

Creating “Journey”

You are a traveler on a journey to reach a goal. You travel on the ground amid walls, chased by one or more chasers. The chasers at first move randomly on the ground, and later, begin to chase based on your scent. When you move next to the goal, you win. If you move next to a chaser, you lose.

Created by: Susan Miller, University of Colorado, School of Education

This curricula has been designed as part of the Scalable Games Design project.
It was created using portions of prior work completed by Fred Gluck and Cathy Brand.

Lesson Objective:

- To create a game of the student's own design where a traveler explores an unknown world.

Prerequisite Skills:

- Students are presumed to have the following skills. Return to the Frogger Lesson Plans for detailed explanations on these skills.
 - Create agents
 - Basic agent behavior including:
 - Key control
 - Random movement
 - Ending the game

Computational Thinking Patterns:

- Cursor Control
- Collision
- Diffusion
- Hill Climbing
- Polling

Length of Activity:

- Three to Five 30-45 minute lesson, although some students may advance more quickly

Activity Description:

- Part 1: Create a basic world with a Traveler and Chaser
 - Part 2: Make the Chaser chase the Traveler
 - Part 3: Enhance the game so that the traveler collects more than one goal
- Challenge: Ice Arrows: Create ice arrows which can freeze the Chaser

Table of Contents

Teacher Instructions:	Part 1 – Basic Game
Student Handout 1:	Part I - Basic Game (experienced student)
Student Handout 1A:	Part I - Basic Game (new student)
Student Handout 1B:	Agent Creation Short Cuts
Teacher Instructions:	Part 2 – Making the Chaser Chase the Traveler
Student Handout 2:	Part 2 – Making the Chaser Chase the Traveler
Student Handout 2A:	Troubleshooting Guide for Journey Part II Diffusion and Hill Climbing
Teacher Instructions:	Part 3 – Making the game harder – Polling and Broadcast
Student Handout 3:	Part 3 – Making the game harder – Polling and Broadcast
Student Handout 3A:	Troubleshooting Guide for Journey Part II Polling and Broadcast
Student Handout 4A:	Ice Arrow Challenge Part A (Guided Discovery)
Student Handout 4B:	Ice Arrow Challenge Part A (Direct Instruction)
Student Handout 4C:	Ice Arrow Challenge B (Guided Discovery)
Student Handout 4D:	Ice Arrow Challenge B (Direct Instruction)

Vocabulary/Definitions

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

Amplitude the value associated with an attribute

Attributean assigned feature of an agent (such as scent)

Bracketsmethod of setting information apart using “[“ and “]”

Broadcast controllers broadcast (or send out) a signal

Chaser the agent that chases the traveler

Collisionan event wherein two agents run into each other.

Diffusion.....the process in which an attribute (in this game, scent) moves from areas of strong concentration to weak concentration

Increment.....to increase by one

Hill Climbinga specific form of searching/seeking technique, or algorithm, by which the seeking/searching agent uses information (agent attribute) embedded in the floor.

Local Variable.....a variable held by a specific agent

Methoda set of rules to follow in a specific situation

Parenthesesmethod of setting information apart using “(“ and “)”

Pollingthe process of contacting and communicating with each agent

Propagated.....the spreading of the scent

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

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

Traveler.....the main character who is searching for goals

Teacher Instructions: Part 1 – Basic Game

Teaching Suggestions

Task your students with creating a new game. The basic features of the game are as follows:

You are a traveler on a journey to reach a goal. You travel on the ground amid walls along with one or more chasers. The chasers move randomly on the ground. When you move next to the goal, you win. If you move next to a chaser, you lose.

Give students to talk about the game for a minute.

Consider these prompts:

- How is this game similar to Frogger? Dissimilar to Frogger?
- What skills that students learned from Frogger will they need here?
- What agents will they need?
- What ideas do they have about how to get the chaser to move randomly (and, what does “randomly” mean)?
- What would make the game more challenging? Less challenging?

Once the students have had a few minutes to think through these prompts, provide them with Handout 1 (or display it for the class) so that they may begin work.

You may have some students who have never worked with AgentSheets. Use Handout 1A for those students.

Differentiated Instruction:

This first section is designed to quickly facilitate the students' creation of the basic game.

Students who need a challenge: Some students with more fluency in programming may finish this very quickly – be prepared for them to move on to parts 2 and 3.

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 struggling students with Handout 1A, which provides more directed instruction steps

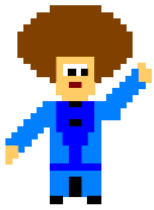
Time management issues: While students can be more engaged when they design their own agents, some students can spend too much time on this design or find it frustrating. Handout 1B provides block images of each agent as portrayed in this lesson.

Vocabulary for ELL Students: Traveler, Chaser, Randomly, Rule Order, Collision

Student Handout 1: Part I - Basic Game

Initial Story: You are a traveler on a journey to reach a goal. You travel on the ground amid walls along with one or more chasers. The chasers move randomly on the ground. When you move next to the goal, you win. If you move next to a chaser, you lose.

Create these Agents:



Traveler



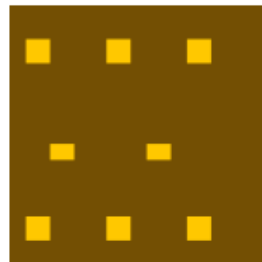
Chaser



Goal

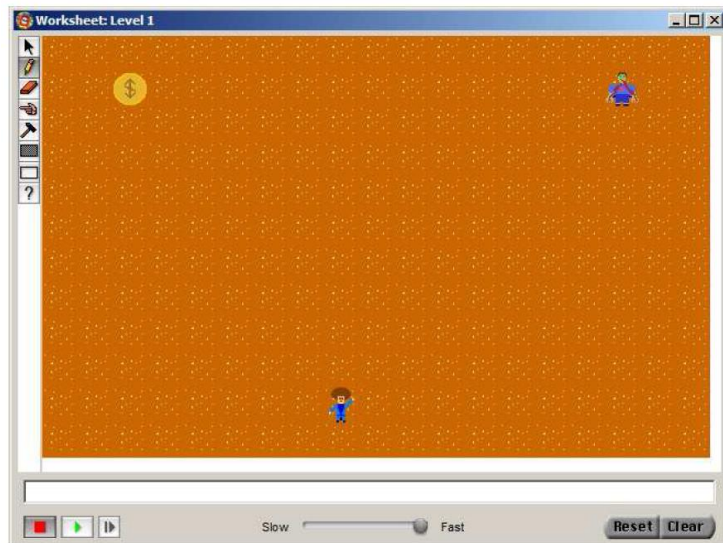


Ground



Wall

Create this initial Worksheet:

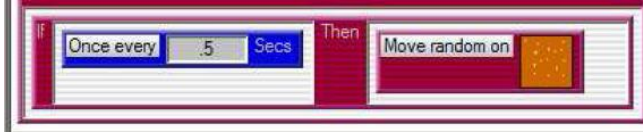


Journey (Continued)

Create the following BEHAVIORS for your agent:

Step 1: Chaser:

Program the chaser to move randomly on the floor.



Step 2: Traveler:

Set up your agent to move with the arrows (cursor control).

Create game ending conditions (collision).

The box below shows how to end the game if your traveler approaches the chaser. Create a similar rule if your traveler approaches the goal.

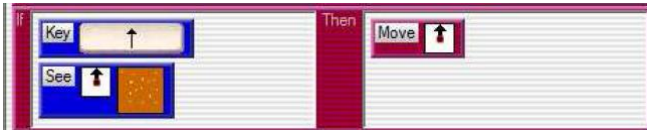


Be SURE to reset the game when it ends.

Step 3: Walls

Add walls to your worksheet. Then, prevent your Traveler from walking through the walls.

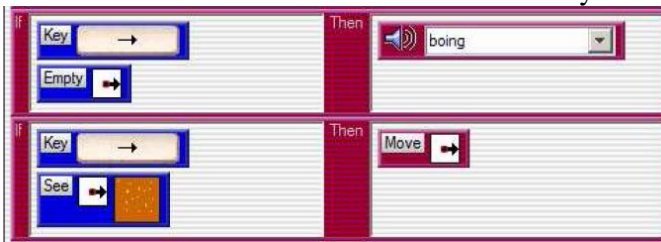
Work with the person next to you to figure out how to prevent the Traveler from walking into a wall. Here is one way to think about it... challenge yourselves to find a different way.



