Creating “Contagion” with AgentCubes Online

A virus is infecting your world. As people move randomly and come face to face with a sick person, they too become sick. Learn how illnesses can spread through this simulation.

Created by: Catharine Brand and Susan Miller, University of Colorado

This tutorial has been designed as part of the Scalable Simulations Design project. It draws inspiration and content from simulations designed by Fred Gluck and an AgentSheets tutorial written by Susan Miller.

This material is based upon work supported by the National Science Foundation under Grant No. DRL-1312129 and CNS-1138526. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

3D - Contagion

Lesson Objective:

- To create a simulation of an epidemic
- To explain and use the Computational Thinking Patterns listed below

Prerequisite Skills:

- Students are presumed to know the following skills. Return to the Frogger Lesson Plans for detailed instructions on these skills.
- Create agents
- Basic agent behavior including:
- Random movement
- Ending the simulation

Length of Activity:

- Five to Eight 30-45 minute lessons, although some students may advance more quickly

Computational Thinking Patterns:

- Simulation Properties
- Perceive/Act
- Scripting

Challenge projects:

- Diffusion
- Hill-Climbing

Activity Description:

- Part 1: Create a simple model of the spread of an illness
- Concept Introduction/Review: Using Simulation Properties
- Part 2: Adding a Monitor to Create a More Accurate Simulation
- Part 3: Make person agents sick for a variable length of time
- Student Challenge 1: Adding fatalities
- Student Challenge 2: Modeling immunity
- Student Challenge 3: Vaccination
- Student Challenge 4: When should I stay home?

SCALABLE GAME DESIGN

3D-Contagion

Table of Contents

Vocabulary/Definitions

Agent Attribute........................a named value-holder or local variable belonging to an agent, which can be assigned different values. Scent and sick_clock are agent attributes.

Algorithm.................................a set of instructions designed to perform a specific task.

Broadcast ....A way for agents to communicate with other agents that are not adjacent to them - agents broadcast (or send out) a method name, telling other agents to check the rules in that method. The method referenced in the broadcast must be defined in the receiving agents' set of methods.

Diffusion...................................the process in which an attribute like scent that belongs to a group of agents such as ground agents changes its value, being larger near its source and smaller farther way from its source

Immunity..................................to keep yourself from being affected by a disease

Increment.................................to increase by one

Hill Climbing ....a specific form of searching/seeking algorithm, by which the seeking/searching agent checks the values of an agent attribute belonging to another agent.

Method.....................................a set of rules to follow when an agent must make a decision.

Randomly.................................to occur in non-systematic ways

Rule Order ....the order in which rules are placed for each agent

Simulation Property................an attribute (value) accessible by all agents.

General Teaching Strategies[1]

Basic Philosophy

- The educational goal of these lessons is to learn and apply Computational Thinking Patterns in the context of a familiar game. Emphasis on these Computational Thinking Patterns is essential for student understanding.
- These lessons are also designed to give students positive experiences with and perceptions of computer science. Research shows that students turn away from high school and college computer science courses if they perceive it as boring, unrelated to what matters to them, and hard. We hope to change that by providing a fun, relevant and accessible computer science experience where they can personalize their experience to make computer science about them.
- Guided discovery is the central tenant of our curriculum. Direct instruction is sometimes used for aspects where students are learning the code for the first time; however, materials have been provided to ensure that students understand the programming concepts, as opposed to simply copying code.
- Whenever possible, students should try to come up with the steps on their own or in small groups, when differentiation and more structure is needed, we have more structured materials available.
- Student materials are available for each portion of the game design. These materials are intended to be used in addition to teacher materials, which provide prompts and discussion points.
- Students may become frustrated with too little teacher support, THIS IS OK! A little frustration and moving at a slower pace is well worth the deeper conceptual understanding that comes with guided discovery.

Guided Discovery Process

- **Model the process** rather than just giving students the answer. As a teacher, focus on explanations and discussions of WHY something works or doesn't work and let the students figure out HOW to make it work.

---

[1] This information is supported by research found in the following documents:

Basawapatna, A. R., Koh, K. H., & Repenning, A. (2010, June). Using scalable game design to teach computer science from middle school to graduate school. In Proceedings of the fifteenth annual conference on Innovation and technology in computer science education (pp. 224-228). ACM.

National Research Council. (2011). *Learning science through computer games and simulations*. (M. Hilton & M. Honey, Eds.). Washington, DC: The National Academies Press.

National Research Council. (2014). *STEM Integration in K-12 Education:: Status, Prospects, and an Agenda for Research*. (M. Honey, G. Pearson, & H. Schweingruber, Eds.). Washington, DC: The National Academies Press.

Repenning, A., & Ioannidou, A. (2008, March). Broadening participation through scalable game design. In *ACM SIGCSE Bulletin* (Vol. 40, No. 1, pp. 305-309). ACM.

- Building the game on your own, before trying it with your class will enable you to see which steps may challenge or confuse your students.

- Have students work through problems independently or in small groups. Ask directing questions or give helpful suggestions, but **provide only minimal assistance** and only when needed to overcome obstacles.

- **Group work is your friend**! It is common for computer programmers to talk through problems with one another, and to use code snippets found from other programs and other programmers. Talking through coding problems enables students to think more critically about Computational Thinking Patterns, as well as the steps needed to solve a problem.

- Additionally, seeing how others solved an issue with code helps students realize that problems often have multiple solution strategies, and that some solutions might be more effective than others. Also group work lets them see that they are not alone and that others have similar and different questions, struggles, inspirations and perspectives.

- Recognize that programming is largely a process of **trial and error**, particularly when students are first learning. It is helpful to encourage this mindset with your students.

- "What have you tried so far? Why didn't it work?" is a great way to start any troubleshooting discussion.

Building Blocks

- Each project is designed to build on the prior one. Very little student support is provided where expertise has already been created. Conversely, material that is new has more support.

- Be sure to talk through the building blocks (especially for PacMan in the area of diffusion and hill climbing) as these Computational Thinking Patterns will appear often in future games and simulations.

- Encourage discussion and reflection on these Computational Thinking patterns. Small group or whole class discussion relating Computational Thinking patters to the outside world can be super productive.

- Remember that conceptual understanding takes time, and it may be necessary to review these concepts multiple times, using different examples, so that all students can be successful.

Support Learning

- Research shows that game design is associated with engaged students, and engaged students show higher levels on conceptual understanding. Allowing students to personalize their games aids in this engagement and motivation. Plus, it makes grading and reviewing games more fun for you.

- Coding may be difficult for some students, and all students are likely to be frustrated at times when the code does not produce the expected results. **Praise students** for sticking with the troubleshooting process and encourage them to share what they learned with others.

- Consider students who are ahead to the role of "code ambassadors" to walk around and help their peers with coding questions.

- Be sure to communicate that **the process is more important than the answer**, and that coding of a project often takes time. Do not place pressure on your students to 'hurry up' and resort to giving them the code. The process of figuring it out on his/her own will result in much stronger conceptual understanding.

Differentiated Instruction

Note that there are many vocabulary words in this lesson that may be new for your students. Take time to define those words. Using the words in context often will reinforce their meaning for the students.

- **Students who need a challenge:** Some students with more fluency in programming may finish this very quickly – be prepared for by having challenge activity materials ready in advance.

- **Students who need more assistance:** Other students (especially those with no Frogger experience) may struggle a bit more. There are two options for differentiated instruction. Consider the needs of the student and the class as you decide which will work best.

  - Option 1: Pair a struggling student with an experienced student
  - Option 2: Provide student with a tutorial found on the Scalable Game Design Wiki[2]. Note that tutorials do not support independent thinking and should only be used when absolutely needed.
  - Vocabulary for ELL Students: : Generate, Absorb, Collision, Agent, Grotto, Depiction, Condition, Transport
  - Time management issues: While students can be more engaged when they design their own agents, some students can spend too much time on this design or find it frustrating.

Note that there are two versions for some of the student pages. The STANDARD version is designed for most students. It presumes basic knowledge of AgentCubes Online. The ALTERNATIVE version is designed for those students who may need more support. It provides explicit directions for the first part of the project.

[2] http://sgd.cs.colorado.edu/wiki/Scalable_Game_Design_wiki

Teacher Instructions:

Part 1 – Basic Contagion

Introduce this project to the students by asking them how diseases are spread.

- Ask students to explain how colds are spread, and determine the 'rules' of an epidemic
  - Healthy people and sick people walk around the world.
  - When they come into contact with one another, the healthy people sometimes get sick.
  - After some time and perhaps some medical care, sick people usually recover but depending on the disease, some may die.
- You may choose to show this video by Emery University that uses images from Contagion, the movie to describe how diseases spread.
  - https://www.youtube.com/watch?v=OH9_hZ9uomk

If you show these videos, ask your students…   • How does the disease spread?   • What happens as more and more people get sick?   • Which parts can we model? Which parts may be difficult to model?   • Are there variables? For example, does it matter how contagious it is? Does it matter how long the illness usually lasts? Does it matter if you go see a doctor?

Explain that these are all design features that must be considered when planning a simulation. Now, tell them that they will be designing their own contagion simulation.

As a class, briefly create a description of the Contagion simulation similar to the one below.

