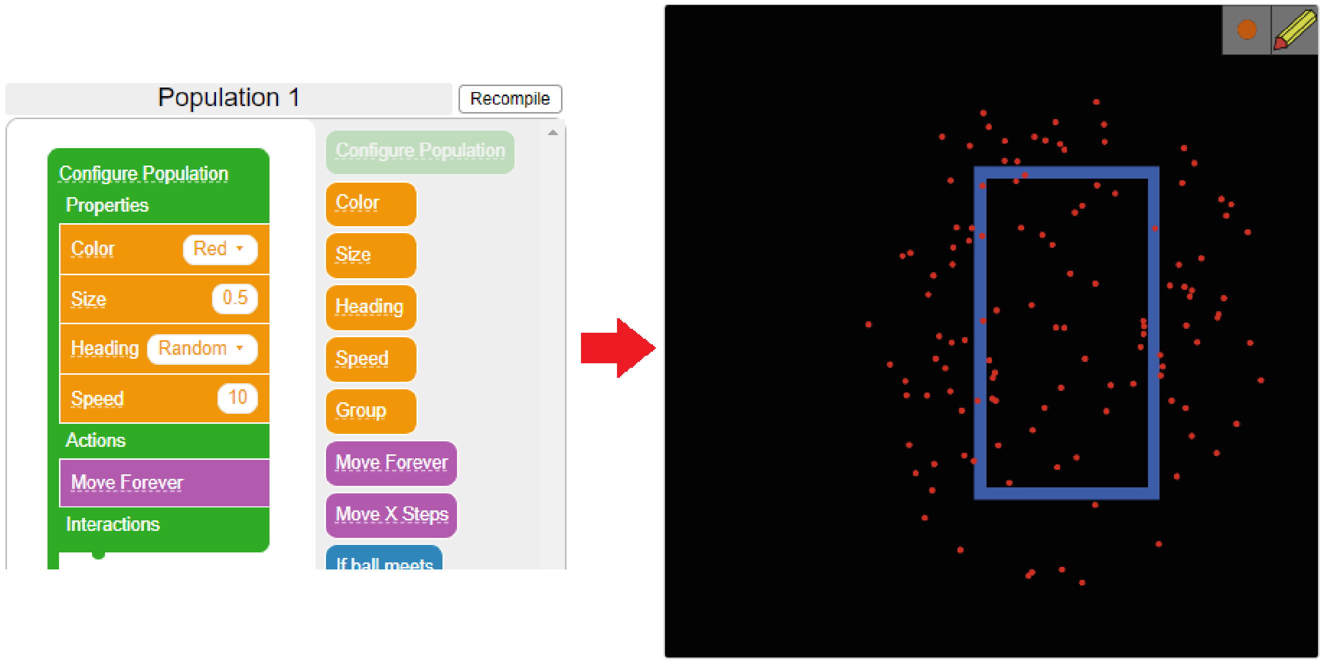
Add Balls — number-of-balls-t… 2 — Erase All

Wall Collisions

Population 1	Population 2
0	0

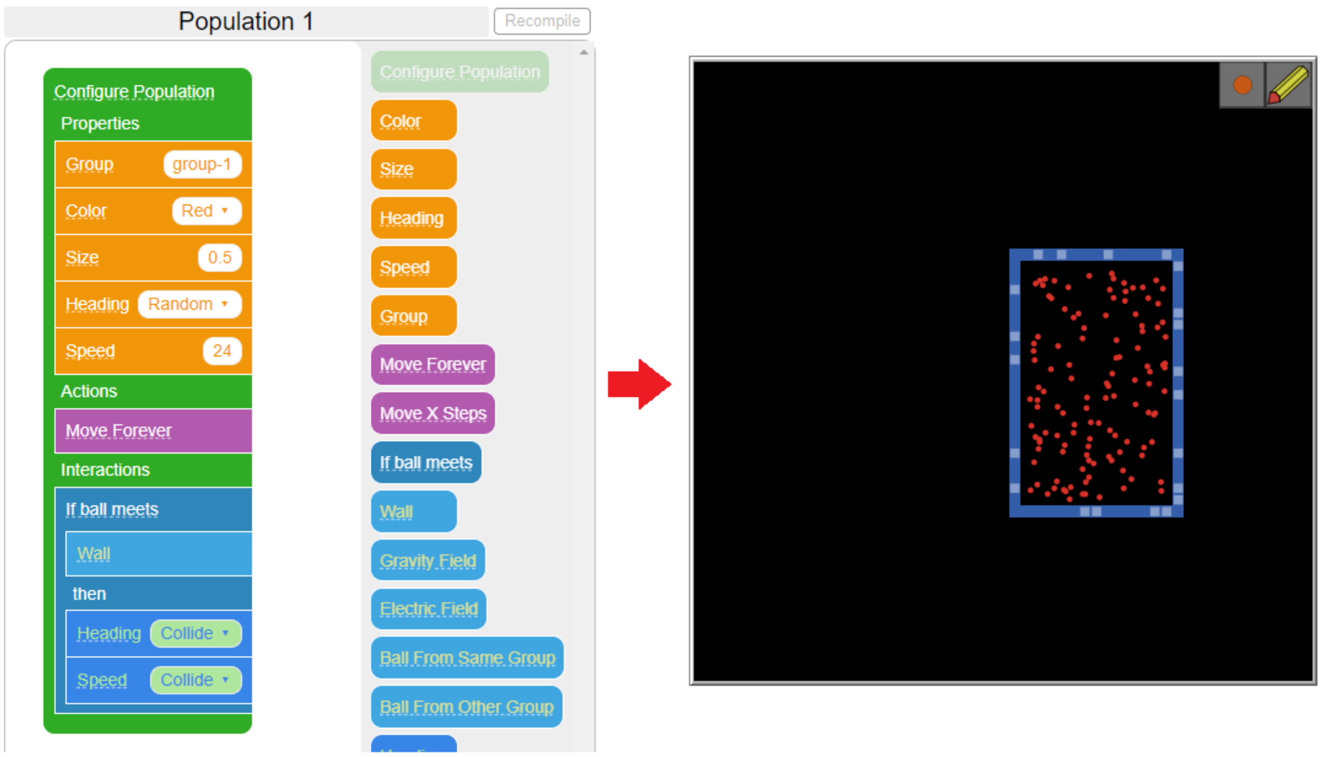
☐ log-enabled

5. Now lets run the model and see what happens. Nothing. The balls aren't moving. This is because we have not defined our *actions* in the block interface to move our balls. To do so we will drag the *Move Forever* block into the *Actions* slot and then run the model.


Population 1 — Recompile

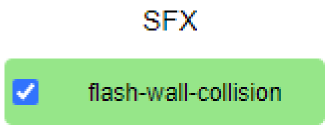
Configure Population
Properties
Color: Red
Size: 0.5
Heading: Random
Speed: 10
Actions
Move Forever
Interactions

6. The balls move right through the container! We want them to stay contained inside the container and collide with the container walls from the inside. The reason they did not interact with the container walls as if they didn't exist is because we have not yet defined an interaction for our ball population when it meets a wall. We will first define a collision interaction with walls in the block interface, then erase all balls and redraw them inside the container and run the model.



Population 1

Configure Population
Properties
Group group-1
Color Red
Size 0.5
Heading Random
Speed 24
Actions
Move Forever
Interactions
If ball meets
Wall
then
Heading Collide
Speed Collide

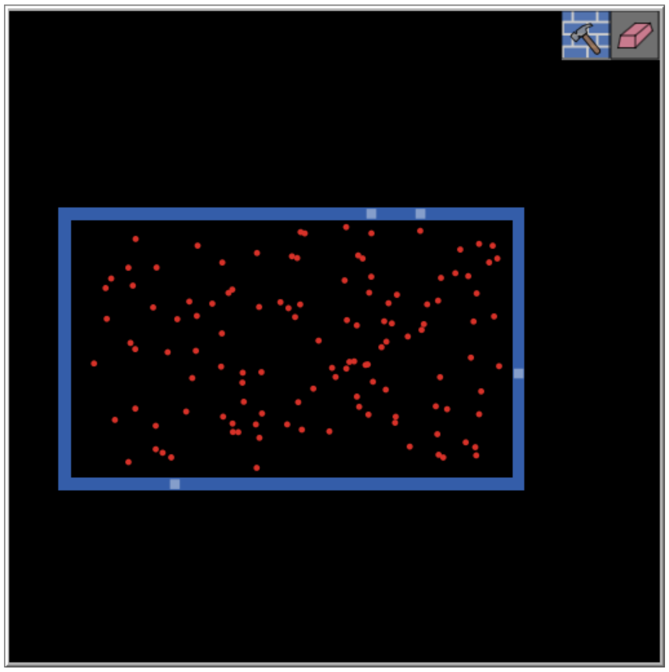


SFX

☑ flash-wall-collision

7. The balls are now colliding against the container walls. The wall collisions can be observed due to the flashes that appear when a ball collides with a wall. This feature can be toggled on or off with the *flash wall collision* widget.

8. The next part of the experiment is observing what happens when we make the container bigger, thus lowering the ball density inside the container.



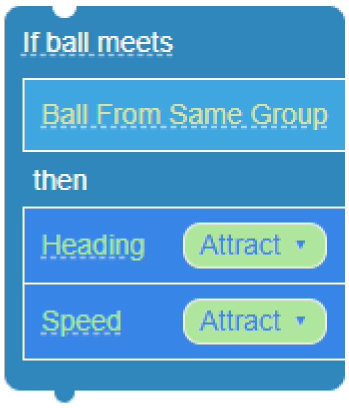
9. There are significantly less wall collisions in the bigger container than the former smaller one. This can be observed by the lower amount of flashes that indicate a wall collision. Since there are less

collisions of particles inside the bigger container, it means that the pressure is lower inside the container. What we can learn from the system that we just modeled is that pressure is a function of both the amount of particles and the size of area they are in.