Step 4: Don't allow the traveler to cheat!

Problem: Traveler can cheat by moving off the game ground. Talk with the person next to you about where and when this can happen on your worksheet.

Add rules that make a sound for attempted movement off the ground. Note the importance of rule order for the new rules. Here is an example to prevent the Traveler from moving right off the worksheet. What other direction limit will you need?



Student Handout 1A: Part I - Basic Game

Initial Story: You are a traveler on a journey to reach a goal. You travel on the ground amid walls along with one or more chasers. The chasers move randomly on the ground. When you move next to the goal, you win. If you move next to a chaser, you lose.

Agents:



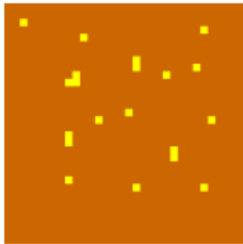
Traveler



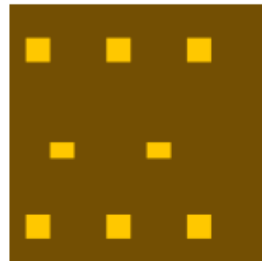
Chaser



Goal



Ground

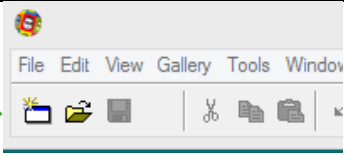
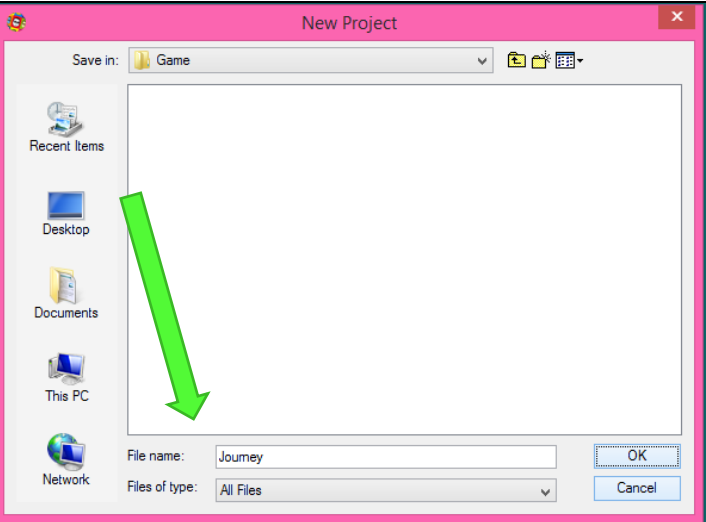
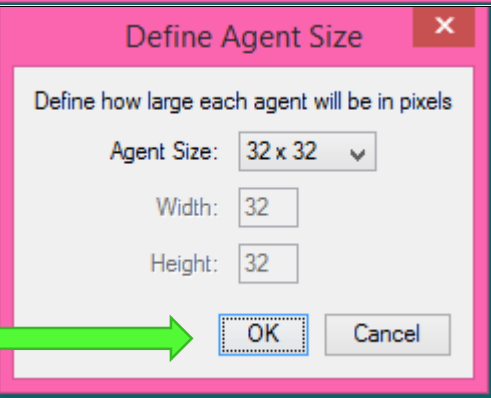
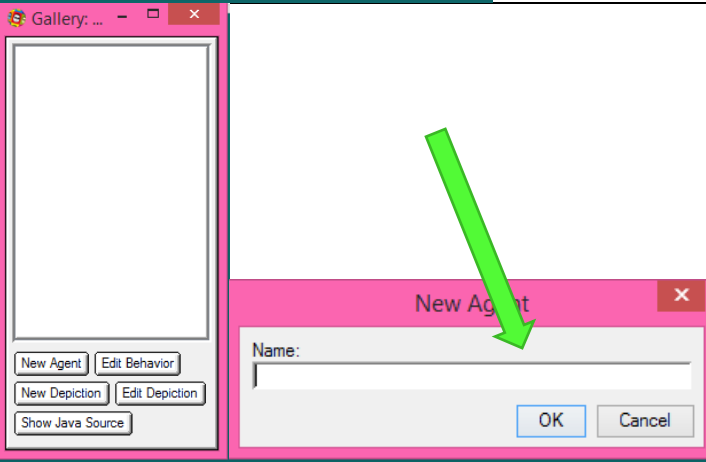


Wall

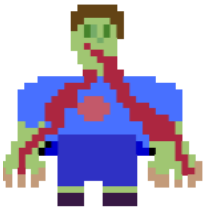


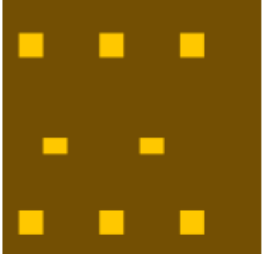
Initial Worksheet:



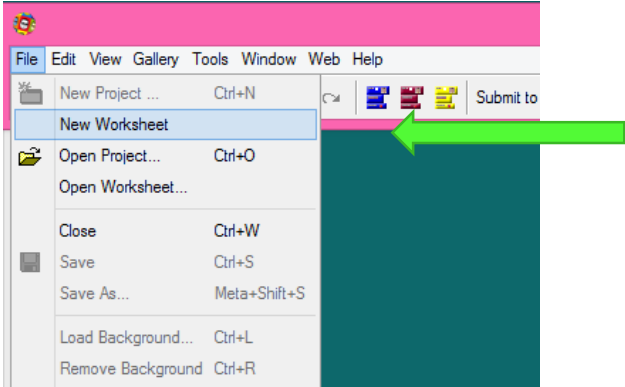
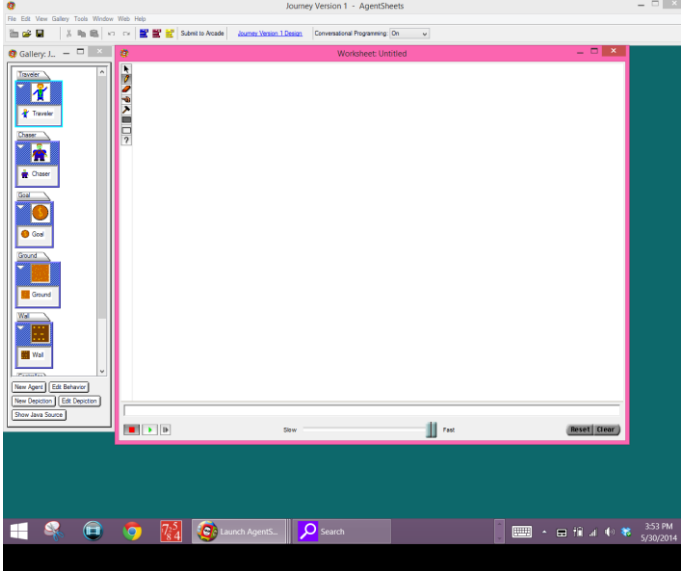
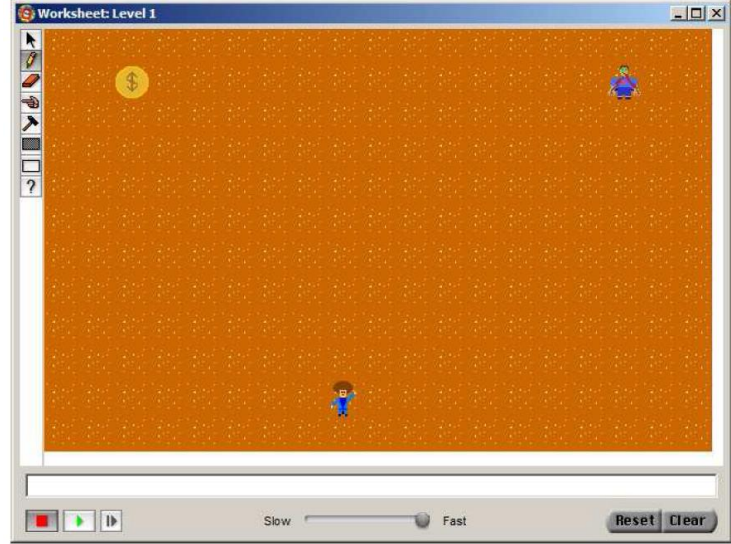
Journey (Continued)