- identify simulation objects, called agents, by locating nouns in the simulation description
  - When the nouns are the same, with different descriptive adjectives, we may want to use different depictions.
- identify agent interaction by locating verbs in the simulation description

It might look something like this:

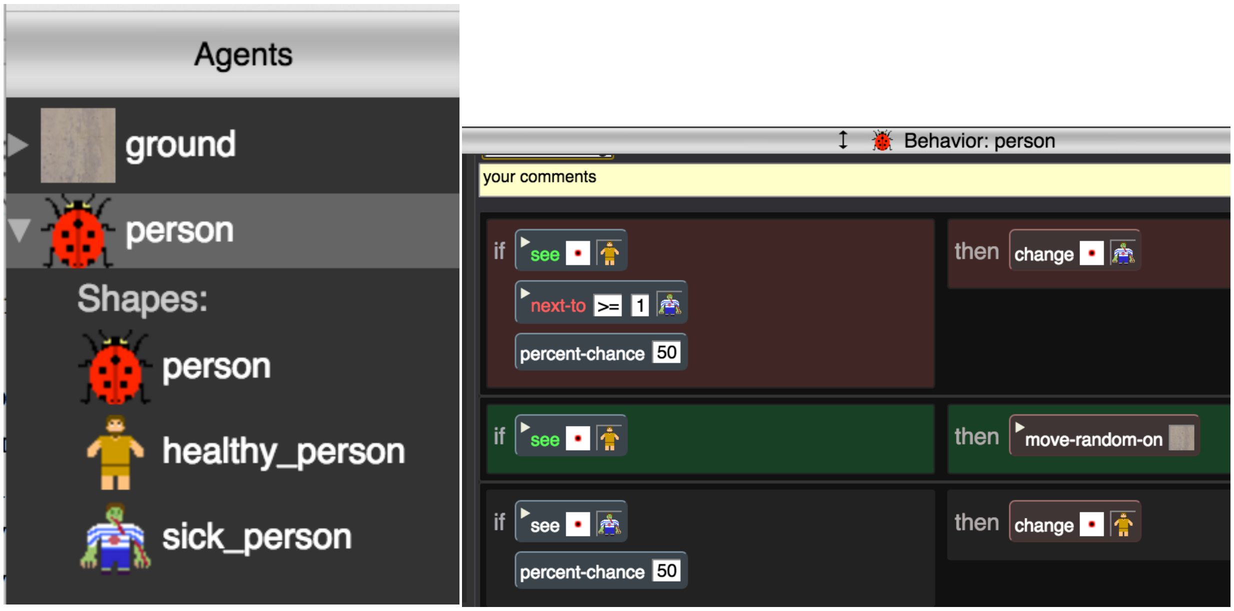
> *Healthy* **people** and *sick* **people** walk around the **city**. When they come into contact with one another, *healthy* **people** get sick with a 50 percent probability. *Sick* **people** recover with a 50 percent probability.
>
> Agents:
> - people (two depictions: healthy, sick)
> - city (tiles that cover the world so the agents can walk on something)
>
> Actions:
> - Next to, with a 50% probability, change to sick people
> - With a 50% probability get better

Give the students 10 minutes or so to work with a partner to discuss what steps will be needed to create this simulation. If students have already completed Frogger and Journey or PacMan, they should have a pretty good idea of how to do this. Add new students to a pair of experienced students who will be willing to talk through their thinking. Stress the need to think through the programming process. At this point, there should not be hands on the keyboards (even though some will want to jump right in to programming!).

This lesson is intended to be taught in a guided discovery manner. Be sure to give students time to work on their own and figure things out using the program. Encourage students to work together and talk through problems with one another. Emphasize that troubleshooting is a normal and important part of programming.

Solicit and discuss possible ideas, without providing any evaluative feedback (do not tell students if their ideas are good/bad, right/wrong). Once there has been some class discussion, provide students with the appropriate handout.

Code for the Person Agent



- **STUDENT HANDOUT 1A provides the details needed for experienced students to get started on their own. You may choose not to print the second page with the code if the students can figure it out on their own. (Page 3 of the Standard Student Packet)**
- **STUDENT HANDOUT 1B provides <u>step-by-step instructions</u> for creating a world and agents for students who are new to AgentCubes or have missed a class. (Page 3 of the Alternative Student Packet)**

3D-Contagion

Student Handout 1A: Part 1 – Very Basic Contagion

You will be modeling or simulating the spread of a disease. You will begin with a simple model and then make it more realistic.

In this first model, we have a very basic world. Healthy people and sick people walk around. If you are healthy and next to a sick person, you might get sick. If you are sick, you might recover and become healthy again.

Talk with a partner, and answer these questions:

1. What agents are needed? Think about these agents. Do you need different agents for healthy/sick people, or one agent (people) with two shapes (healthy/sick)?
2. What actions will they have?

Once you have a plan, begin to make the simulation.

Once your simulation is programmed correctly, use the single step control above the world to run it one step at a time and see what happens.

- What happens when you increase the percent chance of getting sick to 100%? What would this mean in real life?
- What happens when you decrease the percent chance of recovering to 0%? What would this mean in real life?
- What probabilities for getting sick and recovering seemed most realistic?
- Does the simulation end?
- How can you tell how many agents get sick? Die? Recover? Are there ways to use programming to help you do this?
- Do sick people move? Should they?
- In what ways is this a realistic simulation? In what ways does this simulation **not** match what really happens?

Student Handout 1B: Very Basic Contagion

(No ACO Experience)

You will be modeling or simulating the spread of a disease. You will begin with a simple model and then make it more realistic.

In this first model, we have a very basic world. Healthy people and sick people walk around. If you are healthy and next to a sick person, you might get sick. If you are sick, you might recover and become healthy again.

Talk with a partner, and answer these questions:

3. What agents are needed? Think about these agents. Do you need different agents for healthy/sick people, or one agent (people) with two shapes (healthy/sick)?
4. What actions will they have?

Once you have a plan, begin to make the simulation. If you need more help, look at the following pages for directions.

Once your simulation is programmed correctly, use the single step control above the world to run it one step at a time and see what happens.

- What happens when you increase the percent chance of getting sick to 100%? What would this mean in real life?
- What happens when you decrease the percent chance of recovering to 0%? What would this mean in real life?
- What probabilities for getting sick and recovering seemed most realistic?
- Does the simulation end?
- How can you tell how many agents get sick? Die? Recover? Are there ways to use programming to help you do this?
- Do sick people move? Should they?
- In what ways is this a realistic simulation? In what ways does this simulation **not** match what really happens?

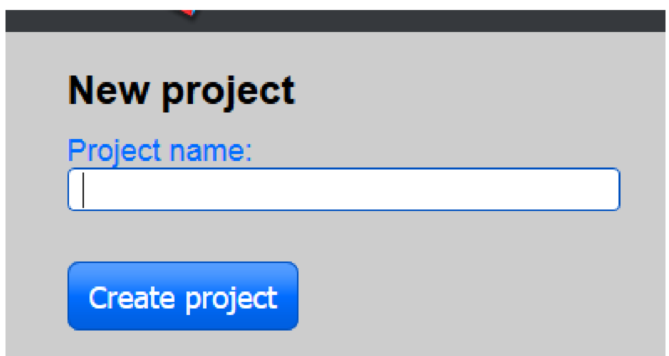
Part 1: Create Agents

<table>
<tr>
<td>Step 1:
Create Simulation</td>
<td>


Go to https://www.agentcubesonline.com/
If you have an account, click on the Login link. If not, click on the Sign up link.

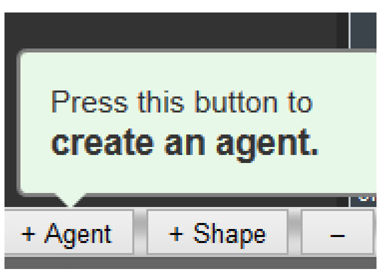
After you login, click on the blue “New project” link below your login name.
</td>
</tr>
<tr>
<td>Step 2:

Name the Simulation

Name it Contagion and click Create project.</td>
<td>


</td>
</tr>
<tr>
<td>Step 3:

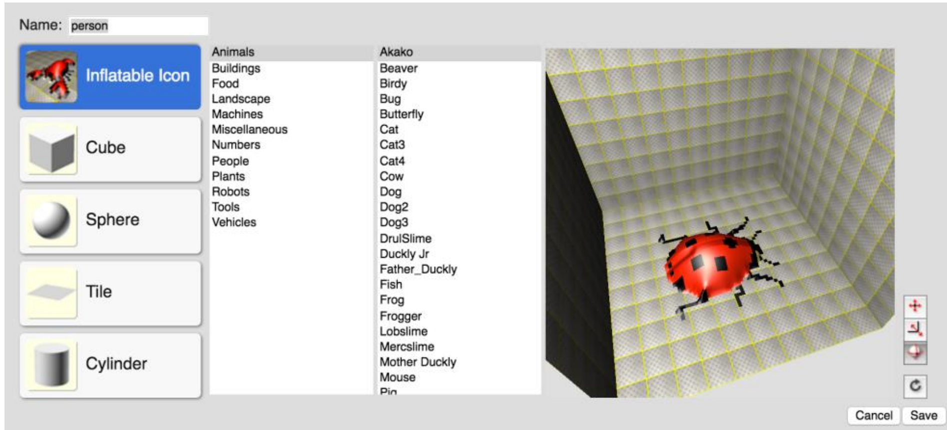
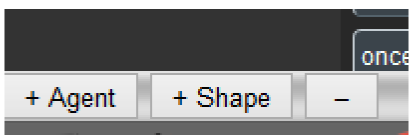
Create Agent