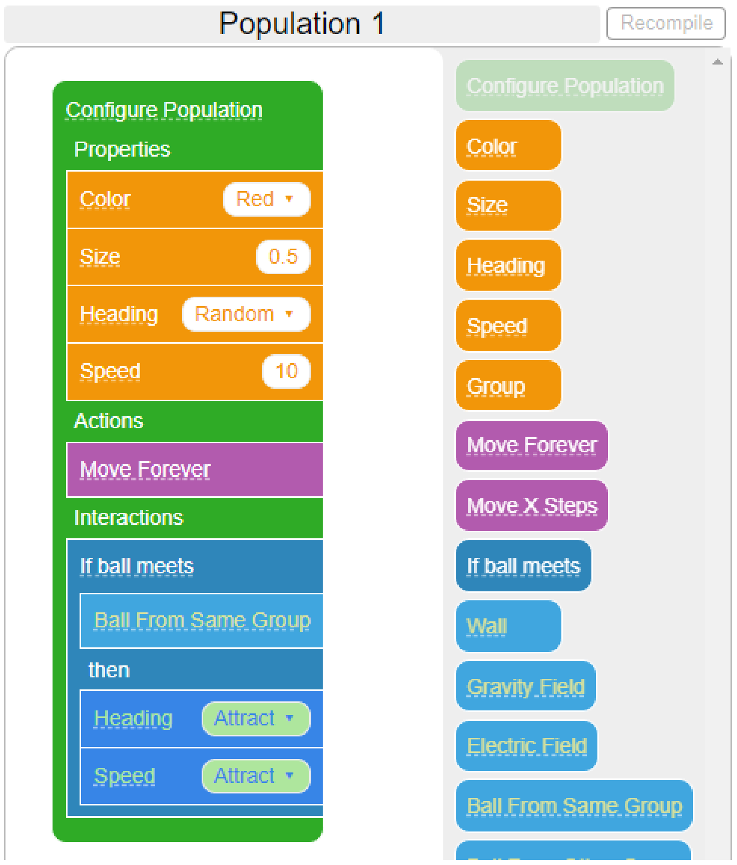
6.5.2. Solar System

In this example we will model a the formation of a solar system. The entities in this model will be bodies of mass which will be represented by balls, and they will interact with each with each other by means of gravity (attraction force). This example will be more brief than the former.

1. We need to define the *interactions* of the balls so that they gravitate towards each other in order to simulate bodies of mass forming a solar system. This is a form of attraction force, so we will define the ball population in this model to interact with itself by means of attraction.


If ball meets
Ball From Same Group
then
Heading Attract
Speed Attract

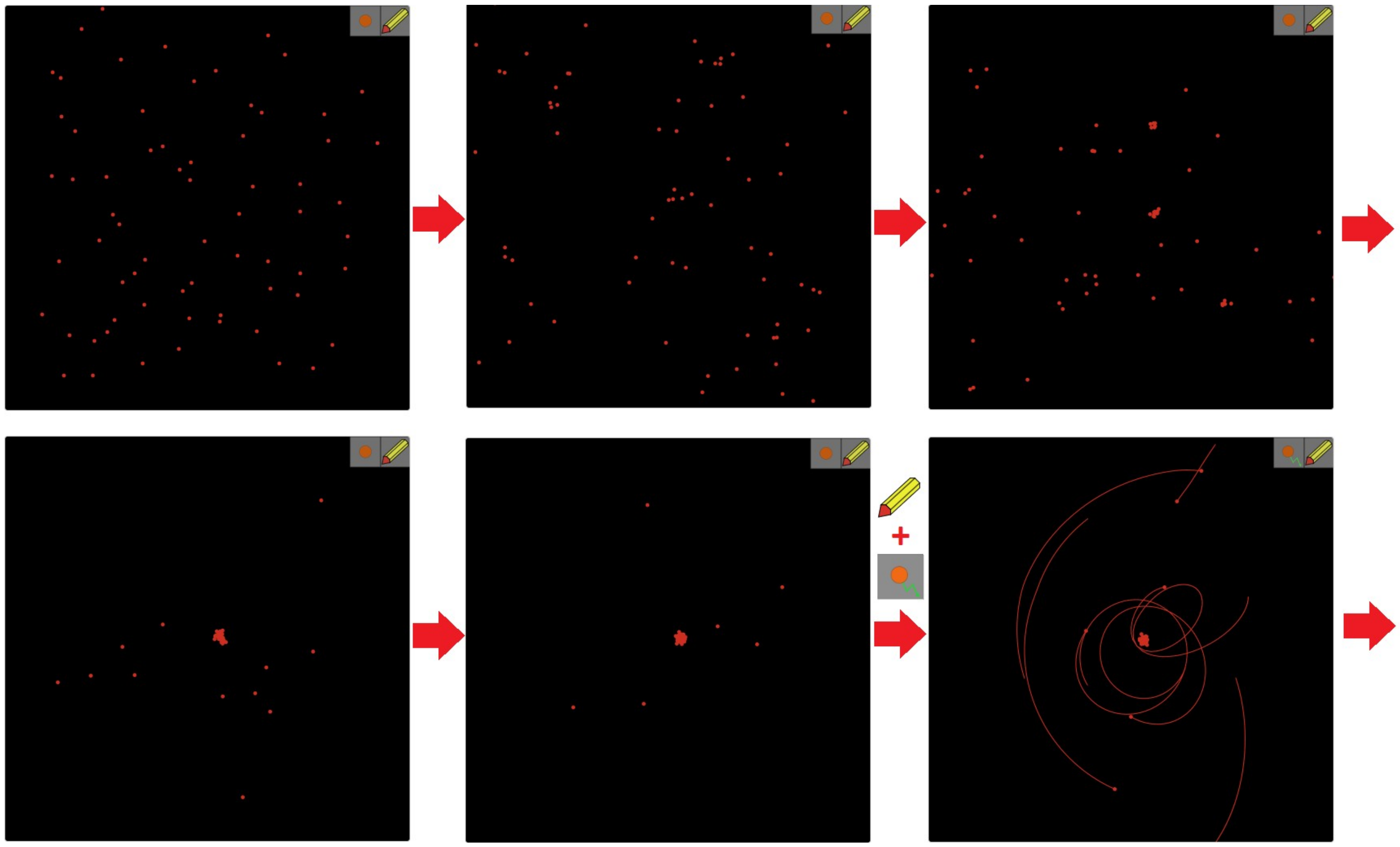
2. Now we have left to define the *actions* and *properties* in the blocks interface. The *actions* is trivial, *Move Forever* since the balls need to move. The *properties* will be set to have default values since they won't make too much difference. The *size* property is also the mass of the ball, so the bigger the mass the bigger the attraction force.


Population 1

Recompile

Configure Population
Properties
Color Red
Size 0.5
Heading Random
Speed 10
Actions
Move Forever
Interactions
If ball meets
Ball From Same Group
then
Heading Attract
Speed Attract

3. Now we need to create the bodies of mass in the universe. We will add balls using the brush, run the model and see the solar system develop over time.



The trace tool was used to let us observe the "planets" (balls) that orbit the center of mass.

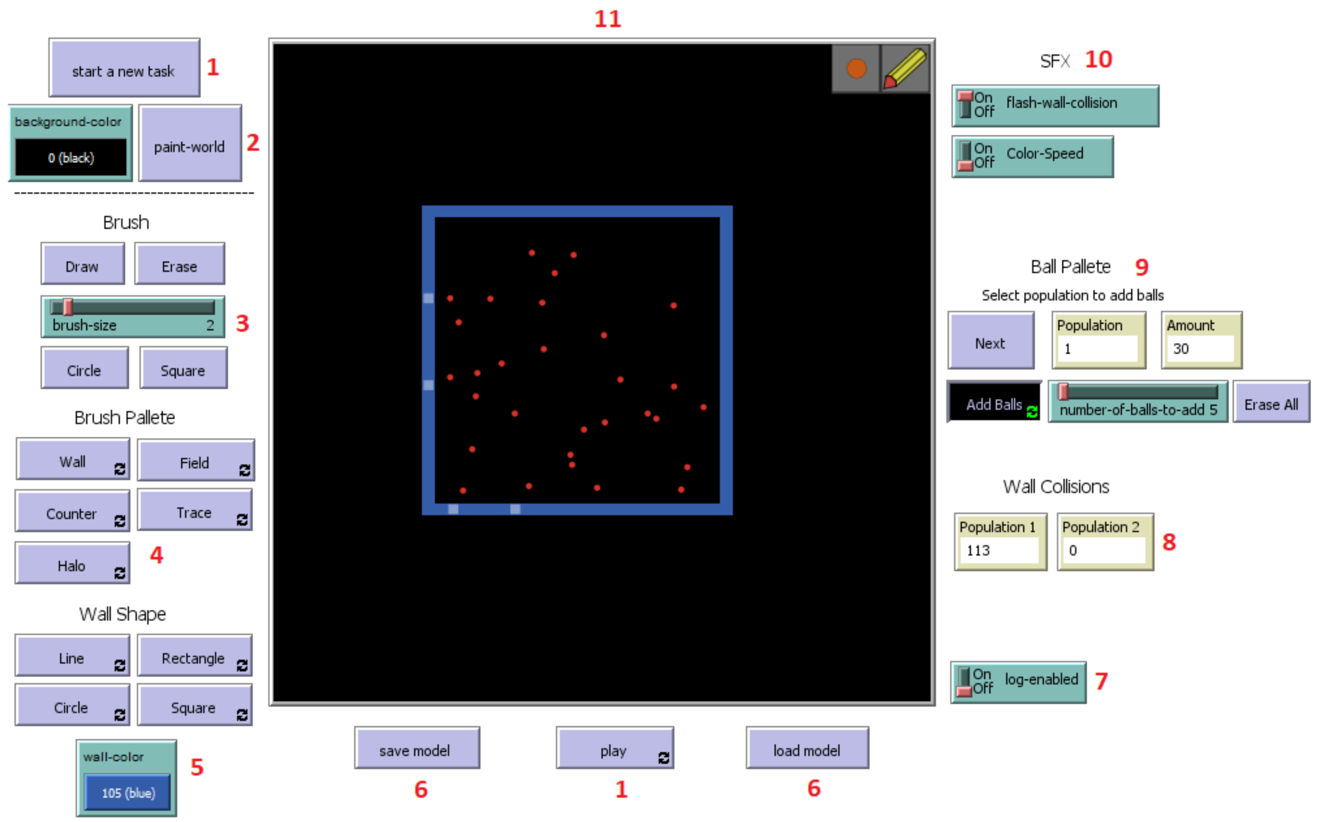
In this example we were able to simulate a model of a forming solar system and observe how gravity affects it.

6.6. User Guide

This chapter is a guide to using all functionalities offered in this project.

6.6.1. Model interface sections

The model interface is organized in sections that group common widgets together. The sections in the picture below are numbered.



background-color 0 (black)

brush-size 2

wall-color 105 (blue)

Population 1

Amount 30

number-of-balls-to-add 5

Population 1 113

Population 2 0

1. Start/Stop model
2. Background
3. Brush configuration
4. Brush pallet
5. Wall shape
6. Save/Load world state
7. Logging
8. Wall collision monitor

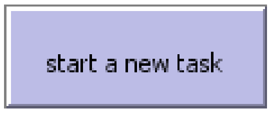
9. Ball Pallet
10. SFX
11. World view

6.6.2. Start/stop model

Controls used to start, stop and initialize model.

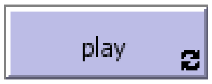
6.6.2.1. Initialize

Resets the world to an initialized state.



6.6.2.2. Run/Pause

Run/stop execution of model.