<p>Step 1</p>	<p>Create Game</p> <p>Click on the new game icon (far left)</p>	
<p>Step 2</p>	<p>Name the Game</p> <p>Name it Journey and click OK</p>	
<p>Step 3</p>	<p>Define Agent Size</p> <p>Do not change - Click OK</p>	
<p>Step 4</p>	<p>Create Agent</p> <p>Click on New Agent</p> <p>Name it Traveler</p> <p>Click OK</p>	

Journey (Continued)

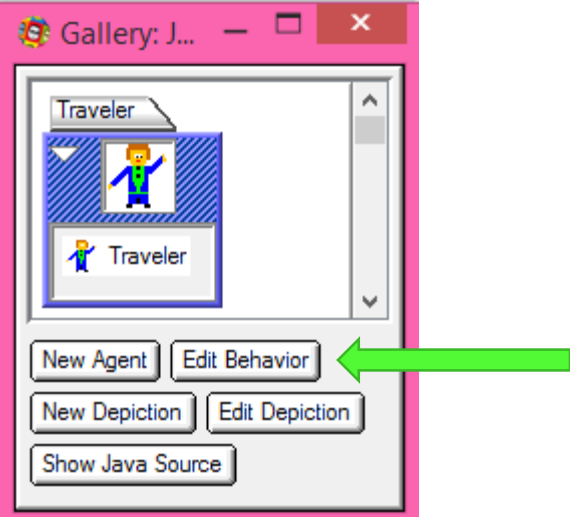
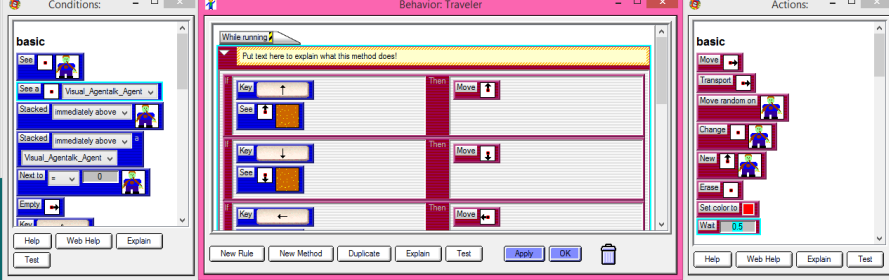
<p>Step 5</p>	<p>Edit Agent</p> <p>Click EDIT DEPICTION</p> <p>Click CLEAR to erase the current image.</p>		<p>Click on <i>Color> Mask Color>> White</i> to make the white background sections see-through</p>
<p>Step 6</p>	<p>Draw Traveler</p> <p>Click DONE</p>		<p>Here is an example of one way to draw him. You can be creative. If you make a mistake, use the eraser or click CLEAR to clear the whole area.</p>
<p>Step 7</p>	<p>Draw remaining agents</p>	<div style="display: flex; justify-content: space-around; align-items: flex-start;"> <div style="text-align: center;">  <p>Chaser</p> </div> <div style="text-align: center;">  <p>Goal</p> </div> <div style="text-align: center;">  <p>Ground</p> </div> </div> <div style="text-align: center; margin-top: 20px;">  <p>Wall</p> </div>	

Journey (Continued)




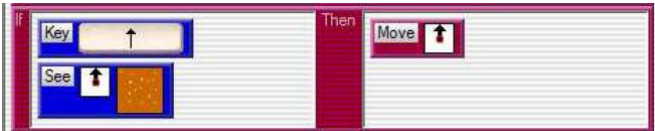
<p>Step 8</p>	<p>Make the workspace</p> <p>Click File>>New Worksheet</p>	
<p>Step 9</p>	<p>Make the worksheet bigger</p> <p>Notice it is big, but not so big that it fills up the whole space.</p>	
<p>Step 10</p>	<p>Use the tools to place items on the worksheet.</p> <p>Pencil: places items one at a time</p> <p>Filled in Rectangle: Places items in an array.</p>	 <p>It is important that you do not draw over an agent with the Ground agent. This means if you place a Chaser on the worksheet, do not draw the Ground over it without erasing the Chaser first.</p>

Journey (Continued)

This is a good time to save the worksheet!

<p>Step 11</p>	<p>Create behaviors for your agents</p> <p>Read the explanation</p> <p>and then</p> <p>Click Edit Behavior</p>	<p>The kind of behaviors that we will give to our Agents are called rules. Rules are made up of an IF-THEN statement. For controlling the Traveler using the cursor keys, one of the rules we need should be that “IF the Up key is hit, THEN the Traveler will move up.” Overall we should have 4 rules, one for each direction (Up, Down, Left, Right).</p> 
<p>Step 12</p>	<p>Create a behavior</p> <p>When you first open this, it will be blank. You are going to drop and drag the conditions (on the left) and the actions (on the right) to create the rules.</p>	 <p>Take a look at this rule...it says,</p> <p>IF I <u>click on the up arrow</u>, THEN <u>my traveler will move UP</u></p> <p>Create the rules to have the traveler move up, right, left and down.</p> <p>NOTE: Each rule has to be separate...use NEW RULE to create each new rule.</p>

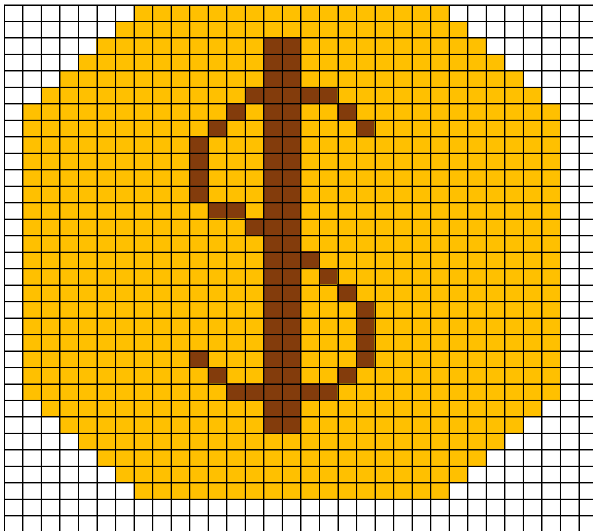
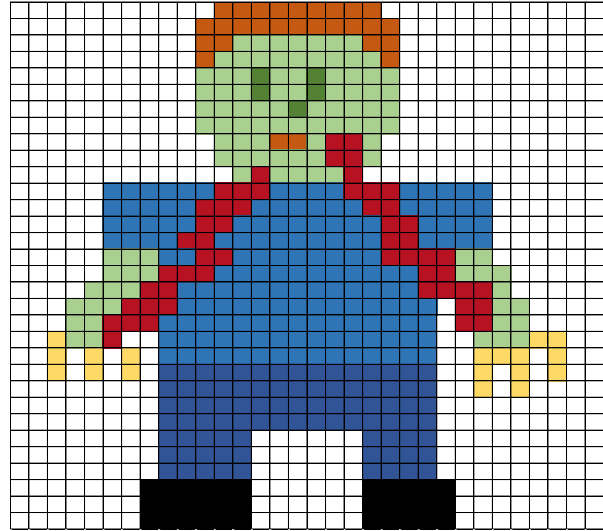
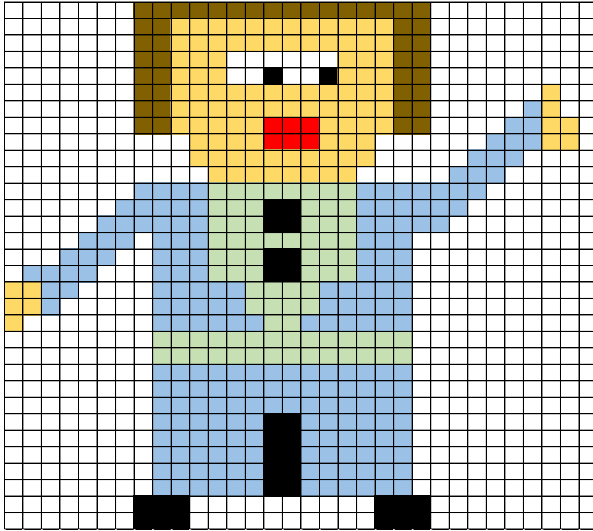
Journey (Continued)