Click on New Agent at the bottom left of the AgentCubes Online window.</td>
<td>


</td>
</tr>
</table>

3D-Contagion (Continued)

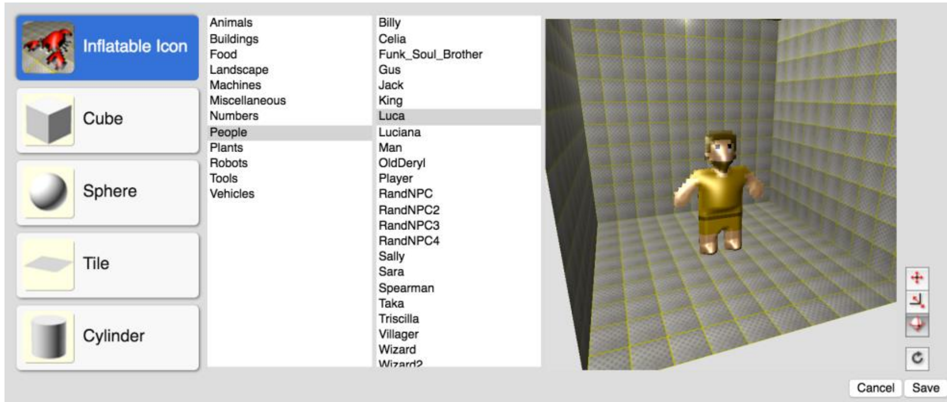
<table>
<tr><td>Step 4:
Name agent “Person”

Simply choose any inflatable icon and click OK.</td><td>
</td></tr>
<tr><td>Step 5:
Select “Person” from the agent list on the left, and click “+Shape” button at bottom left of window.</td><td>
</td></tr>
<tr><td>Step 6:
Now we will make the shape for a healthy person so name the shape healthy_person.</td><td>
</td></tr>
<tr><td>Step 7:
Using the drop down menu under “Person" on the left, click on small box next to healthy_person</td><td>
</td></tr>
</table>

3D-Contagion (Continued)

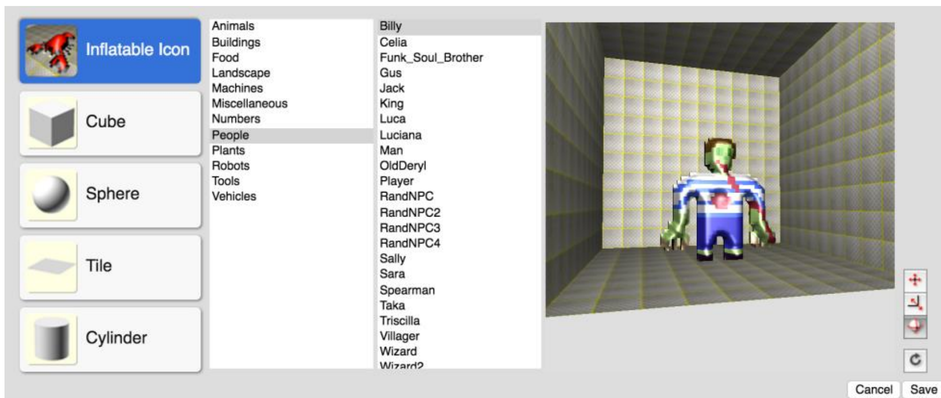
Step 8:
Pick a person from the inflatable icon people list to be your healthy person.

Luca or Sally are good choices.

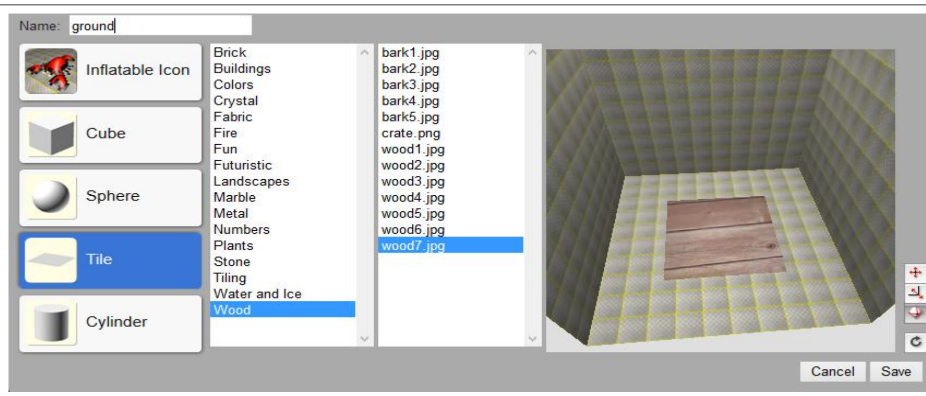


Step 9:
Make a sick_person shape for your person.

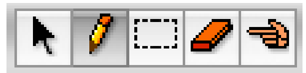
Use the zombie Billy or color your healthy person green or red so you can clearly identify the sick person.

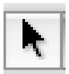


Step 10:
Click +Agent to create a new agent to be the ground. Select any Tile and name it “City”.

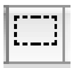


Part 2: Create the World

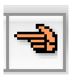
Use these tools to build your world: 

 **The Select Tool is used to move agents or check an agent attribute.**

 **The Pencil Tool draws a single agent on the world.**

 **The Dotted Rectangle Tool puts an agent in every grid square inside the rectangle made by selecting tool, clicking on the world and dragging.**

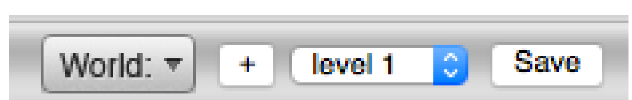
 **The Eraser Tool erases agents from the world.**

 **The Hand Tool is used to call a method created by the programmer.**

The world is the simulation space where the agents perform their actions.

Step 1: Make your world. **Use the Dotted Rectangle tool to put a layer of city tiles on the world.** **Use the dotted rectangle tool to draw a group of healthy people on the world.** **Use the eraser to erase a healthy person in the middle of the group and then use the pencil to draw a single sick person in the middle of the group.**	

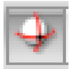
Click on the <u>Save</u> button when you have your world set up the way you like.

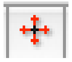


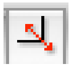
Once you save your world, you can use the reset button  **to return your world to the saved state.**

3D-Contagion (Continued)

Now that you have created a world, use the 3D tools to move your world around so that you have a 3D view of it rather than a bird's eye view looking down from above.

 **The ball tool tilts and rotates your world until you see a 3D view like this picture.**

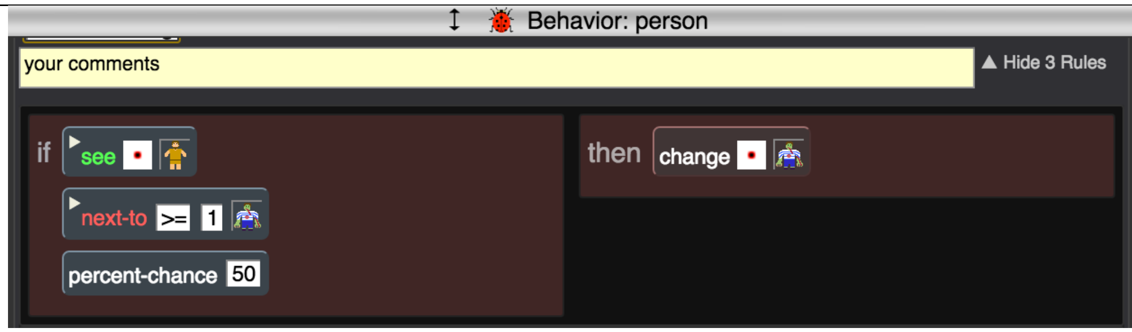
 **The pan tool moves your world around in the window.**

 **The zoom tool zooms in and out on your world.**

Save Your World again!

Part 3: Program the Agents

Click on an agent from list of agents on the left side of the AgentCubes window, and you will see the methods and rules that control its behavior in the window below the world.

Step 1: **Create Rule 1**	 **Rule 1: If you SEE that you are healthy and you next to <u>at least 1</u> sick person with a 50 percent chance, then get sick.**

Why should you use next-to >= 1 person?

- **Should a healthy person who is next to 2 sick people get sick?**
- **Should a healthy person who is next to 1 sick person get sick?**

Since both these situations should be true, use next-to >= 1 sick person, which means <u>next to AT LEAST ONE (and maybe more)</u> sick persons.

Step 2: **Create Rule 2** **Program the words in bold type!**	 **Rule 2:** You did not get sick so **If you see that you are healthy, then move random on the city tiles.**
Step 3: **Create Rule 3**	 **Rule 3: If you see that you are sick, with a 50 percent chance, then get well.**

Teacher Instructions: Using simulation properties

Students will find that they want to quickly be able to change the probabilities that the healthy people become sick and recover. To do this, students can create a simulation property (also known as a global variable).