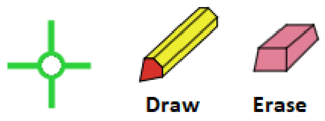


6.6.3. Brush

The brush can be used to add/remove entities in the model. It is used like a brush, by “painting” with it in the world view. The user may configure the brush’s shape and size.

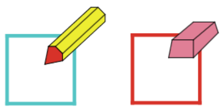
6.6.3.1. Cursor

The brush cursor is displayed in the world view as either a pencil drawing or an eraser erasing. If the brush is currently used to draw a wall shape, then a cross hair cursor will be displayed.



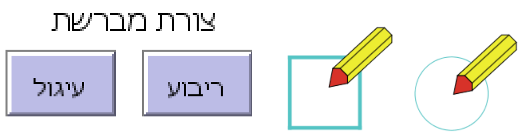
6.6.3.2. Outline

The brush outline encompasses the patches that the brush is painting on. It is colored cyan if it is drawing and red if erasing.



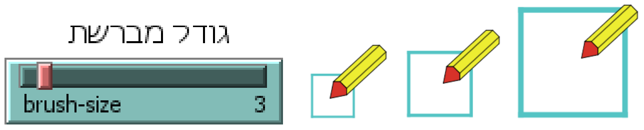
6.6.3.3. Shape

The brush shape can be configured to be either a square or circle. The brush’s shape affects which patches get painted on.



6.6.3.4. Size

The brush size can be configured to paint on a bigger or smaller area.



6.6.3.5. Draw

Set the brush to draw.

6.6.3.6. Erase

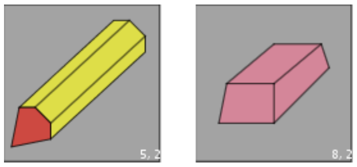
Set the brush to erase.

6.6.3.7. Icons

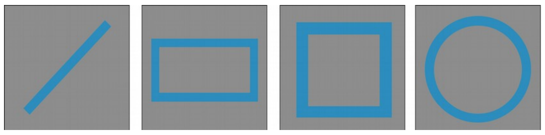
Brush icons are displayed in the top right corner of the world view that indicate what the brush is currently painting, and if the brush is set to draw or erase.

6.6.3.7.1. Draw/Erase icon

Indicates if brush is set to draw or erase. In the bottom right corner of the icon are the brush coordinates.

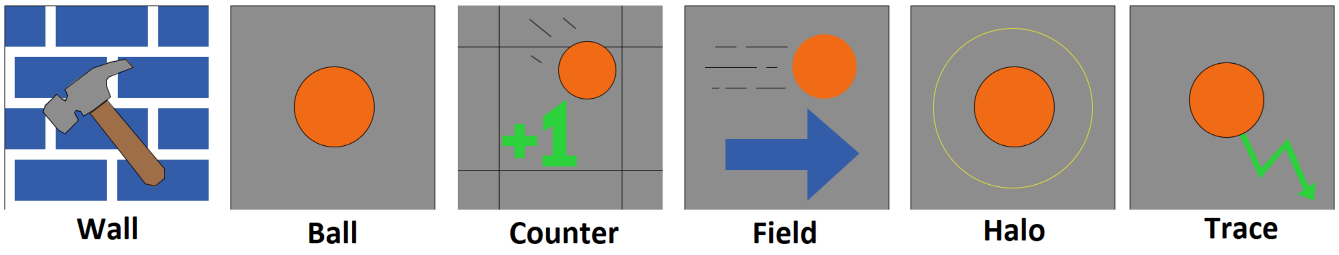


If brush is currently drawing a wall shape, then the shape will be shown as the icon instead of a pencil.

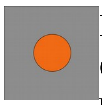


6.6.3.7.2. Brush type icon

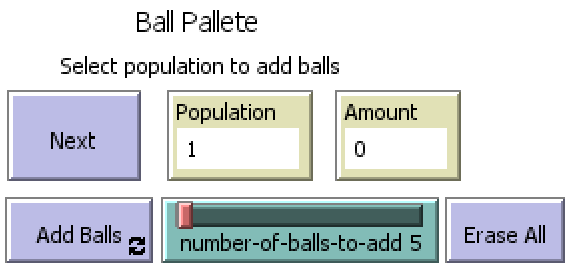
Indicates what the brush is painting.



6.6.3.8. Add/Remove balls



Balls can be added or removed using the brush. Clicking on "Add Balls" will set the brush to adding balls, and clicking anywhere inside the world will create them. Selecting which population of balls to create is done by clicking "Next", it switches between populations. It is possible to erase balls in a certain location with the brush or remove a whole population with the "Erase All".



6.6.3.9. *Add/Remove walls*

Wall can be created or removed with the brush. Walls can be drawn free-form or as shapes that are very convenient.

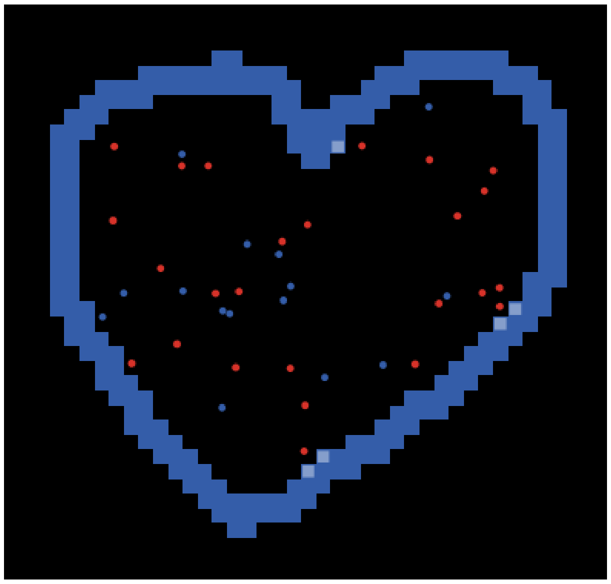
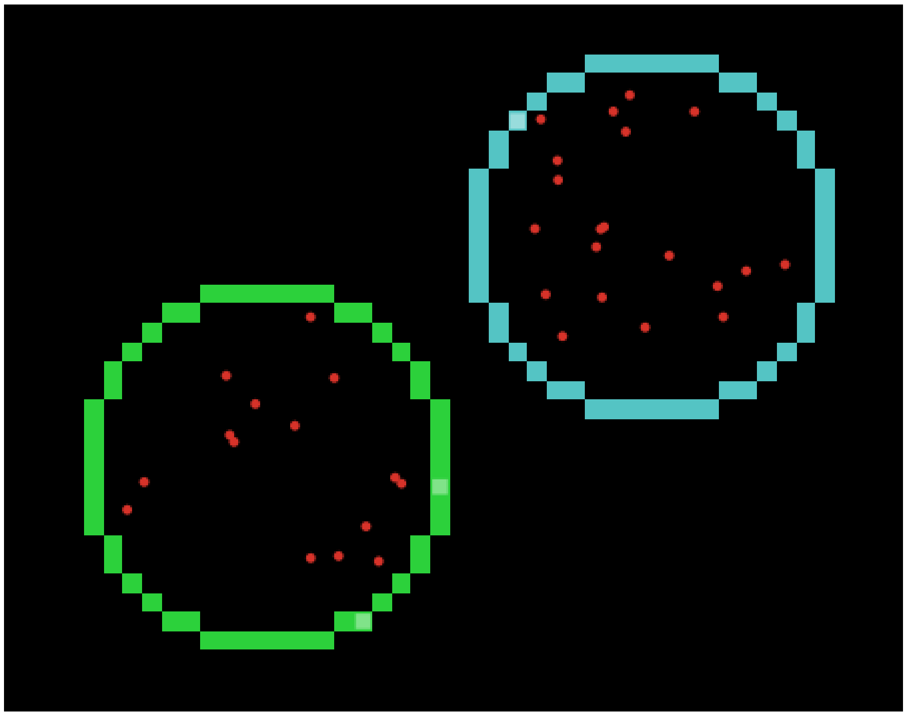


Figure 1: A heart shaped wall drawn with a brush.

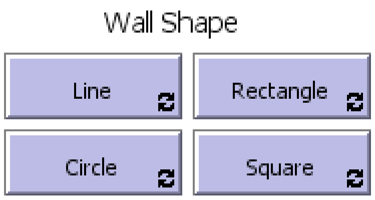
6.6.3.9.1. Set color

The color of wall being drawn can be set with the color chooser widget.



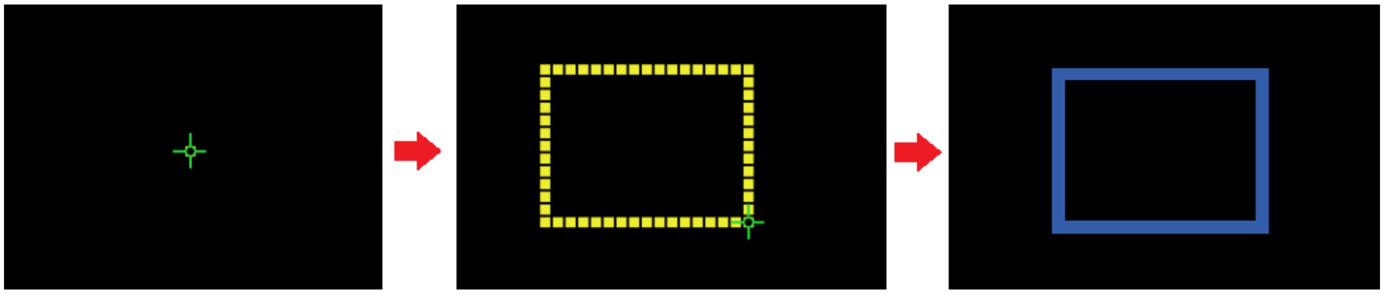
6.6.3.9.2. Wall shapes

The model supports drawing 4 shapes: Rectangle, Circle, Square and Line.



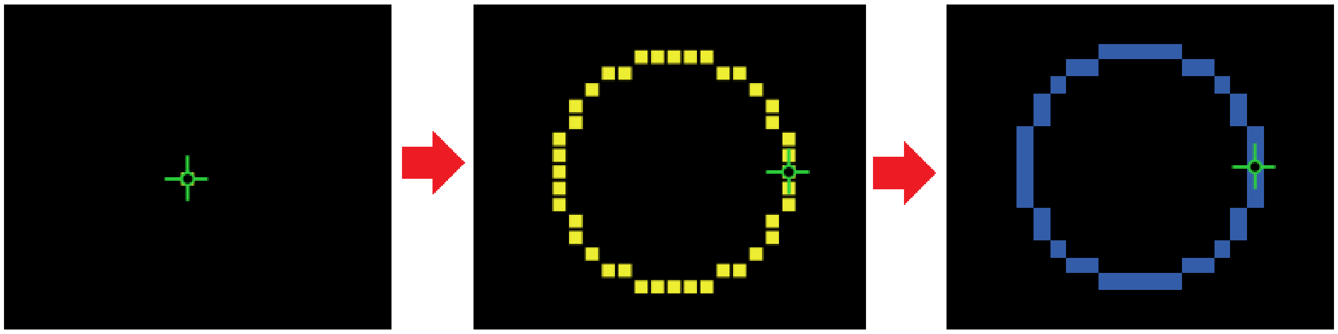
6.6.3.9.2.1. Rectangle

Rectangle shaped walls can be created with the brush. When the mouse is first clicked it sets the center of the rectangle, dragging brush around while pressed down changes dimensions of rectangle (highlighted in yellow). Releasing mouse creates the rectangle shaped wall. Sides are thickness of 1 patch.