<p>Step 13</p>	<p>Create rule to end the game when the traveler is next to the Chaser</p> <p>Click on Traveler and Edit Behavior</p> <p>Add these rules</p>	 <p>Don't forget the last action – reset simulation!!!</p>
<p>Step 14</p>	<p>Create rule to end the game when the traveler is next to the Goal</p>	<p>No hints here – your turn to figure it out. Use step 13 as a hint.</p> <p>Don't forget the last action – reset simulation!!!</p>
<p>Step 15</p>	<p>Program the <u>Chaser</u> to move randomly</p>	 <p><i>Click on the agent to add behaviors to that agent</i></p>
<p>Step 16</p>	<p>Add walls to your work sheet</p> <p>This is a good time to save the worksheet!</p>	
<p>Step 17</p>	<p>Prevent your Traveler from walking through walls</p> <p>Part a) Add the code shown</p>	<p>Work with the person next to you to figure out how to prevent the Traveler from walking into a wall. Here is one way to think about it... Challenge yourselves to find a different way.</p>  <p><i>Click on the agent to add behaviors</i></p>

Journey (Continued)

	<p>Part b) Add code for the remaining directions</p>	<p><i>to that agent</i></p> <p>Note an important programming point: The two conditions are in the same box...this is an AND statement. It reads like this:</p> <p><i>IF <u>the up arrow is pressed</u> AND <u>the traveller sees ground above him</u></i></p> <p><i>THEN <u>he moves up</u></i></p>
<p>Step 18</p>	<p>Determine where the traveler can cheat</p>	<p>Traveler can cheat by moving off the game ground. Talk with the person next to you about where this can happen on your worksheet.</p>
<p>Step 19</p>	<p>Stop the traveler from cheating</p>	<p>Add rules that make a sound for attempted movement off the ground. Note the importance of rule order for the new rules. Here is an example to prevent the Traveler from moving right off the worksheet. What other direction limit will you need?</p> <div data-bbox="516 848 1175 1081" style="border: 1px solid black; padding: 5px;"> </div> <p><i>Click on the agent to add behaviors to that agent</i></p>

Student Handout 1B: Agent Creation short-cuts



Teacher Instructions:

Part 2 – Making the Chaser Chase the Traveler

Overview:

In this part of the project, students will change the game to make it harder to win and more interesting. Instead of the Chaser moving randomly, the Chaser will now actually move toward the traveler.

Instruction:

Talk to your students about the Chaser.

Consider these prompts:

- Does the chaser really ‘chase’ the traveler? Why or why not?
- Why would we change the game so that he really did chase the traveler?
- How could we change the game so that he could chase the traveler?

[Give students a minute or two to discuss this with the person next to them. Then solicit their ideas.]

[Say to your students] Imagine the traveler emits a scent that the Chaser could smell...would that make it easier for the Chaser to find him? [Give the students an example they can relate to...bacon cooking in the kitchen, the smell of fresh coffee, etc.]

Any of these videos can be used with the class to provide different views of how diffusion of scent and subsequent “hill-climbing” actions are used by different animals

This YouTube video explains how dogs use scent for search and rescue

<https://www.youtube.com/watch?v=XXXU0uKLWo0>

This YouTube video of a MythBusters® segment shows how quickly sharks will respond to fish blood in a pool...students may be disappointed to see that the sharks do not respond to human blood with the same enthusiasm!

<https://www.youtube.com/watch?v=gU9CQT-snIo>

A more scientific explanation of how to determine if sharks can smell blood in the water.

<https://www.youtube.com/watch?v=uqv9EmfkkGE>

Explain: In Part I of this project, the Chaser agent simply moved around randomly on the ground. In this next phase of the design, the Chaser will intelligently seek the Traveler agent using a computational thinking pattern called “hill climbing.”

Imagine the traveler agent emits a scent. Hill climbing is an algorithm to find the direction in which the scent is strongest. The scent will be propagated by the ground agents using a computational thinking pattern called “diffusion.” Diffusion is a fundamental physical process by which matter moves down a gradient from highest to lowest concentrations. The closer to the source of the scent, the greater its amplitude.

Pass out Student Handout 2

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 them to move on to part 3.

Students who need more assistance: Other students (especially those with no Frogger experience) may struggle a bit more. Pairing the student with an experienced student should alleviate many problems.

Vocabulary for ELL Students: Algorithm, propagated, attribute, local variable, diffusion, amplitude, method, parentheses, brackets

Student Handout 2

Part 2 – Making the Chaser Chase the Traveler

So far, your Chaser just moves randomly...he doesn't actually chase the traveler, does he? That's about to change!

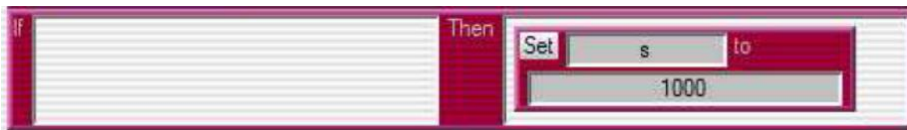
The chaser will intelligently seek the traveler agent using a computational thinking pattern called "seeking." In this instance, we will use a specific method of seeking called hill climbing. Imagine the traveler agent emits a scent. Hill climbing is a procedure or **algorithm** to find the direction in which the scent is strongest.

The scent will spread out or be **propagated** by the ground agents using a computational thinking pattern called "diffusion." Diffusion is a fundamental physical process by which matter moves from areas of highest concentration to areas of lowest concentrations. The closer to the source of the scent, the greater its amplitude.

This phase of the project introduces the concept of an "**agent attribute**," which is unique information that is stored within each occurrence of an agent. Computer scientists call this attribute a **local variable**.

Step 1:

First, let's make sure our traveler gives off a scent. To do this, we need to set an attribute "s" (We have given the arbitrary name of the agent attribute "s" for scent) for the traveler.



This rule says to the Traveler, "If you aren't doing anything else, leave a scent at level 1000 on the ground."



This rule should be AFTER all the other rules for the Traveler, at the end of the list.

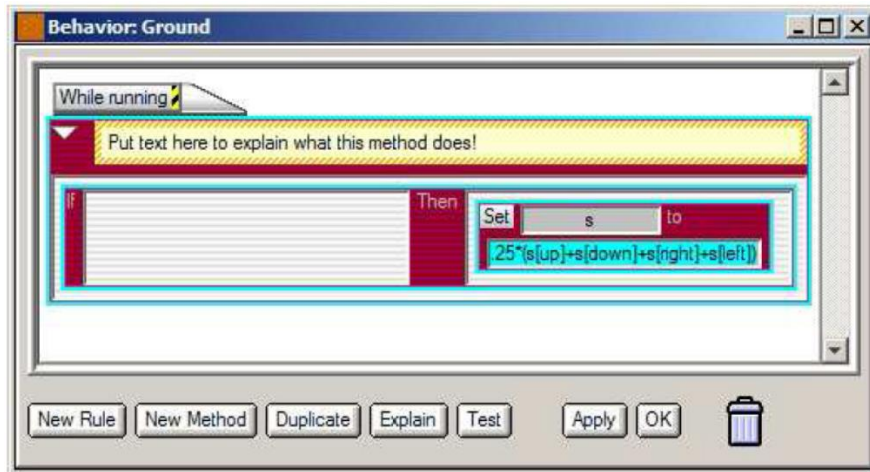
Step 2:

Now, since the scent is diffusing, or spreading out, we need to find the average of the scent from the area around that piece of ground. Think of it as the smells are coming in from the North, South, East and West. The smell in the center, then, is the average of these four smells. How will you create that programmatically?

The ground agent will have the behavior below; the single action is to calculate and store the average of the four surrounding agents' agent attributes. Remember, you used the arbitrary name of the agent attribute "s" (for scent).