For example, rather than include a number percent in the code, use **@Get_sick** in the percent condition 

to represent the probability of a healthy person becoming sick when next to a sick person**.**

Remember that the @ symbol must be used in front of the simulation property name in order to access the its value.

Connecting simulation properties to variables:

> *Simulation properties are simply variables. In math, we often use x and y as our variables. Sometimes, if we are talking about girls and boys, we might use g and b as our variables to help us keep track of which item each variable represents. In programming, rather than using a letter, you can use a word, like Get_Sick. This makes it very easy to track each variable.*

Students may choose to change the names of their simulation properties – this will give you an opportunity to discuss good naming practices.

If students have not worked with simulation properties before (in Journey or PacMan, for example) this is best taught as a whole class session once the issue is raised. If the students have worked with simulation properties in the past, challenge them to use them now. Give them time to figure it out on their own before you offer help. If students have not used simulation properties, Student Handout: Using Simulation Properties (Page 4 of Standard Student Packet, Page 10 of the alternative packet) will be helpful.

3D-Contagion

Student Handout: Using Simulation Properties

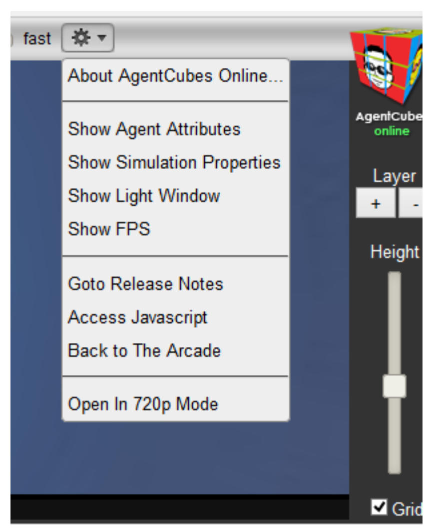
What if we want to look at different percentages to see how the model changed? For example, some diseases like colds, which are spread by coughing and sneezing, pass easily from person to person so there is a big chance that an exposed person will get sick. Other diseases are not transmitted as easily so there is a smaller percent chance that an exposed person will get sick. That's difficult to model when the percentage is built into the code. We can fix that by using <u>simulation properties</u>. Simulation properties (or global variables) are <u>variables</u> that are accessed by all the agents in the simulation. We will use these variables in the percent-chance conditions because it is easy to change the value and different diseases can be modeled.

For example, rather than include a number percent in the code, use **@Get_sick** in the percent condition 

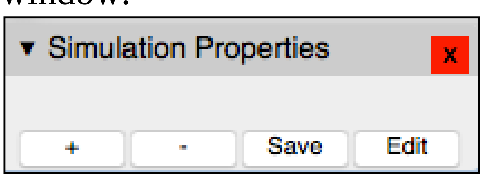
to represent the probability of a healthy person becoming sick when next to a sick person**. Remember that the @ symbol must be used in front of the simulation property name in order to access the its value.**

Follow these steps to create and save the Get_sick Simulation Property:
Step 1: Create a global variable.

Open the simulation properties. Click on the Gear button  in the top bar of the AgentCubes Online window, then choose Show Simulation Properties.



Step 2:
Click on the + button in the Simulation Properties window.

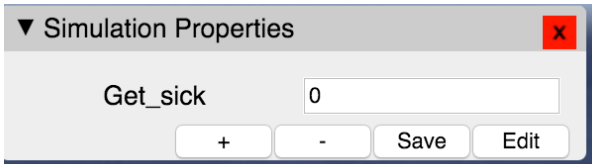


Step 3:
Name the property Get_sick.

3D-Contagion (Continued)

Step 4:

Select the Get_sick property and click on the Edit button.

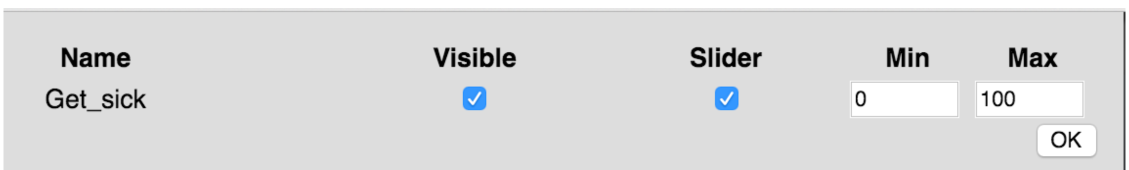


Simulation Properties:

Simulation properties are simply variables. In math, we often use x and y as our variables. Sometimes, if we are talking about girls and boys, we might use g and b as our variables to help us keep track of which item each variable represents. In programming, rather than using a letter, you can use a word, like Get_Sick. This makes it very easy to track each variable.

Step 5:

Click the slider button and set the max value to 100.



Step 6:

Now the value of Get_sick can be changed to any number from 0 to 100 by using the slider.

Save your Simulation Property now
or it will not be there next time you open the project!

Follow these steps to create the other simulation property that is useful in the contagion simulation:

Recover = probability that a sick person will recover.

The values of this simulation property must be between 0 and 100.

Part 2a: Teacher Instructions – Adding a Monitor Agent

When the students ran the first version of the simulation with the single step button, person agents who were not next to the original sick person may appear to become sick on the first step. This happened because at each step of the simulation, each person agent's check for a sick neighbor was immediately followed by the same agent's actions. Sometimes a person agent had gotten sick and moved near another agent which then got sick on the same turn.

Ideally, only agents next to a sick agent should get sick on each step of the simulation. The way to make this happen is to have the monitor tell the agents what to do. First each person agent looks around and decides what it should do. When all the person agents have had an opportunity to look and decide what to do, each person agent in turn then does its actions. The monitor is **scripting** the agents' behavior, telling the agents first to **perceive**, then to **act**.

In this lesson, students will add a monitor agent and script the person agents' behavior so that the simulation works in a more realistic and correct way.

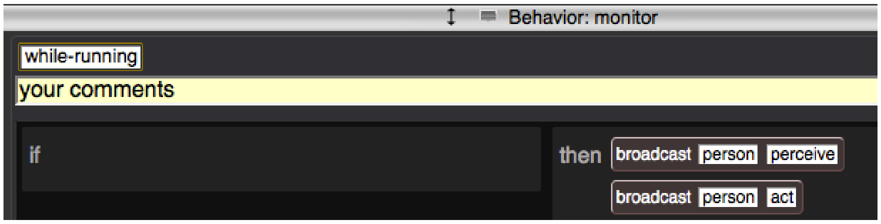
Our monitor agent will have a role similar to the World Health Organization (WHO) created by the United Nations to provide leadership on global health issues. This organization eradicated smallpox, and is working on eliminating several communicable diseases including HIV, malaria and tuberculosis. This organization was also very involved in trying to mitigate the recent Ebola outbreaks. As part of its responsibilities, the WHO organization surveys populations every year to determine who has what illness.

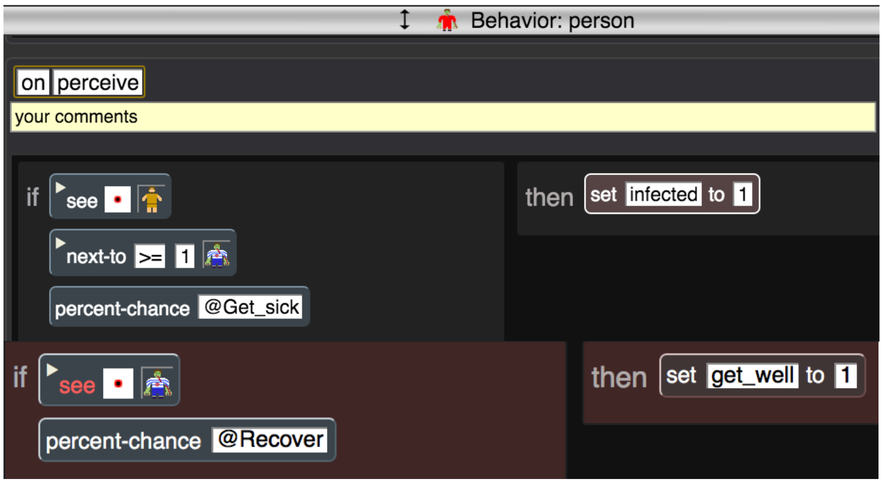
Eventually, our monitor agent will have 4 responsibilities:

1. It will script the behavior of the person agents by telling all the person agents to check their perceive methods and then telling them all to check their act methods.
2. It will count all the healthy and sick person agents in the world at each step of the simulation.
3. It will plot each of these numbers so that a graph is created that shows the change in each population over time.
4. It will end the simulation when the world has no sick people in it or no healthy people.

Provide students with Student Handout 2a (page 6 of the standard packet, page 12 of the alternative packet). Each step is completed in its own lesson. Some code is provided to the students. Other code they must figure out. All code is provided in the teacher handout.

Use of Monitor to conduct Perceive/Act steps





Student Handout 2a: Adding a Monitor Agent

Step through your simulation again. Pay close attention to who gets sick on each step. You might be seeing people get sick who aren't next to a sick person. This is because the computer does two things you might not know…first, it runs faster than your eye can track. This means that an agent might move without you seeing it. Second, when it checks each agent on the screen, it starts at the top left corner, works through the row to the right, and then moves down to the next row. This doesn't do a good job of modeling what happens in the real world because even though someone may be next to a sick person, they might move before they check to see if they get sick. So we are going to change the programming so that all people agents check to see if they are next to a sick person at the same time!