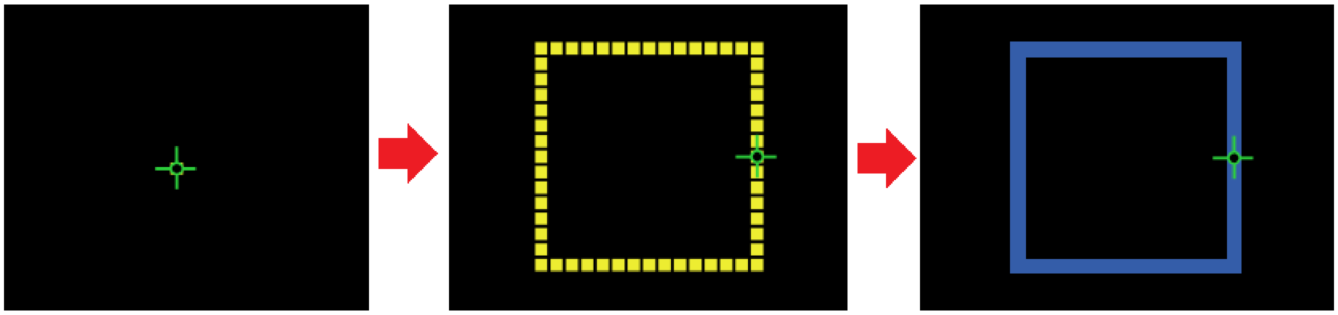
6.6.3.9.2.2. Circle

Create a circle shaped wall with brush. When mouse is first clicked it sets the center of the circle, dragging brush around while pressed down changes dimensions (radius) of circle (highlighted in yellow). Releasing mouse creates the circle shaped wall.



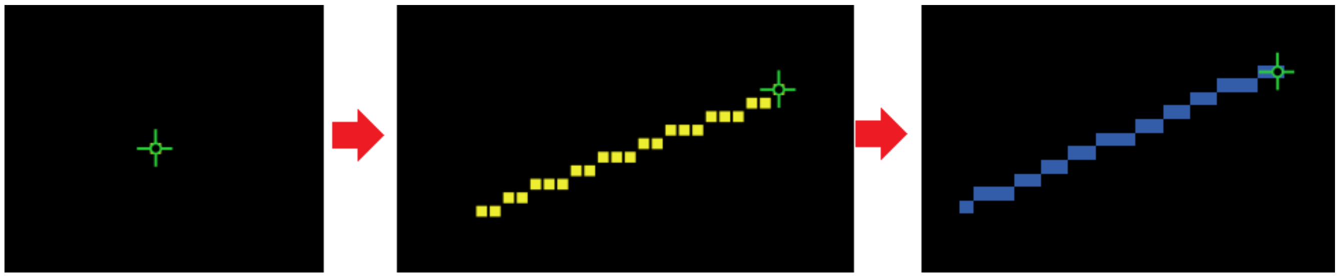
6.6.3.9.2.3. Square

Create a square shaped wall with brush. When mouse is first clicked it sets the center of the square, dragging brush around while pressed down changes dimensions of square (highlighted in yellow). Releasing mouse creates the square shaped wall. Sides are thickness of 1 patch.



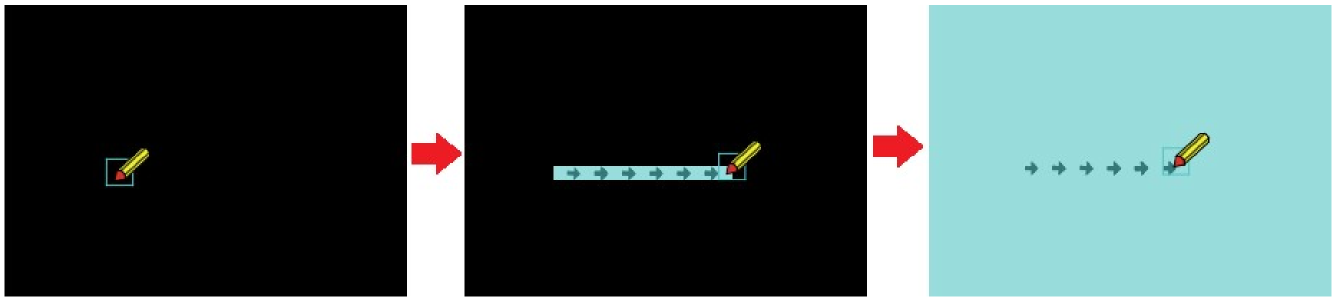
6.6.3.9.2.4. *Line*

Create a wall line with brush. When mouse is first clicked it defines the starting point of the line, when mouse is released the end point. Line is highlighted in yellow while it is being configured.



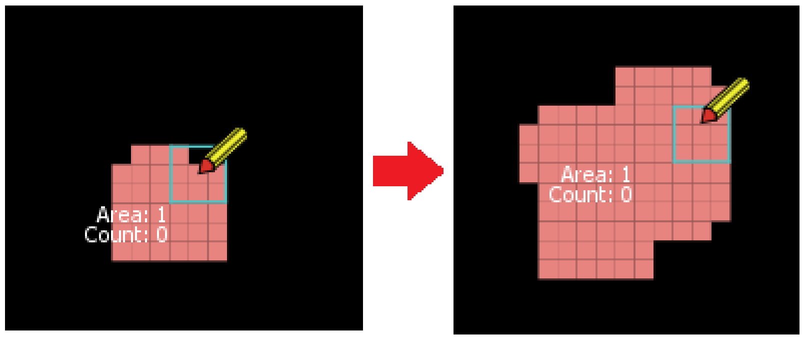
6.6.3.10. *Add/Remove field*

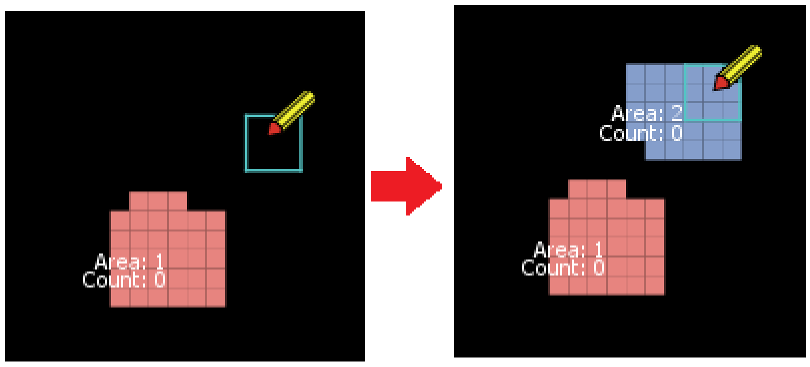
Fields can be created with the brush by pressing down and dragging brush to set the direction of field. Remove electric field bet setting brush to erase and clicking on a field.



6.6.3.11. *Add/Remove counter*

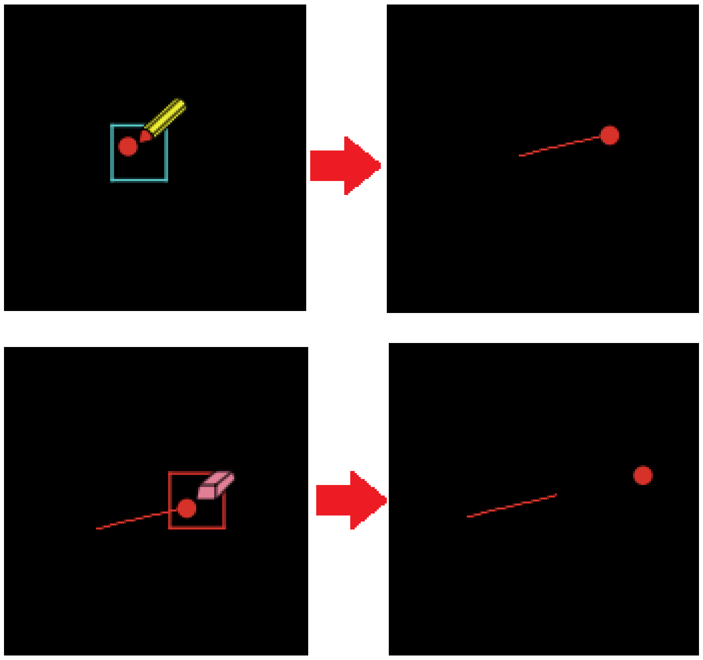
Counters can be added or removed with the brush. If brush was drawing on patches that contain an existing counter when first pressed down, then it will add that counter to the patches it is drawing on. Otherwise it will create a new counter. Different counters have different colors.





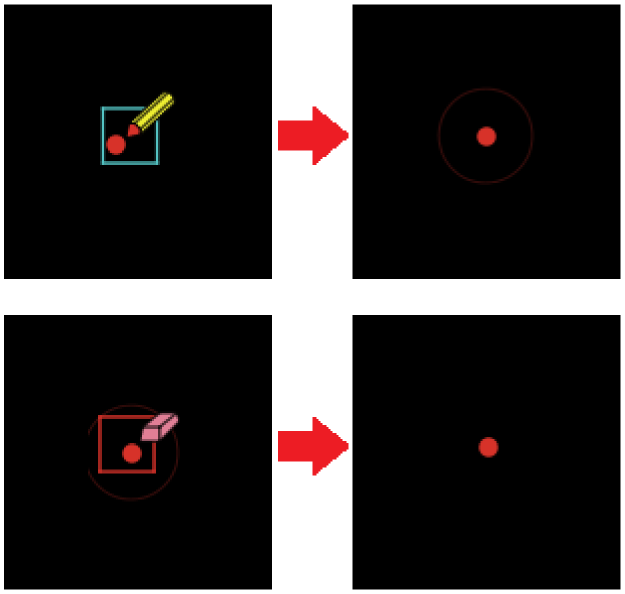
6.6.3.12. *Add/Remove trace*

Balls can be set to start or stop tracing the path they move in using the brush.



6.6.3.13. *Add/Remove halo*

Halos can be added or removed to balls with the brush. Balls will have a halo set around it to help distinguish it from others.