The "set" action sets each ground agent's attribute "s" to the average of the attributes in the agents above, below, and on each side:

$$s = 0.25*(s[\text{up}]+s[\text{down}]+s[\text{right}]+s[\text{left}])$$



Why do we multiply by 0.25?
When you find the average of a set of numbers, you add them up and divide by the number of numbers.

In this case, dividing by 4 is the same as multiplying by 0.25



Match both the parentheses “(” and the brackets “[” as shown in the equation.

A METHOD is like a set of rules to follow in a specific situation. You can create a METHOD by clicking “New Method” at the bottom of the behavior box for the Chaser.

Step 3:

For the Chaser to know which way to walk, he has to determine where the scent is the strongest. If this were real life, he would smell up, smell down, smell left and smell right. Wherever the smell was strongest, he would walk in that direction. We need to program the Chaser to do this.

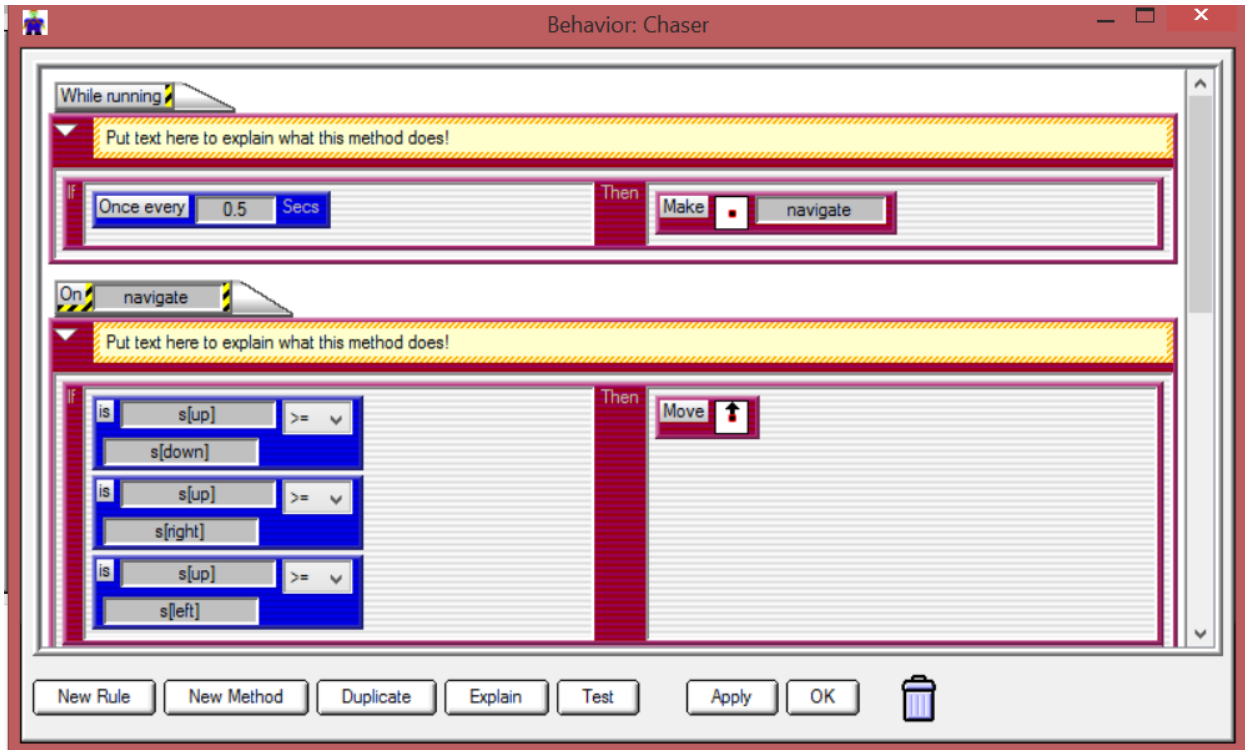
We will create a METHOD for the Chaser to follow a set of rules.

Journey (Continued)

Take a look at the programming below. Here's what it says...

ONCE EVERY 0.5 seconds, follow the Navigate procedure.

IF the smell **above you** is greater than or equal to any of the other smells in different directions (down, left or right), THEN move up.



Now, add the rest of the rules so that the Chaser knows what to do if the smell down (s[down]) is greater...What if the smell to the left is greater? What about the smell to the right?

Run your game to see if the Chaser chases the Traveler! If it isn't working, it's time to do some troubleshooting.

Check the following:

- Location of the rules
- Use of Method
- Use of parentheses and brackets

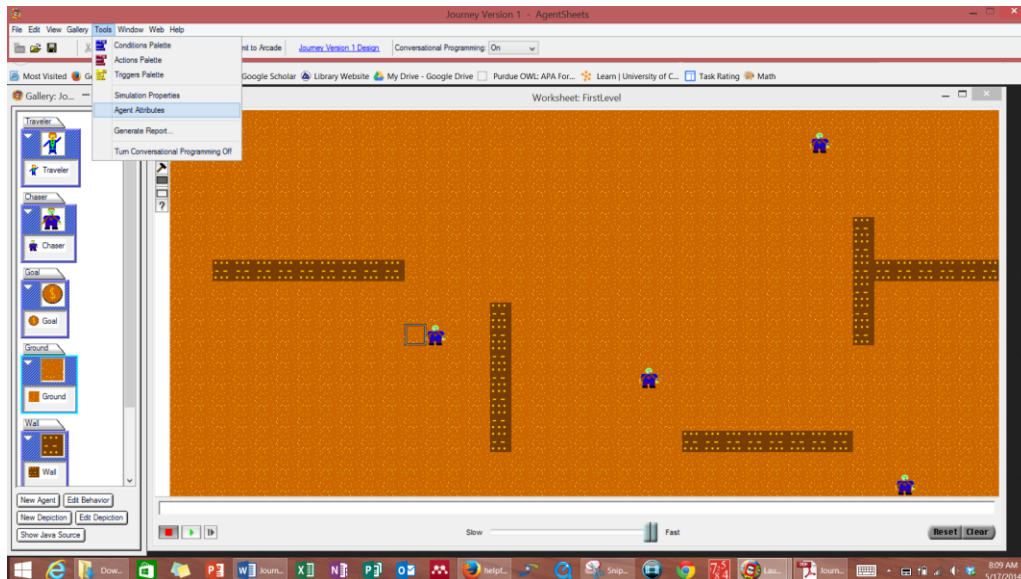
Student Handout:

Troubleshooting Guide for Journey Part II

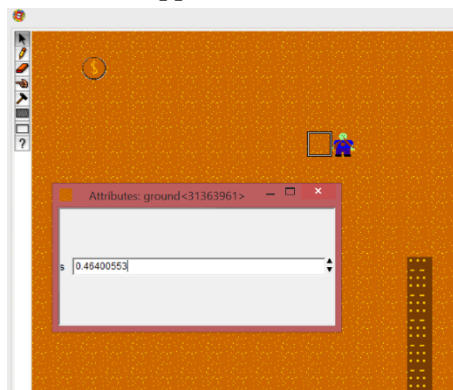
Diffusion and Hill Climbing

Step 1:

To determine what is happening in your game, it is sometimes helpful to look at the agent attributes. On your worksheet, click run and then click stop. Do not reset at this point. Your traveler has now 'left his scent' on the ground. You can see his scent (the value of s) by clicking anywhere on the ground and then on Tools>>Agent Attributes



A box will appear that lists the attribute value for that agent. You can see it in the box below. In this box is the value of the scent to the left of this Chaser. By checking the attributes of the four boxes around the Chaser (up, down, left and right) and then running the game again, you can see if your Chaser is doing what you expected him to do. If he isn't, go back and check your rules and methods. Some things to consider:



- **Spelling**
- **Parentheses/Brackets**
- **Rule Order**

Teacher Instructions

Part 3 – Making the game harder – Polling and Broadcast

Background:

In a classroom, when students are working on an assignment, teachers regularly ‘poll’ the room to see if everyone is done yet. S/He does this by asking students to raise their hand if they are still working. If no one raises his/her hand, the teacher knows everyone is done. Once everyone is done, the assignment is finished. Students will use this same concept to change their game to make it more challenging.