First each person agent looks around and decides what it should do. When all the person agents have had an opportunity to look and decide what to do, each person agent in turn then does its actions. The monitor agent **scripts** the agents' behavior, telling them first to **PERCEIVE**, then to **ACT**.

We will also need to create a 'switch,' which is a local variable that keeps track of whether each agent should 'switch' from healthy to sick, or sick to healthy. We will call these INFECTED, and GET_WELL.

Agent: Monitor (Be sure to place the monitor on the worksheet!)

Rules:

- The Monitor tells your person agents to PERCEIVE who is around them, and determine if they need to <u>switch</u>. Then, the Monitor tells them to ACT, based on that knowledge.
    1. PERCEIVE
        - If you are healthy and next to at least one sick person, identify yourself as infected (INFECTED = 1), based on some probability
            - Consider what percent of the time that should be.
        - If you are sick, identify yourself as getting better (GET_WELL = 1), based on some probability.
            - Consider what percent of the time that should be.
    2. ACT
        - If you are healthy and now infected, change yourself to a sick person, reset your INFECTED switch back to zero.

- If you are sick, and no longer infected, change yourself to a healthy person, reset your GET_WELL switch back to zero.
- Otherwise, move randomly around the world.

PERCEIVE and ACT are methods. Using a METHOD is like having a special set of procedures that get done only when asked. A fire drill in your school is a real-world example of a method. No matter what else is going on, when you hear the fire alarm, you stop what you are doing, and follow the directions of your teacher (usually to find the nearest exit and leave the building!). This method (fire alarm) is 'called' by the sound of the alarm.

In AgentCubes Online, we 'call' a method by 'broadcasting' to the agents. We then put the rules in a method box with the appropriate agent, which is associated with the method name.

Let's get started:

Some code is provided. Some is not. Work with a partner if you get stuck!

Step 1: Create the monitor agent and place it on the worksheet.

Step 2: Create a single rule for the Monitor: Have the monitor tell (broadcast) the person agents to PERCEIVE and then tell (broadcast) the person to ACT

Create the METHODs

Step 3.

In the <u>person</u> agent.

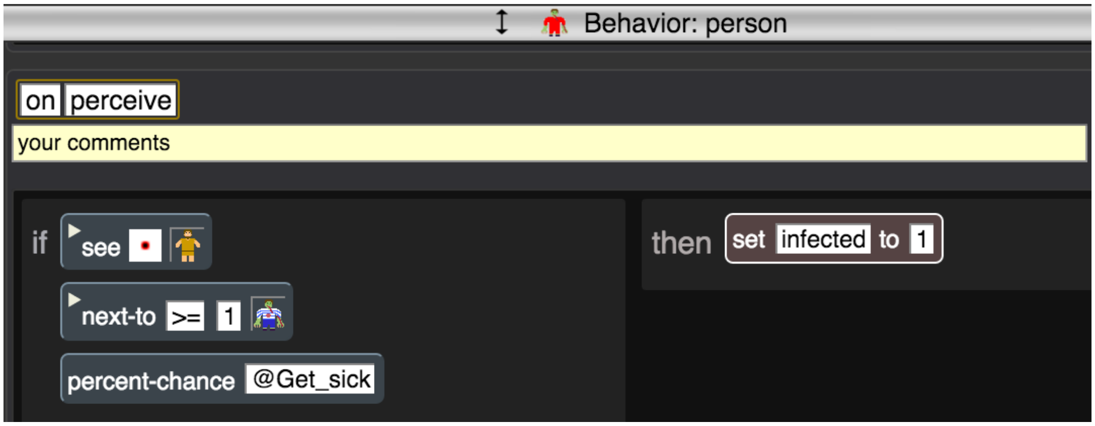
1. Use the +Method button to add 2 new methods to the person agent.
2. Click on the word "unknown" in the upper left corner of each new method.
3. Name one method "perceive" and the second method "act".
4. Click on the header for the person's while running method and then click on the minus button [−] on the bottom of the AgentCubes Online window to delete the while running method.

3D-Contagion (Continued)

PERCEIVE METHOD

Step 4.

Add a rule to the perceive method: if they are healthy and next to a sick person, with some probability, then set the switch for labeling them as INFECTED.



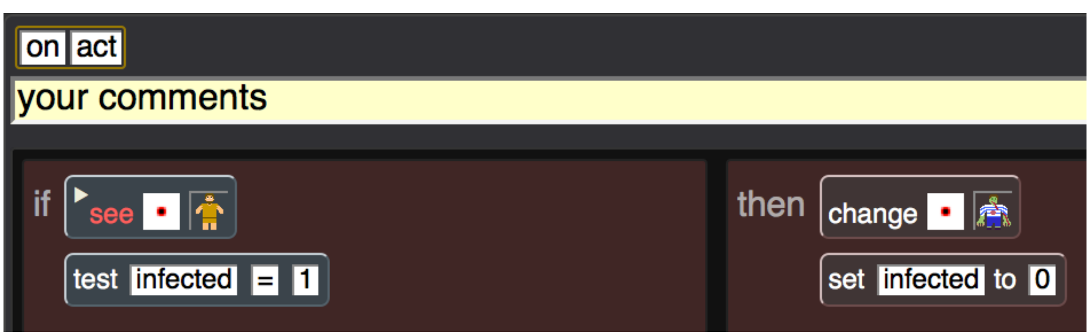
Set the agent attribute "infected" to 1 if the person will get sick. Infected will stay equal to 0 if the person does not get sick this turn.

Step 5.

Add the rule to the perceive method that makes the person get well. Set the local variable "get_well" to 1 if the person will recover. Remember that the actual change to a healthy person will happen in the act method.

ACT METHOD

Step 6. ***Add a rule*** to the act method that makes a healthy person who is infected get sick. If a person is infected, change the person to look sick and reset the infected agent attribute back to 0.



3D-Contagion (Continued)

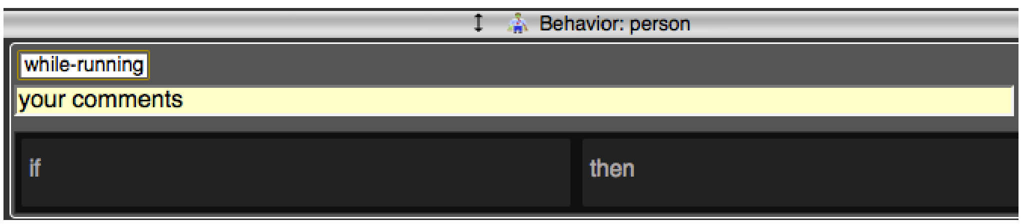
Step 7.

Add a rule If you are a healthy person, and not infected, just make move randomly on the world.

Step 8.

Add a rule If you are a sick person, and should get well, change to a healthy person and reset the get_well attribute to zero.

HINT: Double check to ensure that the person agent's 'while running' method contains no rules (it will look like it has one empty rule) so that the person agent will only do the perceive and act methods when the controller broadcasts the perceive and act messages!



In our version, we chose to have only the healthy people walk around. Is that realistic? What changes might make this more realistic?

Part 2b: Teacher Instructions – Counting/Graphing the people

The World Health Organization tracks illnesses throughout the world. Ask students why tracking illnesses might be important. Recent examples including Ebola and Zika virus may come up as possible topics. In the United States, the Center for Disease Control (CDC) tracks illnesses. While some tracking is to prevent spreading of the disease, there are other reasons to track illnesses:

> ***Nearly 26 million Americans have asthma. The Tracking Network has contributed to better management of asthma.*** *The Tracking Network made possible a report on deaths in the U.S. due to asthma, challenges in treating this condition, and strategies to educate patients and others about asthma. Health departments and policy makers used the report to make recommendations about asthma management.* *http://www.cdc.gov/about/facts/cdcfastfacts/surveillance.html*

Provide the students with Student Handout 2b (page 10 of the standard packet, page 16 of the alternative packet). Class discussions during this lesson might focus on the following topics:

The use of equations and how they differ from mathematical equations

For example, the equation x = x+ 1 is false mathematically. There is no value of x that makes that statement true. Yet, in programming, equations are not considered in the same way. Instead, they are algorithmic. The equation time = time + 1, means that in each cycle, the value of 'time' will increase by 1.

What can be determined based on the graphs

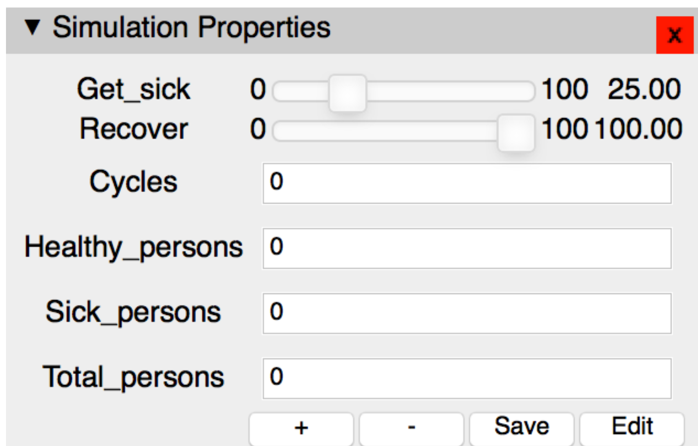
It would be interesting to have students compare their graphs with different values for the percent of people becoming ill versus the percent who are recovering. Could students guess at the values that others have chosen, just by looking at the graphs?