6.6.4. Save/Load world state

6.6.4.1. Save

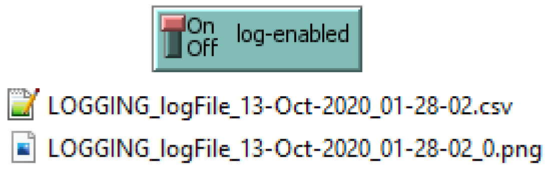
Save current world state to a file.

6.6.4.2. Load

Load world state from a file.

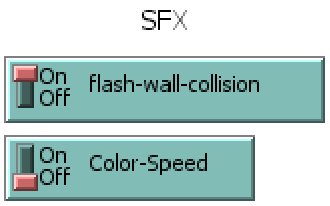
6.6.5. Logging

The model features logging a users interactions with the model. Every time the brush is used or the model is initialized the log gets updated. A new log file is generated and downloaded to the browser's default download folder every time the log changes. It logs the ball populations physical properties, entities were drawn with the brush, what time it happened and more. Pictures of the current model state are also logged (PNG format) after every time the brush is used or the model is initialized. It is possible to enable or disable logging with a switch widget.


On
Off
log-enabled
LOGGING_logFile_13-Oct-2020_01-28-02.csv
LOGGING_logFile_13-Oct-2020_01-28-02_0.png

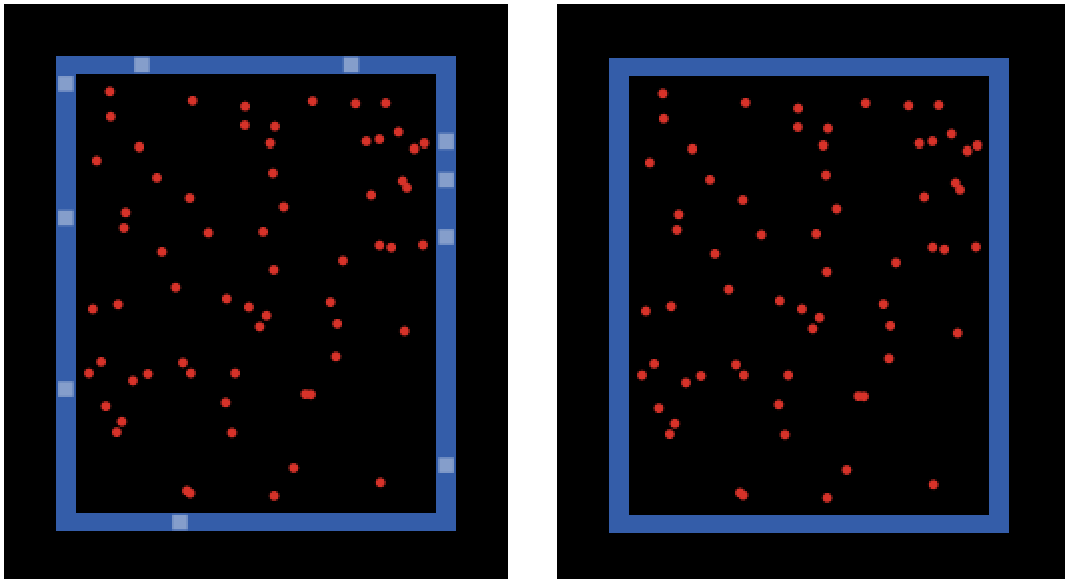
6.6.6. SFX

The SFX section has widgets relevant to visual aspects of the model



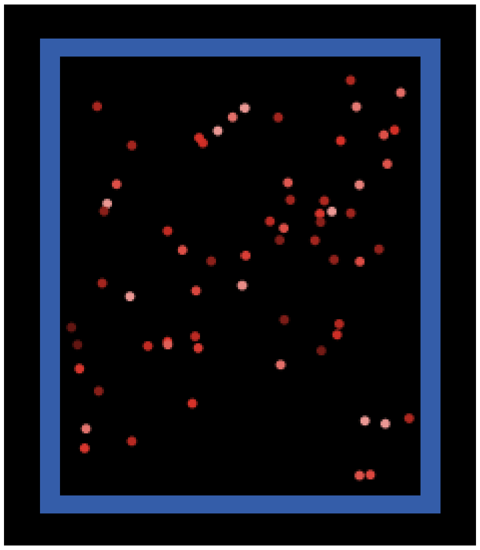
6.6.6.1. Flash wall collisions

Toggle to enable/disable flashing wall collisions. This is useful if in certain models there is a need to observe and approximate the rate of wall collisions.



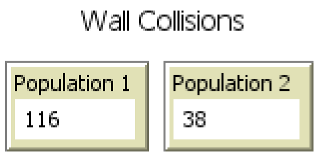
6.6.6.2. Color ball speed

Balls are colored in a brighter or darker shade according to their speed. The brighter the color the faster the ball is relative to the rest of the balls in the population.



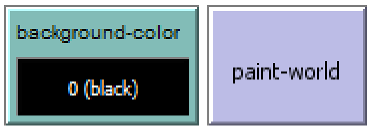
6.6.7. Wall collision monitor

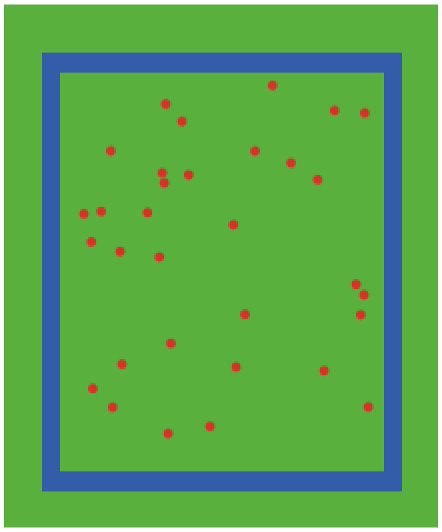
Monitor the amount of collisions per population.



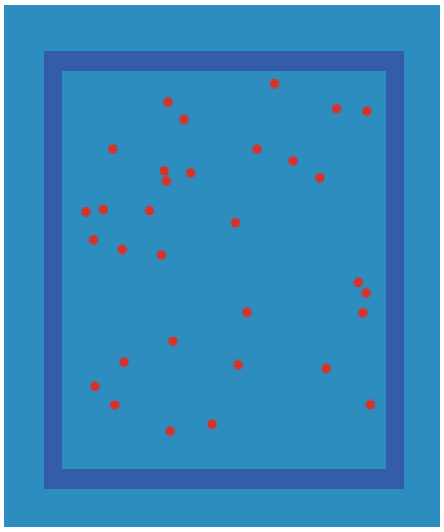
6.6.8. Background

The background color can be set in a color chooser widget and clicking the "paint-world" button.