Introduction to students:

Using the example of the classroom, guide your students through a discussion of how to poll for answers. Now, tell them they are going to use this same concept to change their game. This time, the game looks like this:

Rather than standing by the goal to win, the traveler will find there are many more goals to collect. Now, in order to win, the traveler will run around and pick up all of the goals before being caught by a chaser.

Give students a couple of minutes to discuss this programming activity.

Consider these prompts:

- Who will poll (look to see if there are still more goals to be collected)
- What stops the game?
- What steps (code) will change?
- How would the worksheet change?

[Give students a minute or two to discuss this with the person next to them. Then solicit their ideas.]

Students will struggle with the idea of who polls. Introduce the idea of a controller, an agent that is responsible for tracking the number of goals left on the worksheet. Remind the students that they should take time to think through each programming step so they can use these skills later.

Hand out worksheet 3

Differentiated Instruction:

Students who need a challenge: Some students with more fluency in programming may finish this very quickly – be prepared for them to move on to the challenge.

Students who need more assistance: Other students (especially those with no Frogger experience) may struggle a bit more. Pairing the student with an experienced student should alleviate many problems.

Note that this is a challenging lesson for students – plan extra time and additional help from more experienced students.

Vocabulary for ELL Students: polling, broadcast, increment

Student Handout 3

Part 3:

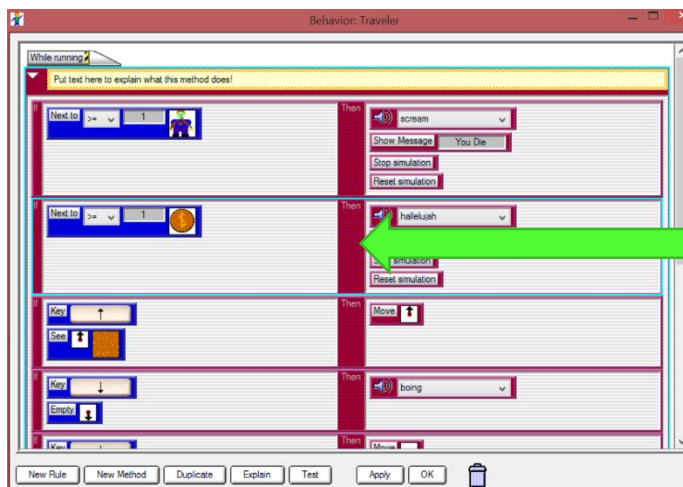
Making the game harder – Polling and Broadcast

In this enhancement to the Journey project, the Traveler must “collect” – that is get next to – multiple goals in order to win. The game does not end until all of the goals are reached.

To accomplish this, we introduce the concept of a SIMULATION PROPERTIES, which are bits of information that are shared among all agents in a project.

We create a new agent, the “Controller” to manage the process of polling the goals to determine when they are all “collected”; that is, when there are none left on the worksheet. Finally, we must change the behavior of the traveler agent so that it no longer declares the game is over when it gets next to a goal.

Step 1: Remove the rule from the Traveler that the game is over when next to the goal.



Highlight the rule by clicking on the bar between the condition and action. Then press the delete button on your keyboard.

Step 2: Create a Controller agent



Create a Controller agent. **Set the agent on your worksheet.** The agent does not have to be in the active part of the worksheet – he can be on the white space if you’d like.

Journey (Continued)

Step 3: Counting up the goals to see if you won

Imagine this conversation...

The teacher has given an assignment to the class and wants to know if everyone is finished. She says to the class, "Put your hand up if you are still working." Hands go up. She counts them – there are five students still working. "Okay, put your hands down and keep working."

A few minutes later, she does it again. She says to the class, "Put your hand up if you are still working." Hands go up. She counts them – there are two students still working. "Okay, put your hands down and keep working."

A few minutes later, she does it again. She says to the class, "Put your hand up if you are still working." This time, no hands go up. "Everyone is done, put your books away"

That's what this programming will look like. The Controller will say, "Goal count starts at zero" (like the classroom, no hands are up when the teacher asks who is still working). When the goals 'hear' the Controller ask (broadcast) the question, the goals respond back (raise their hands). The controller counts the goals. If the answer is more than zero, nothing happens and the game continues. If the answer is zero (meaning that there are no remaining goals on the board), the game ends.

First, we need to create a simulation property called 'goals'. This property is the count of the hands. To do this, go to Tools>>Simulation Properties>>New. Type in goals and click Save.

To refer to this property, we use the symbol @. (This is similar to how we use the hashtag to tag posts, like #simulation.) Therefore, when we refer to the goals, we type @goals.



Journey (Continued)

There are three parts to the Controller behavior.

Part 1: Set the number of goals to zero. (this is like the teacher saying “hands down”)

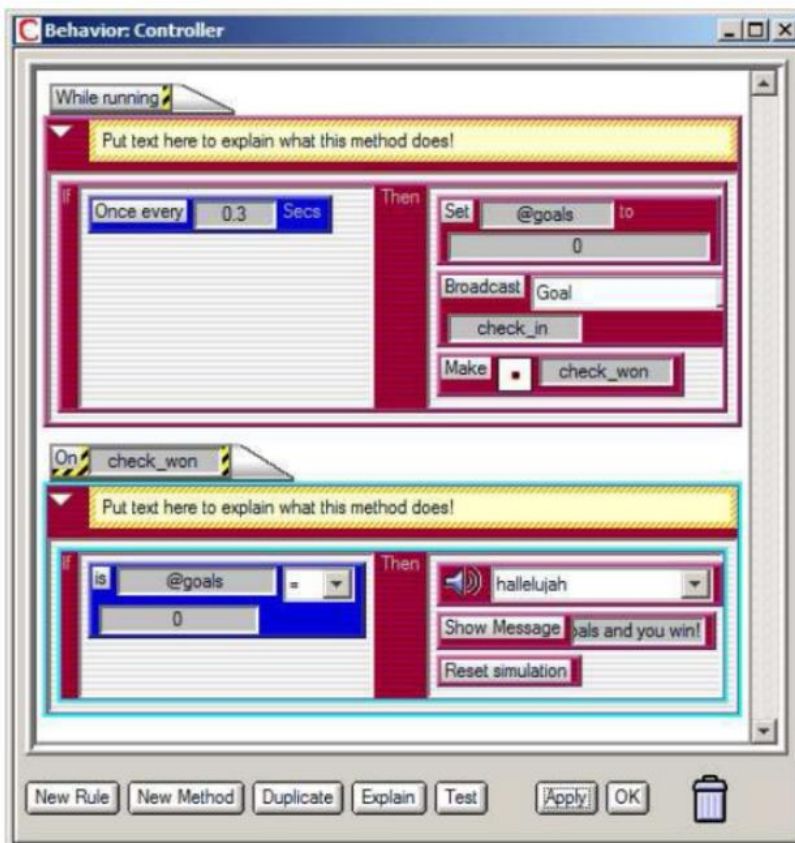
Part 2: Ask the goals (broadcast/polling) if they are still on the board

Part 3: Use the count of the goals to see if the game is done.

How do Simulation Properties Work?

In the “While Running” method, the control first sets the simulation property “@goals” to zero. Then it broadcasts a signal to all goals. This broadcast is called polling. Finally, the controller calls upon the “check_won” method to determine whether the game is won. This is true only if there are no goals remaining, which is determined by the @goals simulation property being zero. If any goals are left, we will see that this simulation property will be greater than zero

Your behavior for the Controller should look like this:



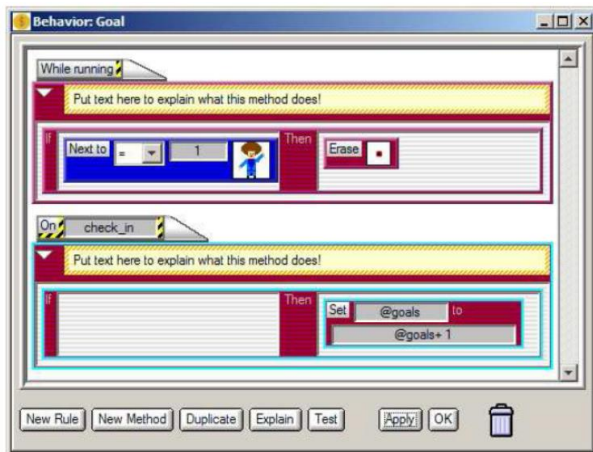
Journey (Continued)

Goal behavior changes: There are two behavior changes required for the goal agent.