Part 2b: Student Instructions – Counting/Graphing All the Sick and Healthy Persons in the World

The World Health Organization tracks illnesses throughout the world. Graphs help us to visualize whether the disease is spreading, and whether people are recovering. In order to make a graph with the number of sick and healthy people, the monitor agent must keep track of the numbers of sick and healthy persons over time.

COUNT THE PEOPLE
Step. 1

Make new simulation properties that track the number of healthy persons, sick persons and total persons. I've chosen to name mine like this:



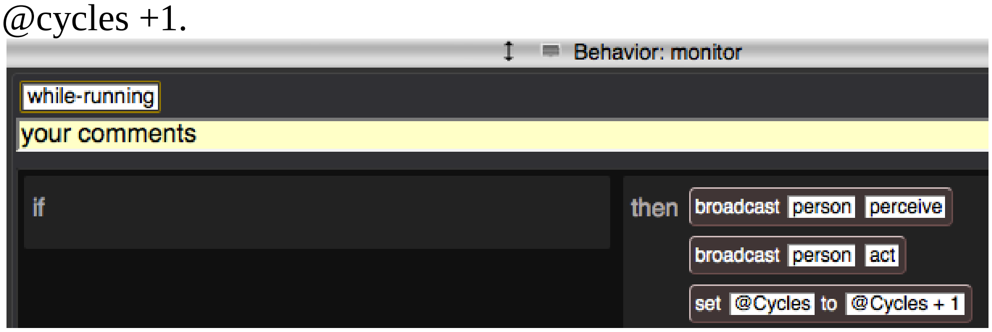
This time I didn't set them up as sliders. Why did I make this choice? Talk to a partner about why sliders are not helpful in this situation.

Make sure to click on the Save button!

Step 2.

Add the **set** actions that update the simulation properties to the rule in the monitor agent's while running method.

I want the time to increase by one each time we count. Therefore, I set the @cycles to @cycles +1.

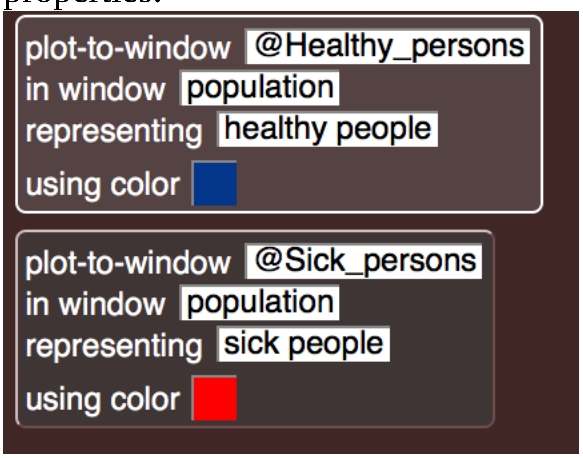


Add some additional statements to this same rule:

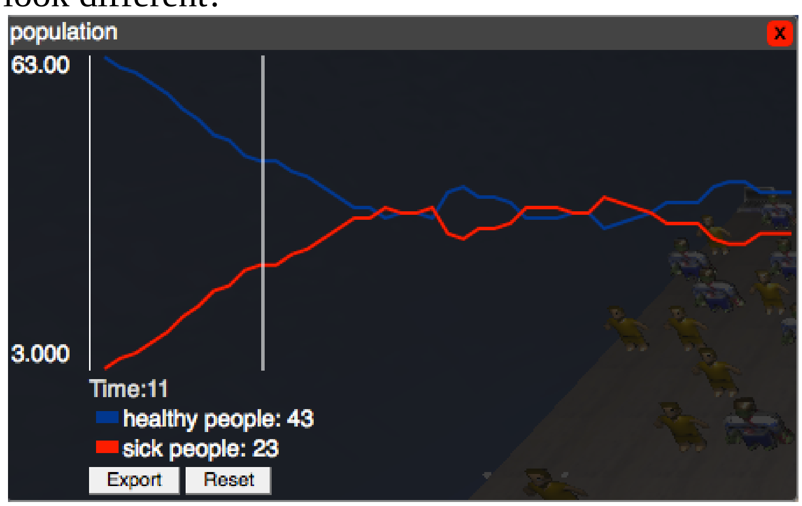
1. Set @Healthy_persons to agents_with_shape("healthy_person")
2. Set @Sick_persons to agents_with_shape("sick_person")
3. Set @Total_persons to agents_of_type("person")

GRAPH THE PEOPLE

Add these two actions to the main rule of the monitor agent to graph the simulation properties.



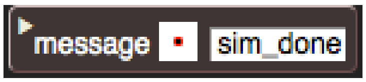
Run your simulation and look at the graph. Does it look like mine? What might cause it to look different?

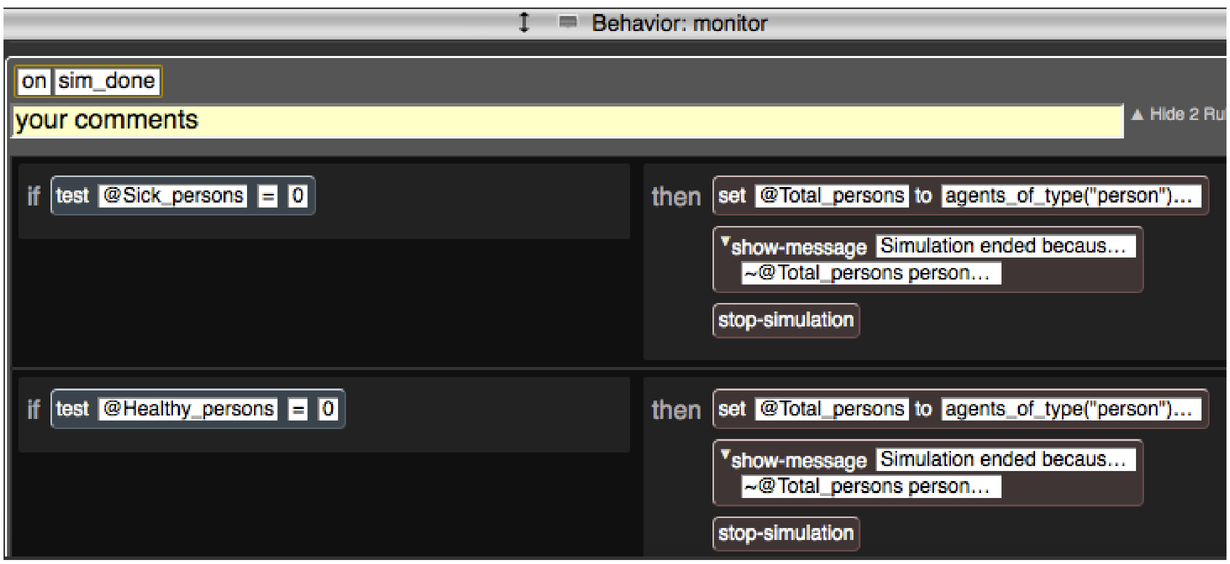


3D-Contagion

Part 2c: Teacher Instructions – Ending the Simulation

Students will determine appropriate end points for their simulation. Discuss as a class why different end points might be chosen.





Provide students with Student Handout 2C (page 12 of the standard packet, page 18 of the alternative packet). Once students have made the changes, have all students run their simulation.

- What happens if the Get_sick and Recover simulation properties are both set to 50%?
- What happens if Recover is set to 0 and Get_sick is set to a large value? What if Get_Sick is a small value?
- Whose simulation runs the longest? What conditions make that happen? What does that tell us about disease control?
- In what ways is this simulation realistic? In what ways is this simulation different from what really happens?
- What changes would you like to make that would model the real world more closely?

Part 2c: Student Instructions – Ending the Simulation

Unlike a game that can be won, simulations run until an endpoint decided by you. You might choose the end the simulation after 100 cycles. That would help you answer the question, ‘What happens after 100 days of an illness.’ Or you might want to run the simulation until either the disease has been eradicated (meaning that only healthy people remain) or the disease has wiped out civilization (only sick people remain).

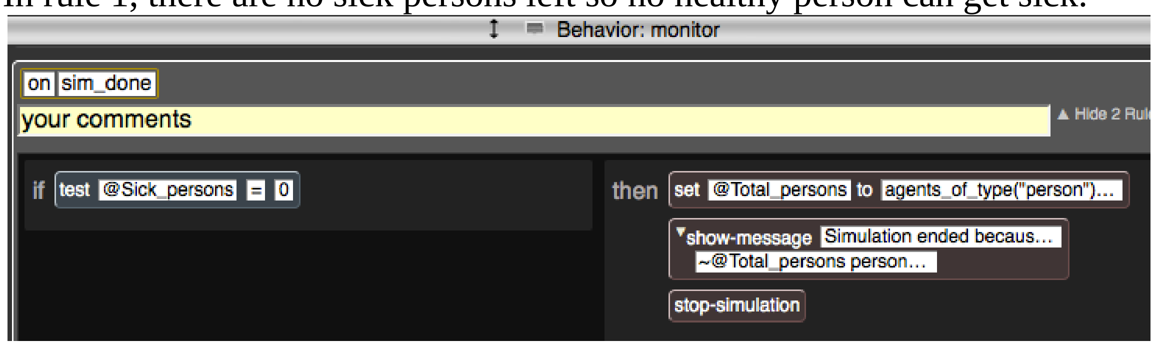
We will show you how to end the simulation once everyone is healthy. You can decide if you want to include more possibilities.