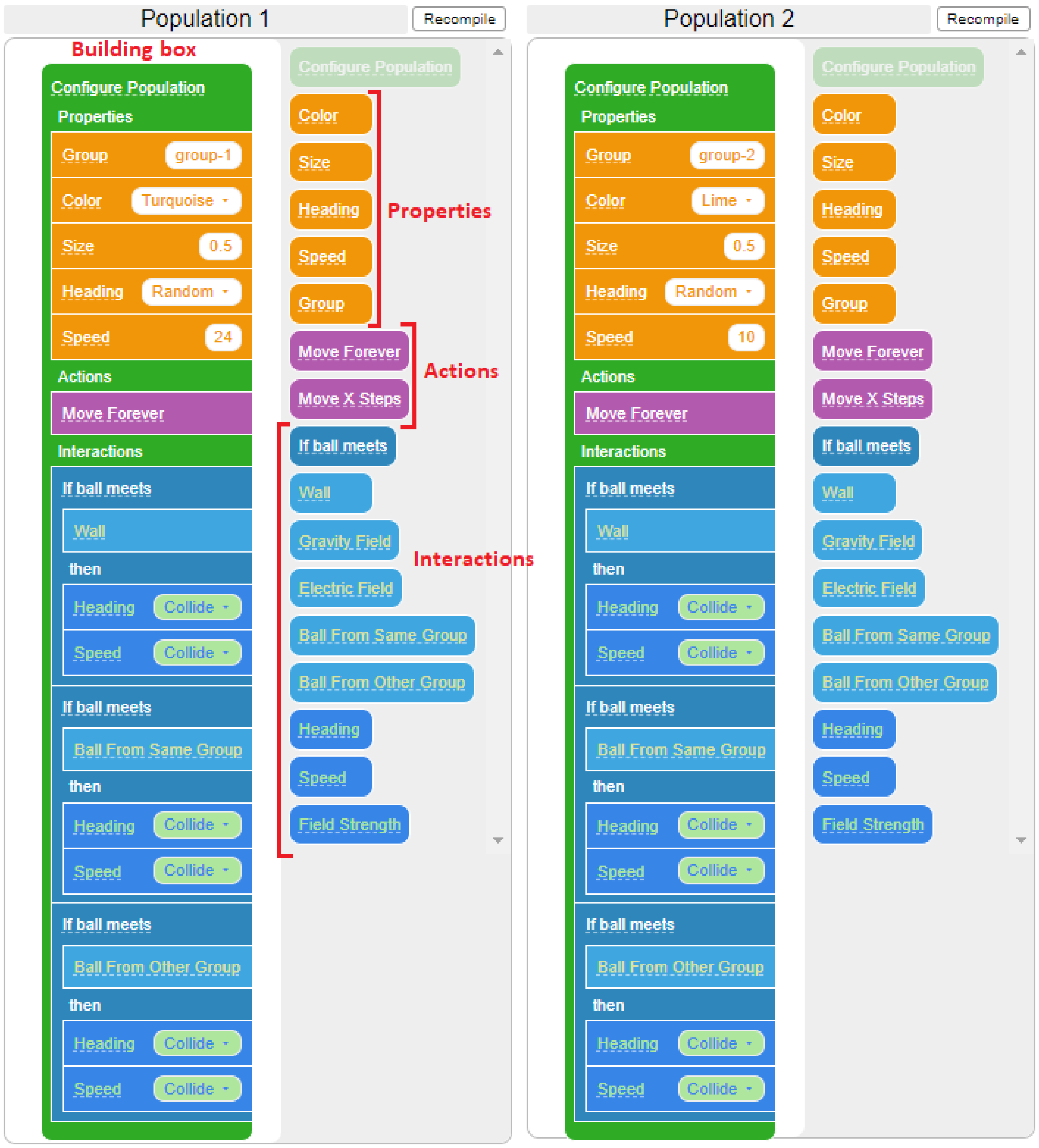
Green



Blue

6.7. Block-based programming interface

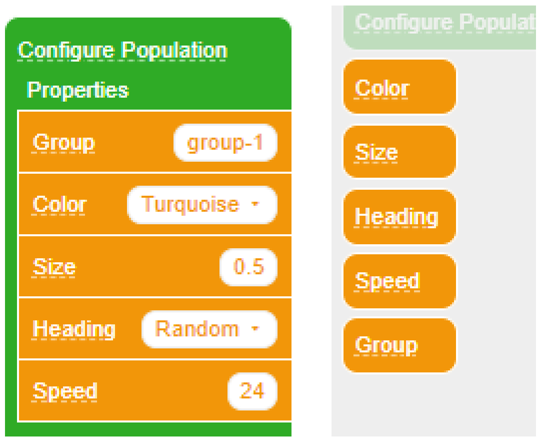
The block-based programming interface contains two identical block spaces, one for each population to define their physical properties. The blocks are color coded to fit into one of the three categories of: **properties**, **actions**, **interactions**. In order to define a specific physical property, the blocks need to be dragged into the building box (area where blocks are arranged inside the block space). If a block that defines a physical property is absent, meaning it was not added to the building box, then a default value for that physical property is set. For example, if the "color" block is not used, then the populations color is defaulted to gray.



```
Population 1
Configure Population
  Properties
    Group      group-1
    Color      Turquoise
    Size       0.5
    Heading    Random
    Speed      24
  Actions
    Move Forever
  Interactions
    If ball meets
      Wall
    then
      Heading  Collide
      Speed    Collide
    If ball meets
      Ball From Same Group
    then
      Heading  Collide
      Speed    Collide
    If ball meets
      Ball From Other Group
    then
      Heading  Collide
      Speed    Collide

Population 2
Configure Population
  Properties
    Group      group-2
    Color      Lime
    Size       0.5
    Heading    Random
    Speed      10
  Actions
    Move Forever
  Interactions
    If ball meets
      Wall
    then
      Heading  Collide
      Speed    Collide
    If ball meets
      Ball From Same Group
    then
      Heading  Collide
      Speed    Collide
    If ball meets
      Ball From Other Group
    then
      Heading  Collide
      Speed    Collide
```

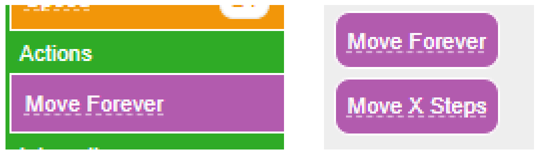
6.7.1. Properties

The blocks for setting properties are color coded *yellow*. They should be dragged into their allotted "Properties" clause in the "configure population" block. If a property block is not placed, then a default value will be assigned to that property.



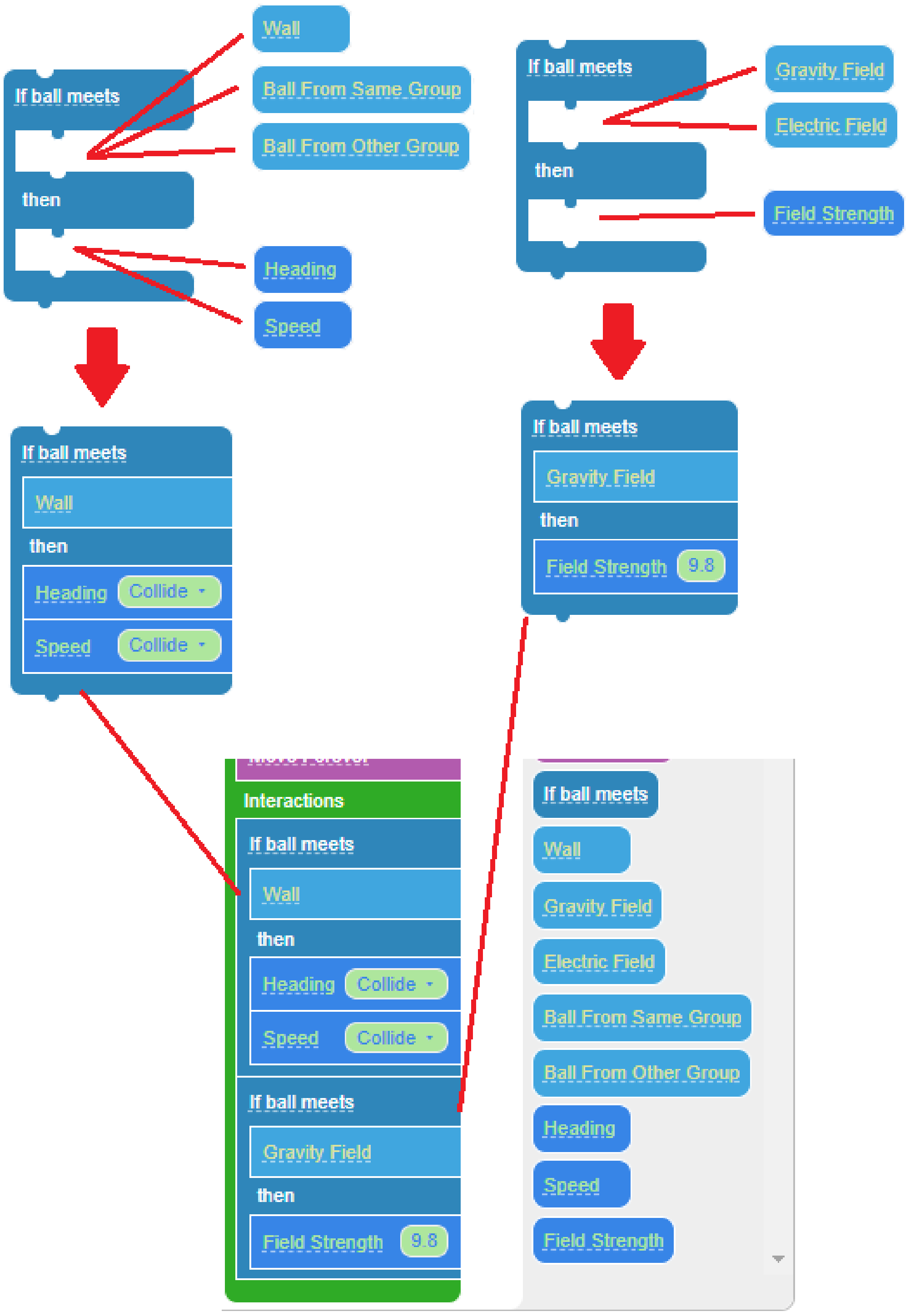
6.7.2. Actions

The blocks for setting actions are color coded *purple*. They should be dragged into their allotted "Actions" clause in the "configure population" block. If an actions block is not placed, the ball population defaults to not moving. Only one of the actions blocks should be placed (NetTango currently does not support setting these type of limits). If more than one block is placed, only the last block is taken into account.



6.7.3. Interactions

The blocks for setting actions are color coded *blue*. The way to define an interaction with the blocks is by first dragging an "if ball meets" block and filling it with blocks that complete the interaction. The top slot is allocated for the entity the ball interacts with, and the bottom slot is for defining the change that occurs to the ball. A field interaction needs to define the field strength of the interaction, and a wall/ball interaction defines the heading and speed change.



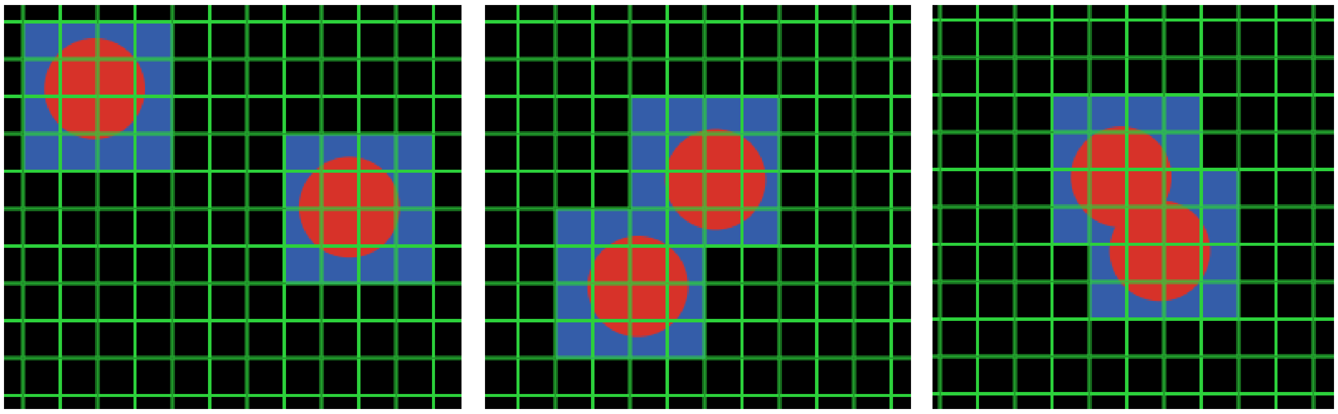
If an interaction with an entity is not defined in the blocks interface, then the default is that the entities do not interact. If for example the wall interaction was not defined, the ball would not interact with the wall, it would pass through it.

6.8. Algorithms

This chapter describes the main algorithms used in the computational model programmed in NetLogo.