- The first step is to have the goal be collected by the traveler. We can simulate this by **erasing** the goal when the Traveler gets next to it.
- The second behavior change for the goal agent is to respond to the “poll” (broadcast) called *check_in* from the Controller, to update the @goals simulation property.

Helpful
Tips

This second change is in the form of a separate method; it is not part of the continually running “While Running” method, since it only runs when called by the controller agent.



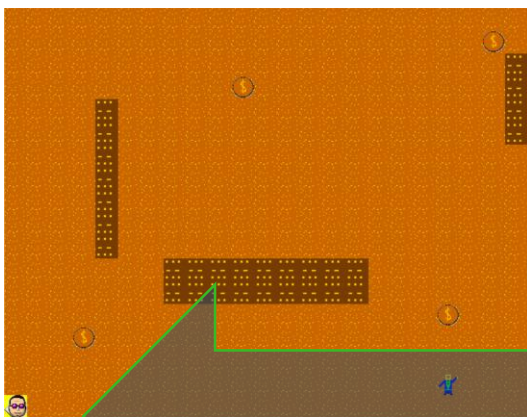
During *check_in*, each remaining goal agent will increase (or **increment**) the @goals simulation property. If no goal agents remain, then the @goals property will be zero, which the controller agent will detect and declare the game won.

Helpful
Tips

Before you test this, check your worksheet: We have placed one Controller on the bottom left of the worksheet. Note that the Controller agent does not need to be in the active area of the worksheet, since it does not interact directly with any agents.

In addition, we have placed several additional goals on the worksheet, so that the traveler must “collect” – that is, get next to -- each of them in order to win the game.

You can also add more Chasers to make the game more difficult!



Student Handout:

Troubleshooting Guide for Journey Part III

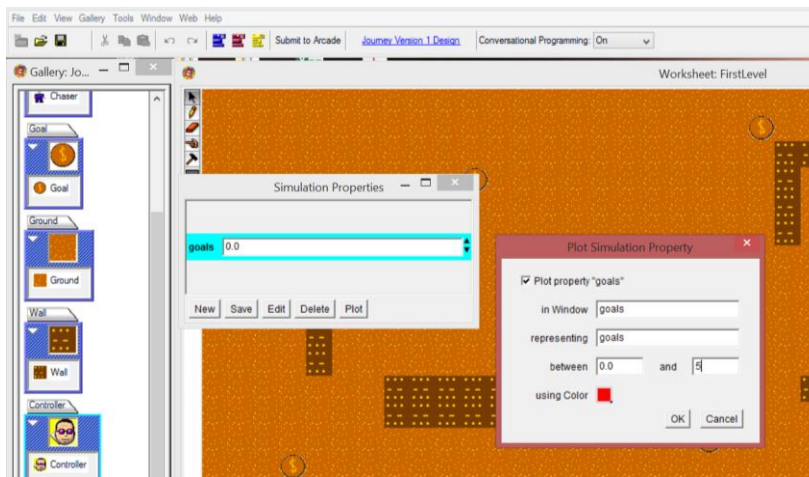
Polling and Broadcast

Common Problems:

1. Is your Controller agent on the worksheet?
2. Did you type in @goals where you needed to?
3. Do you refer to the correct agents in each step?

More detailed troubleshooting:

To determine what is happening in your game, it is sometimes helpful to look at what the simulation property is doing. To do this, have your worksheet open as well as the simulation property box (Tools>>Simulation Property). Click on the property, and then click on Plot. It will look like this:



Click Plot Property 'goals'. Change the plot to graph between 0 and the total number of goals on your worksheet

This will provide a graph that shows you what's happening 'behind the scenes' while you play the game. This information will help you determine where a mistake may be. For example, if the goals never goes above 0, there is a problem with the method 'check_in' or the broadcast. If the goals goes to zero but the game doesn't end, there is a problem with the game ending commands with the controller.

End of Unit Review Sheet - Journey

A) The main computational thinking patterns we reviewed were:

- 1) **Cursor Control**: intentionally moving an agent.
 - a. Using keyboard keys to move an agent.
 - b. Example is moving the Traveler.
- 2) **Absorb**: deleting agents on the screen.
 - a. Use the “Erase” action in Agent Sheets.
 - b. Examples are erasing the goals.
- 3) **Collision**: when 2 agents collide (run into each other).
 - a. Use the “See” condition
 - b. Use the “Stacked” condition, OR
 - c. Use the “Next to” condition.
 - d. Examples are the collecting goals and winning the game.

B) The main NEW computational thinking patterns we learned were:

- 1) **Diffusion**: spreading the scent (smell) of an agent across a medium (like the background). We use an agent attribute (like $s = 1000$) on the agent with the smell, and we diffuse the smell by setting the attribute on the background using the average of the 4 smells around it; like the smell on the city background,

$$s = (s[\text{left}] + s[\text{right}] + s[\text{top}] + s[\text{bottom}]) / 4.$$

- 2) **Hill Climbing**: following the highest scent. It only works if there is diffusion done with it, so they go hand in hand. Example is the method we created on the chaser to follow the highest value of the scent “s” around him.
- 3) **Broadcasting**: is when we “shout out” to all agents of a certain type requesting them to execute a specific method.
 - a. Use the “broadcast” action in Agent Sheets.
 - b. Example is the broadcast to the Controller - the method `check_in`” to check in with the goals to see if they are still there.

C) Other concepts we covered in Agent Sheets are:

- 1) Troubleshooting the simulation, and considering rule order.
- 2) Using sounds and messages in the game.
- 3) Timing our actions using the “Once every” condition.

Student Handout 4A:

Ice Arrows 1.0 Challenge

Before you start this challenge:

You must have a complete basic journey game with a Traveler who wins if s/he reaches the treasure and Chasers who either move randomly or chase the Traveler. The Traveler loses if a Chaser gets too close. The worksheet should have walls that the Traveler and Chasers can not cross.

Description of the Challenge:

- Your Traveler shoots ice arrows up towards the top of the world when the space bar is typed.
- A Chaser hit by a moving ice arrow freezes and cannot move.
- A frozen Chaser hit by a moving ice arrow unfreezes and can move again.
- Ice arrows should not go through walls or stack up in piles.

Gamelet Design Activity:

In the description above, circle nouns to identify the agents and underline the verbs to identify actions associated with each agent. Mark adjectives to identify new shapes for an agent.

Create new agent: ice arrow

- draw an upward facing ice arrow

Create new depiction (image): frozen Chaser

- select your chaser agent
- click on the New Depiction button at the bottom of the gallery window
- create the frozen Chaser
- The Chaser's depiction stores its **state**: frozen or unfrozen.

Create New Rules:

Traveler

- Add a rule so that an ice arrow is generated (fired upwards) when the space bar is hit.
- Where should this rule appear?
 - Above or below the win rule?
 - Above or below the move rules?



Remember that special cases appear above default behavior!



Journey (Continued)

Ice Arrow

- Add a rule that makes the ice arrow move up.
- Add a rule so that your ice arrows are ABSORBED BY (do not go through) the walls.
- Add rules so that the ice arrow “hits” the Chaser right above it using the Make action: This action should be read as “make the agent above me check the rules in its “hit” method”. The ice arrow must “hit” both frozen and unfrozen chasers!
 - In what order should these rules appear in the ice arrow while running method?
 - Order your rules with special cases at the top and default behavior at the bottom


Chaser

- Use the New Method button at the bottom of the Chaser behavior window to add a method named “hit”. The method name must exactly match the name in the ice arrow Make action!
- Add a rule to the hit method that freezes an unfrozen Chaser.
- Add a rule to the hit method that unfreezes a frozen Chaser.



Does the rule order matter?