Step 1: Make the monitor check whether the simulation is finished.

Add a final action to the rule in the monitor’s while running method.
This action tells the monitor to send a message to itself to check its ‘sim_done’ method.

Step 2: Program the sim_done method.

For each rule, show a message to the observer that explains what happened to end the simulation.
In rule 1, there are no sick persons left so no healthy person can get sick.



If you would like to include how many cycles the simulation ran, write ~@Cycles cycles and AgentCubes Online will produce a message that says 100 cycles, replace “~@Cycles” with the value of the simulation property Cycles. Make sure that there is a space immediately after “~@Cycles” so that AgentCubes Online does not get confused by an extra character.

Run the game and vary the percentages of time people get sick and recover. What numbers are most realistic for a common cold? For Ebola? Why?

Teacher Instructions: Part 3 – Getting Better: Adding in the length of the illness

In this next lesson, students will learn to improve their simulation by making it more realistic. Take a moment to talk with the students about getting sick. Consider these prompts:

- What happens when you get sick?
- Do you stay sick forever?
- How do you get better?
- Do you have to go to the doctor to get better?
- Why/Why not?
- How long are you sick with a cold?

You may choose to watch this video with your students, describing the common cold: http://www.youtube.com/watch?v=UWgiyQV3nYc

Near the end of the video, the narrator states that the common cold lasts 7-10 days if you have 'average health.' This brings up some more topics for discussion…

- What is 'average health'?
- What makes colds last longer?
- What might shorten the length of a cold?

Students will now build in a 'timer' for the person agent in their simulation, to model the idea that people get better after about 10 days. Each person agent will have a sick_clock that counts up. When the sick_clock equals the value of the simulation property Minimum_sick_time, a rule in the Perceive method will test whether the person recovers **during that simulation clock tick or cycle**, using the probability stored in the Recover simulation property.

For example, if 10 person agents have sick_clocks = Minimum_sick_time and Recover = 50 (meaning 50% of the people get better), on the first tick of the simulation clock, on average, 5 will recover. So those 5 would be sick for just the minimum time of 7 days.

Of the 5 remaining sick people who were still sick, on the second tick of the simulation clock, 3 might recover. Those 3 would be sick for 8 days.

Of the 2 remaining sick people, on the next click of the simulation clock, 1 might recover. So that one person would be sick for 9 days.

On the next tick of the simulation clock, the last sick person has a 50 percent chance of recovery. So if that person recovers, it would have been sick for 10 days.

If the last person does not recover until the next tick of the simulation clock, the last person would be sick for 11 days.

This method lets us have a more realistic simulation in which the majority of sick people recover in the minimum time but a few take longer. And we can not predict exactly which people will be sick longer, just like in real life.

Once students understand the concepts, pass out the Student Handout (found on page 13 of the standard packet, page 19 of the alternative packet). All code is provided to the students for this lesson.

Discussion Prompts:

Run your revised simulation. What happens in your simulation? Is it what you expected?

- What happens if the cold lasts longer than 7 days?
- What happens if some people already have a weakened immune system and stay sick longer? What would change in your simulation properties?
- What happens if there are more sick people in the simulation?
- What happens if there are more healthy people in the simulation?
- In what ways does this simulation reflect the real world? In what ways does the simulation not reflect what really happens?

Student Handout 3 – Getting Better: Adding in the length of the illness

As you discussed with your class, you don't stay sick forever – even though you might feel miserable with a cold, you will get better, generally in a week or so. We are going to change the simulation to reflect that people do get better, and that recovery takes a number of days typical of whatever illness we are modeling. For example, colds usually last about 7 to 10 days.

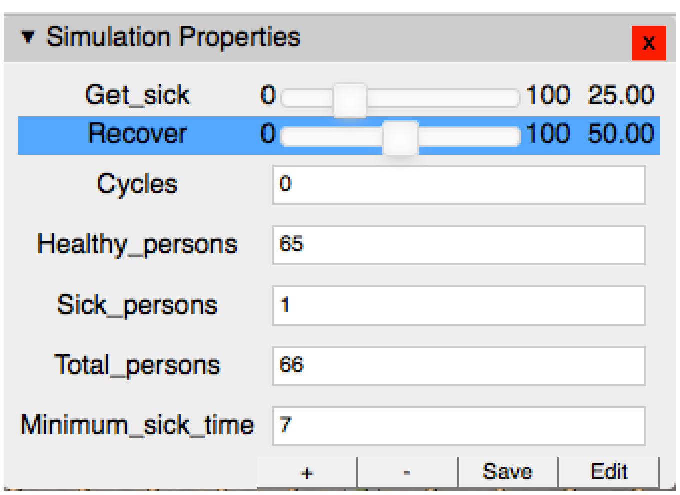
We will use a simulation property, Minimum_sick_days to designate the length of time that our person agents will be sick. We will use a variable called as SICK_CLOCK to count up how long someone has been sick. Confused? Don't be – we'll take this step by step.

Code for Modeling Length of Sickness

Step 1. Create two new simulation properties: Minimum_sick_time and Recover.

To model the common cold, set Minimum_sick_time to 7.

Edit the Recover property to be a slider with values from 0 to 100 because it stands for the probability that the person agent will recover **on any given cycle**. In this case, set Recover to 50 percent. The result will be that some persons recover after 7 days, some after 8, 9 or 10 days and a few take even longer to get well.

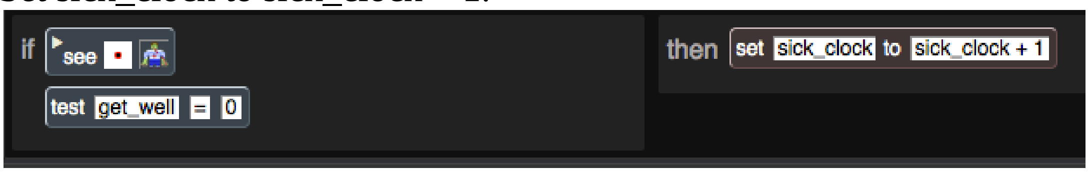


Remember to save your simulation properties!

Step 2. The person agent's sick_clock is automatically set to 0 by AgentCubes Online.

Step 3. If the sick person is still sick, their sick_clock counts up. <u>Add this rule to the Act method.</u>

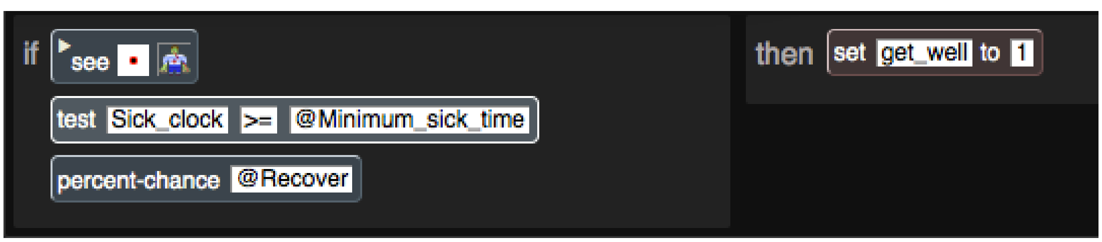
Set sick_clock to sick_clock + 1.



Step 4. <u>Edit the rule in the Perceive method</u> so that if person agent's sick_clock is greater than or equal to the <u>Minimum_sick_time</u>, the person has some chance of recovering. The simulation property <u>Recover</u> controls how likely the person is to recover on any given cycle.

Use a test condition to check whether sick_clock >= @Minimum_sick_time.

Note the "@" before the simulation property names. We need it to get the value of each simulation property.



For example, if 10 person agents have sick_clocks = Minimum_sick_time and **<u>Recover = 50</u>**, on the first tick of the simulation clock, on average, 5 will recover. So those 5 would be sick for just the minimum time of 7 days.

Of the 5 remaining sick people who could recover, on the second tick of the simulation clock, 3 might recover. Those 3 would be sick for 8 days.

Of the 2 remaining sick people, on the next click of the simulation clock, 1 might recover. So that one person would be sick for 9 days.

On the next tick of the simulation clock, the last sick person has a 50 percent chance of recovery. So if that person recovers, it would have been sick for 10 days.

If the last person does not recover until the next tick of the simulation clock, the last person would be sick for 11 days.

This method lets us have a more realistic simulation in which the majority of sick people recover in the minimum time but a few take longer. And we can not predict exactly which people will be sick longer, just like in real life.

Run your revised simulation. What happens in your simulation? Is it what you expected?

- In what ways does this simulation reflect the real world? In what ways does the simulation not reflect what really happens?

3D-Contagion

Student Handout: Challenge 1.0

Deaths

(Page: Standard: 16 Alternative: 22)

Before your start this challenge: You must have a complete basic Contagion simulation that enables people to get better after a certain amount of time.