6.8.1. Object collision detection

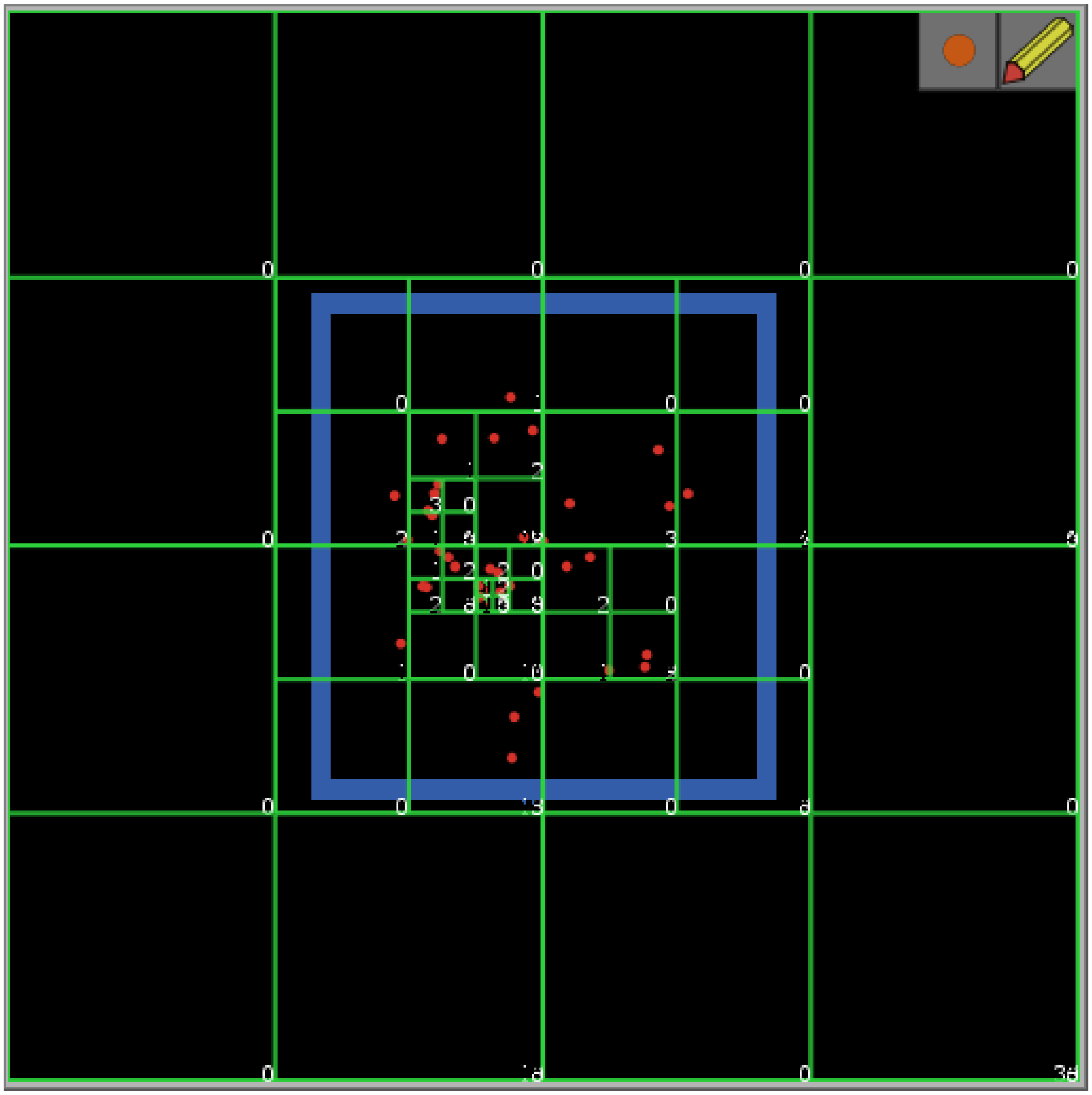
The algorithm to detect ball collision uses grid spacial hashing. NetLogo natively supports grid spatial hashing since the world is a 2D grid which keeps track of the objects present in each patch. The algorithm checks for collisions only with other balls that their bounding box is contained inside patches the balls own bounding box is in. If the bounding boxes are in range of then detection occurs, which checks if the distance between the balls is smaller than the sum of their radius. This algorithm greatly reduces the amount of calculations needed to check for collision between all balls.


No detection no collision **Detection but no collision** **Detection and collision**

6.8.2. Barnes-Hut simulation

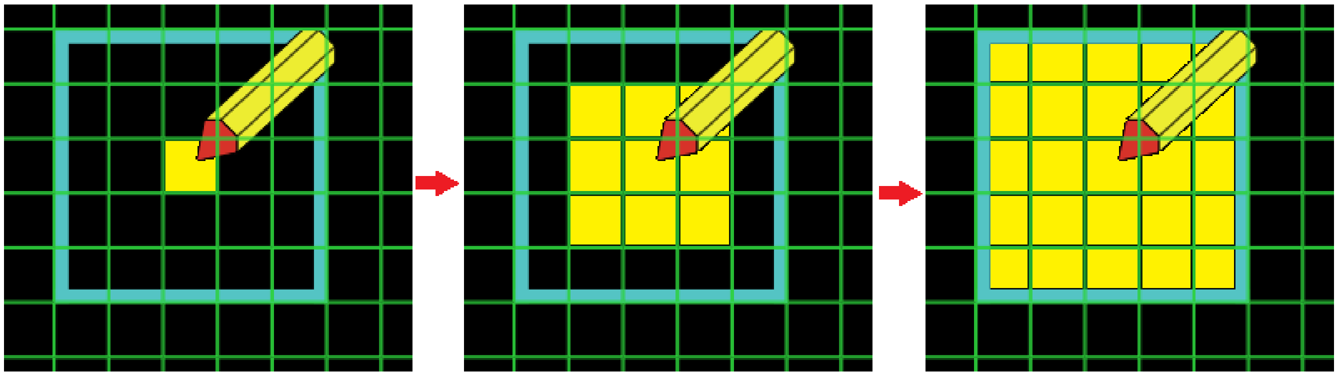
The Barnes-Hut algorithm is used to solve N-body problems that require calculating forces acting upon bodies. The algorithm estimates with with a certain precision the forces acting upon a body in O(log n) time. The algorithm divides the 2D space into cubic cells via a quadtree so that a specific entity only calculates forces of other individual entities if they are from nearby cells, and entities in distant cells are treated as a large entity centered the cell's center of mass, reducing calculation to a single entity that averages the mass of all other entities in that cell.

* Below is a visualization of the Barnes-Hut algorithm divides up the 2D world into cubic cells via a quadtree.



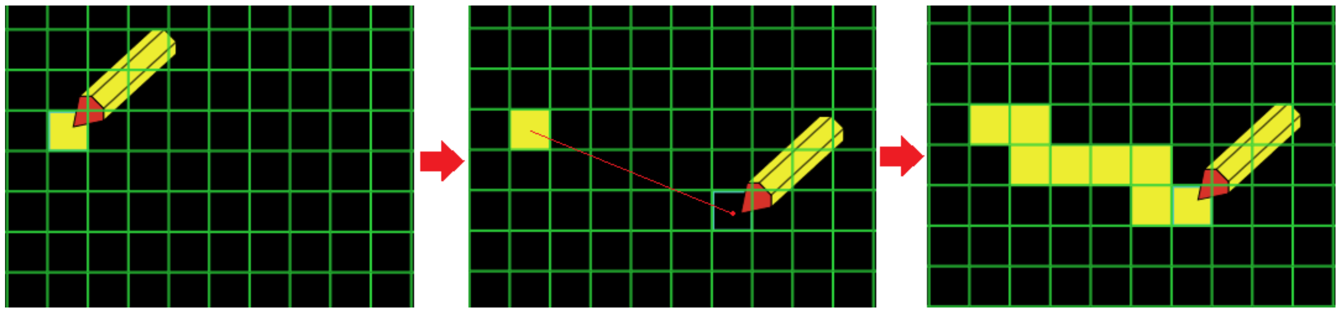
6.8.3. Creating walls and moving balls aside

When a wall is created on a patch that is occupied by balls, if the balls can be moved aside to a neighboring unblocked patch they will be relocated and a wall created in their place. In order to achieve this fluidly without trapping balls inside the newly created batch of walls, the wall are created in order of inside out from the center, with the center being the patch that was directly under the brush. It is important to create walls this way so that any balls will be pushed out without getting trapped.



6.8.4. Painting with brush without skipping patches

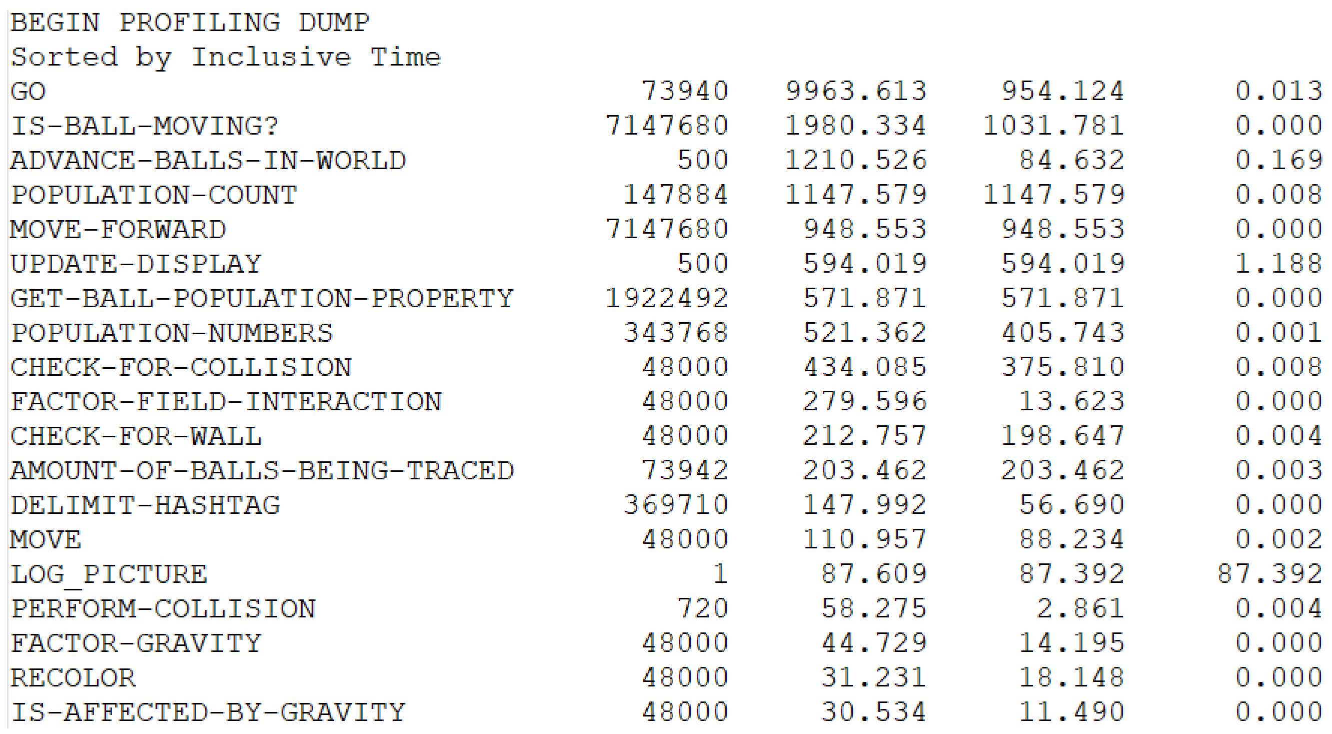
Brush position gets updated discretely (depending how fast NetLogo is updating), so it might jump ahead a few patches from its current position, so it is necessary to calculate the patches the brush skipped over and paint them as well with the brush. This is done by calculating the intersections of the world grid with line between current and last brush position.



6.9. Optimization

NetLogo comes with a built-in *profiler* extension that enables the programmer to time procedure execution on the model and assess where the bottlenecks are. The profiler is an indispensable tool and critical to have made this model work fluidly. The profiler was widely used throughout the project to make sure the model can run simulations without slowing down.