Death

Some people die when they are sick too long.

Design Challenge:

When serious diseases like Ebola are modeled, not everyone gets better. Some people will die. Change your simulations to have some small percentage of people die.

First, change the PERCEIVE method by adding a rule that makes person agents die. When the sick_clock reaches the Minimum_sick_time… I have a chance to get better, I may stay sick, or I may have a chance to die. So, create one rule that checks to see if I recover with a Recover percent chance and a second rule that checks whether I die with a percent chance from a new simulation property named "Fatality". Don't forget the switches - The action for the die rule in the PERCEIVE method sets an agent attribute "die" equal to 1.

Then add a rule in the ACT method that checks for agents with a "die" attribute equal to 1. Add some actions to help the viewer see that people have died. Perhaps they will change shape? Make sure to include a short wait so that the change in shape is visible. Maybe there will be a message or a sound?

Be sure to create and graph a <u>simulation property</u> that tracks the number of dead people. (Something to consider: once the dead people disappear off the worksheet, they are no longer countable – how will you keep track of each death?)

3D-Contagion

Teacher Handout: Challenge 2.0

Modeling Immunity

Code to Add Immunity to Viruses to the Contagion Simulation

Step 1. **Person agents who are not immune to the disease have a chance of getting sick.**	 **Add a test condition to see whether this person agent is immune to the illness.**
Step 2. **Person agents become immune to the illness when they recover.**	 **Add an action that sets a recovered person agent's immune attribute to 1 so that the person agent cannot get sick again.**

Student Handout: Challenge 2.0 Modeling Immunity

(Page: Standard: 17 Alternative: 23)

Design Challenge:

In the current version of the Contagion simulation, person agents get sick over and over. In reality, people do not get the exact same virus more than once. Humans do get many illnesses with similar symptoms like colds but the viruses that cause them are different in small ways.

Immunity

People develop immunity over time…

Instead, people develop an immunity to a virus. Immunity means that a person's immune system can identify and destroy a virus that has infected that person at some earlier time.

How could you represent immunity to an illness in this simulation of contagion which takes place over a relative short time?

You could use an agent attribute to represent immunity.

- If "immune = 1" then the person agent cannot get sick but if "immune = 0" then the person agent has a chance of becoming ill.

Think about when the person agents should not have immunity and when they should be immune to a sickness.

- When should the person agent's attribute "immune" be set? Before the person gets sick? When the person recovers?
- What is the effect of being immune? Can a person agent who is immune become sick?
- When an agent is created, should it be immune to the disease in the simulation?

Additional Design Challenge

Humans do not develop a complete immunity to infections caused by microorganisms like the single-celled protozoans which cause malaria. In the case of malaria, humans who survive the first infection will develop a partial immunity and are less likely to die from a second infection although this partial immunity weakens over time. Malaria is treated with drugs which kill the parasite. Vaccines are in development but not available yet.

How could an illness with repeated reinfections be modeled?

Student Handout: Challenge 3.0 Vaccination

(Page: Standard: 18 Alternative: 24)

Design Challenge: Show the effect of Immunity from Vaccinations

Vaccinations

People develop immunity through vaccinations

Another way to give people immunity is to vaccinate them before they get sick.

How would you add vaccination to the simulation? Here are some ideas…

- Add a nurse or a doctor who uses a hill climbing search to find healthy person agents. Healthy person agents must diffuse their scent through the City tiles. When the nurse is next to a healthy person, then that person could become immune to the illness being modeled.
- Add a hospital and make healthy people search for it and get vaccinated when they reach it.

What happens if most people are immunized? Does the sickness spread?

"Herd immunity" means that the majority of vaccinated people become immune and even if a few people get sick, the illness cannot spread easily. If most people are immune to an illness, those who are too young or too ill to be vaccinated and those whose vaccination failed to make them immune will be protected too. You can model herd immunity by making vaccinations work most but not all of the time. Do this by adding a percent-chance condition to the vaccination rule.

How well does a vaccination need to work to provide herd immunity and stop the spread of an illness?

Student Handout: Challenge 4.0
When should I stay home?

(Page: Standard: 19 Alternative: 25)

Design Challenge: Show the effect of going out when sick

Sometimes, even when people are sick, they still go to work and to school. Many say they do that because they don't want to have to make up all that missed work. How does that affect others when sick people can move around?

Change the simulation so that some percentage of sick people walk randomly through the world. Create graphs for different values of sick people who roam. Does it make a difference? Is it okay if a small number of people do this?

What about the level of contagion? Does it make a difference if the disease is very contagious compared to ones that are not very contagious?

Stay Home?

Should we stay home when we are sick?

Additional Design Challenge

What if we sent all the sick people to the hospital? How would that affect the model?

To do this, create a hospital and have sick people use a hill climbing search to find the hospital. The hospital agent must diffuse its scent through the City tiles. Once the sick people are healthy, they can leave the hospital.

ISTE Standards[3] specific to the implementation of Contagion (Denoted with (□))

Creativity and Innovation

Students demonstrate creative thinking, construct knowledge, and develop innovative products and processes using technology. Students:

Apply existing knowledge to generate new ideas, products, or processes:

- □ Design and develop simulations
- Design and develop computational science models

Create original works as a means of personal or group expression.

- Design original simulations
- □ Model your local environment, e.g., ecology, economy

Use models and simulations to explore complete systems and issues.

- □ Model scientific phenomena, e.g., predator / prey models
- □ Create visualizations

Identify trends and forecast possibilities.

- □ Build predictive computational science models, e.g., how the pine beetle destroys the Colorado pine forest
- Build live feeds to scientific web pages (e.g., weather information), process and visualize changing information

Communication and Collaboration

Students use digital media and environments to communicate and work collaboratively, including at a distance, to support individual learning and contribute to the learning of others. Students:

Interact, collaborate, and publish with peers, experts, or others employing a variety of digital environments and media:

- □ Students work in teams to build and publish their simulations as web pages containing java applets.

Communicate information and ideas effectively to multiple audiences using a variety of media and formats.

- Effectively combine interactive simulations, text, images in web pages

Develop cultural understanding and global awareness by engaging with learners of other cultures.

- □ Students and teachers from the four culturally diverse regions interact with each other

Contribute to project teams to produce original works or solve problems.

- □ Define project roles and work collaboratively to produce simulations and simulations

Research and Information Fluency

---

[3] ISTE Standards for Students (ISTE Standards•S) are the "standards for evaluating the skills and knowledge students need to learn effectively and live productively in an increasingly global and digital world." http://www.iste.org/standards/standards-for-students

3D-Contagion (Continued)

Students apply digital tools to gather, evaluate, and use information. Students:

Plan strategies to guide inquiry.

Explore web sites and identify interesting connections

Locate, organize, analyze, evaluate, synthesize, and ethically use information from a variety of sources and media.

Find relevant related web-based information, compute derivate information

Evaluate and select information sources and digital tools based on the appropriateness to specific tasks.

Understand validity of information, e.g. Scientific journal information vs. Personal blogs

Process data and report results.

Write programs to access numerical information; define functions to process data and create output based on voice or plotting to represent data.

Critical Thinking, Problem Solving, and Decision Making

Students use critical thinking skills to plan and conduct research, manage projects, solve problems, and make informed decisions using appropriate digital tools and resources. Students:

Identify and define authentic problems and significant questions for investigation.

Define research questions and explore approach of exploration

Plan and manage activities to develop a solution or complete a project.

- Outline sequence of exploratory steps
- Experience complete bottom-up and top-down design processes
- Employ algorithmic thinking for creating programs to solve problems

Collect and analyze data to identify solutions and/or make informed decisions.

- Collect data as time series, e.g., collect group size of predator and prey, export time series to excel, explore various types of graph representations, e.g., x(t), y(t) or scatter y=f(x)

Use multiple processes and diverse perspectives to explore alternative solutions.

- Experience and understand design trade-offs, e.g. Bottom-up vs. Top-down

Digital Citizenship

Students understand human, cultural, and societal issues related to technology and practice legal and ethical behavior. Students:

Advocate and practice safe, legal, and responsible use of information and technology.

- Learn how to use tools to locate resources, e.g., images with google image search, but understand copyright issues

Exhibit a positive attitude toward using technology that supports collaboration, learning, and productivity.

- Stay in the flow, where design challenges match design skills
- Experience success through scaffolded simulation design activities
- Mentor other students

Demonstrate personal responsibility for lifelong learning.

3D-Contagion (Continued)

- Explore options of going beyond expected learning goals

Exhibit leadership for digital citizenship.

- In a collaborative setting become a responsible producer of content for diverse audiences

Technology Operations and Concepts

Students demonstrate a sound understanding of technology concepts, systems, and operations. Students:

Understand and use technology systems.

- Know how to organize files and folders, launch and use applications on various platforms

Select and use applications effectively and productively.

- Know how to orchestrate a set of applications to achieve goals, e.g., make simulation and simulations using Photoshop (art), AgentCubes Online (programming), and Excel (data analysis).

Troubleshoot systems and applications.

- Debug simulations and simulations that are not working

Transfer current knowledge to learning of new technologies.

- Reflect on fundamental skills at conceptual level. Explore different tools to achieve similar objectives.