Many of the algorithms and coding was refactored a number of times with the help of the profiler to minimize slowdowns.



```
BEGIN PROFILING DUMP
Sorted by Inclusive Time
GO                                 73940    9963.613    954.124      0.013
IS-BALL-MOVING?                  7147680    1980.334   1031.781      0.000
ADVANCE-BALLS-IN-WORLD               500    1210.526     84.632      0.169
POPULATION-COUNT                  147884    1147.579   1147.579      0.008
MOVE-FORWARD                     7147680     948.553    948.553      0.000
UPDATE-DISPLAY                       500     594.019    594.019      1.188
GET-BALL-POPULATION-PROPERTY     1922492     571.871    571.871      0.000
POPULATION-NUMBERS                343768     521.362    405.743      0.001
CHECK-FOR-COLLISION                48000     434.085    375.810      0.008
FACTOR-FIELD-INTERACTION           48000     279.596     13.623      0.000
CHECK-FOR-WALL                     48000     212.757    198.647      0.004
AMOUNT-OF-BALLS-BEING-TRACED       73942     203.462    203.462      0.003
DELIMIT-HASHTAG                   369710     147.992     56.690      0.000
MOVE                               48000     110.957     88.234      0.002
LOG_PICTURE                            1      87.609     87.392     87.392
PERFORM-COLLISION                    720      58.275      2.861      0.004
FACTOR-GRAVITY                     48000      44.729     14.195      0.000
RECOLOR                            48000      31.231     18.148      0.000
IS-AFFECTED-BY-GRAVITY             48000      30.534     11.490      0.000
```

Example of a profile dump showing execution times

Examples of optimizations made:

- Recoloring balls was slow due to how it was implemented due to unneeded calculations.
- Hash table was slowing down execution, switched to list for storing population physical properties
- Only calculate attract or repel forces if a population can be affected by it.

6.10. Log

This chapter lists an exhaustive list of features and issues documented throughout the project.

6.10.1. Features

Feature	**Labels**	**Status**	**Info**
Set background color	Gfx, world	Done	Added color chooser widget for background oclor
Creation of unlimited ball populations.	Ball properties	Done	Model now supports unlimited amount of ball populations instead of just 2.
Interface to set ball population properties in NetLogo for testing	Interface, Ball properties	Done	Added numerous widgets that enable setting properties for all population for any physical property.
Add/Remove walls with brush	Brush, Wall	Done	Brush will continuously add walls while it is held down. If patches have balls in them, then balls will be moved aside to nearest vacant patch and then wall will be added to patch if balls were successfully moved to another patch.
Add/Remove balls with brush	Brush, balls	Done	Create balls when brush is clicked. Add slider to determine

			how many balls get added, and a button + monitor widget combo to select and monitor which population balls are being added to. Removing ants by painting with brush on balls, or clicking “Remove all” button to remove all balls belonging to population currently selected in ui.
Monitor amount of balls in population	Interface, balls	Done	Added monitor for amount of balls for population currently selected in ui.
Add line wall shape	Brush, wall	Done	Create a wall line with brush. When mouse is first clicked it defines the starting point of the line, when mouse is released the end point.
Add rectangle wall shape	Brush, wall	Done	Create a rectangle shaped wall with brush. When mouse is first clicked it sets the center of the rectangle, dragging brush around while pressed down changes dimensions of rectangle. Releasing mouse creates the rectangle shaped wall. Sides are thickness of 1 patch.
Add square wall shape	Brush, wall	Done	Create a square shaped wall with brush. When mouse is first clicked it sets the center of the square, dragging brush around while pressed down changes dimensions of square. Releasing mouse creates the square shaped wall. Sides are thickness of 1 patch.
Add circle wall shape	Brush, wall	Done	Create a circle shaped wall with brush. When mouse is first clicked it sets the center of the circle, dragging brush around while pressed down changes dimensions of circle. Releasing mouse creates the circle shaped wall.
Brush size	Brush	Done	Brush affects more patches the bigger size it is. Add chooser for brush size.
Brush shape square/circle	Brush	Done	Draw with brush on patches in a square or circular shape. Add chooser for brush shape.
Brush icon to indicate which brush type is currently chosen	Brush	Done	An icon will appear in view that will show which brush type is currently chosen, i.e. “wall” is the brush type, then a wall icon will appear near mouse pointer. Need to take into account brush outline
Show coordinates of brush	Brush, gfx	Done	Brush icon shows coordinates of brush
Brush icon to indicate if brush is set to draw or erase	Brush	Done	An icon will appear in view that will show if brush is set to draw or erase.
Add/remove halo with brush	Brush, halo, ball	Done	Ball will have a halo set around it to help distinguish it from others. Adding and removing a halo is done with the brush.
Add/remove trace with brush	Brush, trace, ball	Done	Balls will trace the path they move in. Setting a ball to start/stop tracing is done with the brush
Radio buttons for brush	Brush, interface	Done	Instead of clicking on forever button “Activate brush” to activate the brush, it is now possible to simply click on the buttons that set what brush is drawing and they remain pressed down, until another brush button is pressed.
Set wall color	Wall, interface, gfx	Done	Color chooser widget added to choose wall color
Create block-based interface	blocks	Done	Currently supports 2 populations
Group widgets in interface by relevance	Interface	Done	Improved interface by grouping related widgets together

Block to run population a limited number of times	Blocks	Done	Added block that sets the population to run only a set number of turns.
Add block to set gravity strength	Blocks, gravity	Done	Added block to set gravity for each population.
Add block to set electric field strength	Blocks, electric field	Done	Added block to set electric field strength for each population.
Logging	Log	Done	Model can now log a users interaction with model. Added switch widget to enable/disable logging.
Add/Remove electric field with brush	Brush, electric field	Done	Create an electric field by pressing down and dragging brush to set the direction of field. Remove electric field bet setting brush to erase and clicking on a field.
Move block based interface to right of model so that blocks and model can be seen at once	Blocks, interface	Done	Blocks moved to the right of model.
Monitor number of wall collisions per population	Balls	Done	Added monitor widget that shows amount of collisions for 2 populations.
Ball collision detection	Balls, efficiency	Done	Algorithm to efficiently detect ball collision. Utilizes grid spatial hashing built-in with NetLogo to check for collisions only with balls in patches around ball.
Barnes-hut simulation	Efficiency	Done	Efficient algorithm for the N-body problem. Calculates forces on balls in O(Log n) time.
Make model unusable before it is initialized	Interface	Done	The “starter” global variable is no longer needed to keep track when model was first used.
Plot amount of wall collisions per second for each population	Balls, walls, plot	Open	Create a plot with a graph for each population that averages the amount of wall collisions for the last number of ticks.
Cross hair cursor for brush when drawing a wall shape	Brush, gfx	Done	Brush has cross hair cursor when drawing a wall shape.
Move walls	Brush	Open	Ability to move walls. moving walls can be done after they are created in a certain shape, or by selecting "move" mode. this means there is a need to record every batch of walls that was created, so that selecting and moving it is possible. When moving cursor should show if something is movable and highlight it, like highlighting the wall. Releasing button erases walls from old location and created them in new location.
Move entities in world	Brush	Open	Similar to moving walls, make moving balls and counters possible as well.
Delete batch of walls	Brush	Open	Ability to delete a batch of walls (such as a shape) created together. When mouse hovers over a wall patch, it highlights all other walls belonging to same batch, and if clicked then deletes all of them. Another possibility is if the “moving wall” feature is added, then maybe a bin icon is presented at the top, and if the wall is dragged to the bin location then it gets erased.
Add/Remove counters with brush	Brush, counters	Open	Add counters to patches brush is drawing on. If brush was drawing on patches that contain an existing counter when first pressed down, then it will add that counter to the patches it is drawing on. Otherwise it will create a new counter. Different counters have different colors.

Add counter information	Counters, gfx	Open	Counter information consists of counter number and amount of balls that have passed through. The information is centered on the center most patch of the counter.
Make wall appear above counter	Walls, counters	Open	This can be done by checking if wall is created in patch with counter, then set that counter to be hidden. When wall is removed then check if counter exists there, if so then set it to shown.
Flash wall collisions	Balls, walls	Done	Walls flash where balls collide with them. A switch widget added to toggle between enabling/disabling flash.
Save world	World	Done	Saves current world state to a file
Load world	World	Done	Loads current world state to a file

6.10.2. Issues

Issue	**Label**	**Status**	**Info**
Tracing ball leaves a line across world when it return cyclically around world	Trace	Done	Refactored code to set pen up before ball moves around world, change its location, then set pen down again to continue tracing. NetLogo does support world wrapping, but this model does not enable it, so it has to program balls moving cyclically, which can cause these problems. It would be better to use NetLogo's built-in world wrapping than programming it by hand, but this would require changing the code in a few places.
Changing parameters in NetTango is very slow, about 40 seconds.	Blocks	Done	Refactored code and reduced 11% of code, this surprisingly cut down compilation time after setting parameter to 3 seconds. Update: NetTango has released newer version, changing parameters is very fast now with no wait at all. Recompiling is also very fast.
Rearranging blocks does not change population properties while model is running	Blocks	Done	Refactored code to check for block changes each turn.
Logging does not work with NetLogo web	Log	Done	It is not possible to use current code to append logs to a file in NetLogo web. A new log file must be created and downloaded from browser every log change (doesn't happen too often, so it wont slow things down or take too much disk space). The log history now has to be kept inside a global variable in the model since it can not be accessed from a file from NetLogo web.
Can not create ball population that has not been initialized in NetTango blocks	Ball, blocks	Done	Ball populations have been set to have default properties and can now be created even if their properties have not been defined in the block based interface.
Compilation issue causes variables to get initialized when recompiling after block change.	Blocks	Done	When recompiling after blocks have been rearranged, variables seem to get initialized. This bug has been reported to the NetTango devop team. Happens when using

			extensions in NetLogo. Can not currently use table extensions, need to implement hash map in netlogo. Update: Bug has been fixed and pushed to staging server, waiting for release start using extensions again.
NetLogo web does not support "in-radius-nowrap" primitive	Code	Done	Refactored to not use that primitive for now
All balls change movement when new balls are added	Balls, Code	Done	Fixed bug.
Current implementation of drawing wall shapes is non-responsive in NetLogo web	Brush, walls	Done	Refactoring code and reusing big parts of brush code used in ants model.
Redundant use of “display” primitive in code	Code	Done	Refactored to not use this primitive when not needed, it slows down execution a lot.
Creating a procedure block then changing it to command block does not change it to command block, unable to connect to procedure block	Blocks	Done	NetTango devops team seem to have resolved this issue.
Setting a wall in a patch should be done by setting a variable in patch and set by setting patch to certain color	Code, walls	Done	Patches have a boolean variable named “is-wall” to determine if its a wall, instead of being the color blue. Code refactored to check for this variable instead of checking if patch is colored blue.
Halo not removed when balll is removed	Balls, halo	Done	Fixed.
Balls get stuck when wrapping around world to a patch that has a wall.	Balls, walls	Open	Balls get stuck inside wall when wrapping around world to a patch that has a wall.
Coloring balls by speed is very slow	Balls, efficiency	Open	Need to refactor implementation, runs very slow due to how color is calculated.
Erasing walls and counters needs to remove flashes as well	Walls, counters	Open	When a wall or counter that has been flashed is removed, the flash still stays. Need to remove flashes as well when it is removed.
Error creating electric field but not moving mouse	Brush, electric field	Open	Creating a field by clicking without moving brush results in an error.