Test your new feature

- If the agents’ behavior does not match the changes you have made, click on each agent’s apply button.
- If your Chaser does not stay frozen, add an action to the ice arrow rule so that the ice arrow is erased as soon as it hits the Chaser. Otherwise, the arrow will freeze and then unfreeze the Chaser. Use the Erase action, , which should be read “Erase me”.
- Do frozen Chasers move? Add a condition to the Chaser rule move rule so that only unfrozen Chasers can move.
- Does the Traveler die and end the game when s/he is next to a frozen Chaser? Check whether the Traveler is next to 1 or more unfrozen Chasers.
- Does the ice arrow freeze or unfreeze a chaser on the other side of a wall? Reorder your rules so that the rule that erases an arrow over a wall comes first and has priority over hitting a Chaser right above it in the square next to the wall. Then the ice arrow will be erased before it can do anything to a Chaser.
- If your ice arrow does not move across the worksheet, check whether your game is running so fast that the arrow movement is not visible and consider adding a timer condition to the if side of your ice arrow move rule so that the ice arrow moves slowly enough to be visible.
- Do your arrows stack up on the edge of the worksheet? Add walls along the edge of the worksheet to absorb arrows.

Student Handout 4B:

Ice Arrows 1.0 Challenge Tutorial

Before you start this challenge:

You must have a complete basic journey game with a Traveler who wins if s/he reaches the treasure and Chasers who either move randomly or chase the Traveler. The Traveler loses if a Chaser gets too close. The worksheet should have walls that the Traveler and Chasers can not cross.

Design Challenge:

Your Traveler shoots ice arrows **up** towards the top of the world when the space bar is typed. A Chaser hit by an ice arrow freezes and can't move or kill the Traveler. A frozen Chaser hit by an ice arrow unfreezes and moves again. Ice arrows do not go through walls or pile up on the worksheet.

Gamelet Design Activity:

Circle nouns to identify the agents and underline the verbs to identify actions associated with each agent. Mark adjectives to identify new shapes for an agent.

Create new agent: ice arrow

- Use the New Agent button at the bottom of the gallery window.
- Draw an upward facing ice arrow

Create new depiction: frozen Chaser

- Select your Chaser by clicking on it.
- Click on the New Depiction button at the bottom of the agent gallery window.
- Draw a frozen Chaser! Make sure that it looks different enough from a regular Chaser that you can identify from a small picture.
- The Chaser's depiction stores its **state**: frozen or unfrozen.

Ice Arrow 1.0



Create arrows that freeze and unfreeze the Chasers

Journey (Continued)

Create New Rules:

Traveler

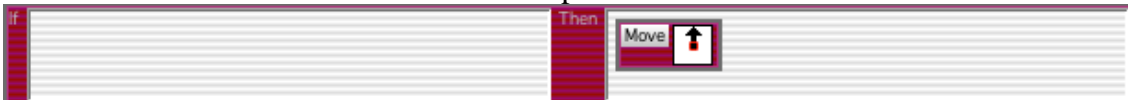
- Add a rule so that an ice arrow is fired upwards when the space bar is hit. (generate CTP)



- Where should this rule appear? Above or below the win rule? Above or below the move rules? Remember that special cases appear above default behavior. *Put this rule below the win rule and the lose rule if you have it in this agent. If you put it above the movement rules, you will make the Traveler fire arrows rather than move if both the space bar and arrow keys are typed simultaneously*

Ice Arrow

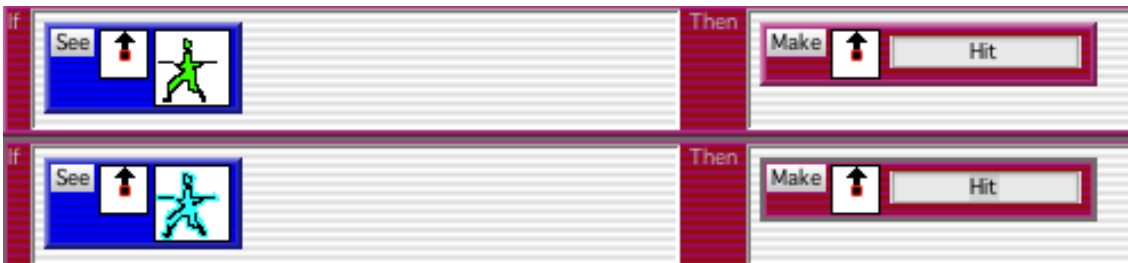
- Add a rule that makes the ice arrow move up.



- Add a rule so that your ice arrows do not go through walls. (absorb CTP)



- Add rules so that the ice arrow “hits” the Chaser right above it using the Make action. This Make action should be read as “make the agent above me check the rules in its “Hit” method”. The ice arrow must “hit” both frozen **and** unfrozen chasers!

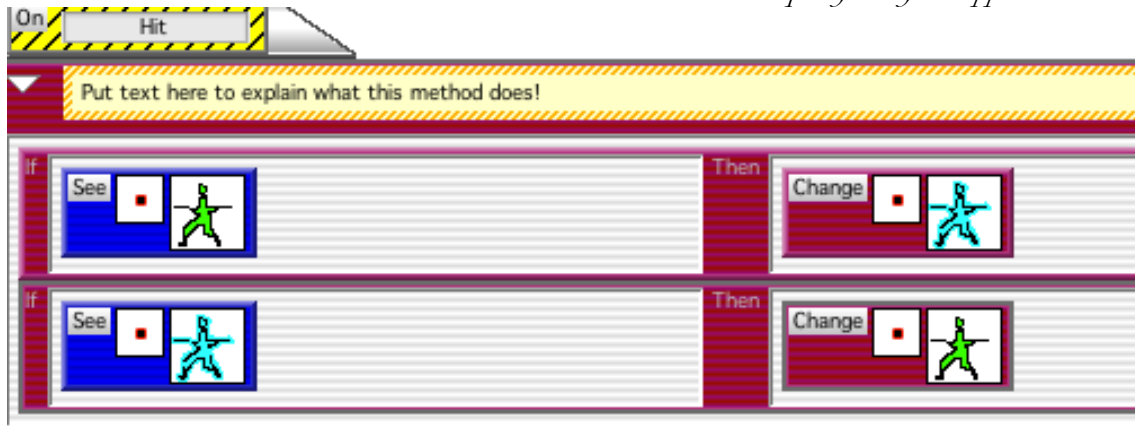


- In what order should these rules appear in the ice arrow while running method? Order your rules with special cases at the top and default behavior at the bottom. *You can put the Hit rules first, followed by the wall rule. The move up rule should appear last.*

Journey (Continued)

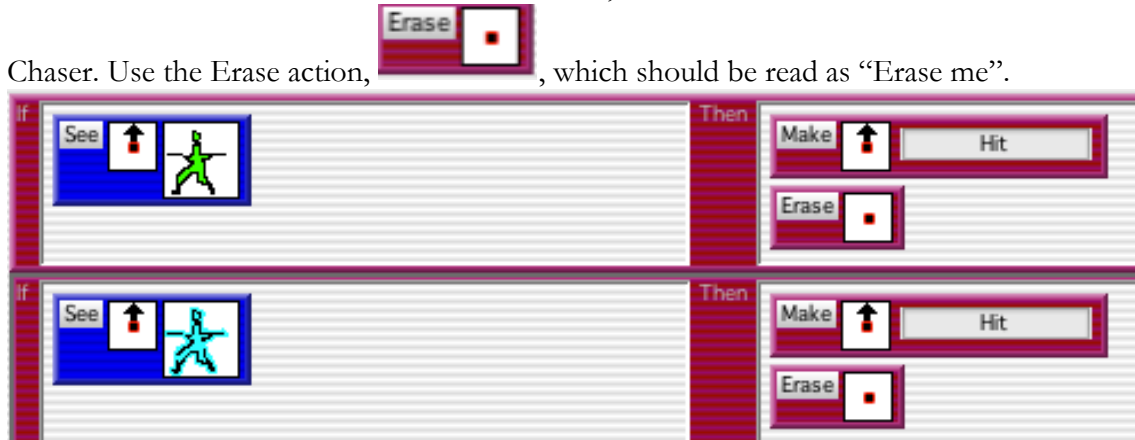
Chaser

- Use the New Method button at the bottom of the Chaser behavior window to add a method named “hit”. The method name must exactly match the name in the ice arrow Make action!
- Add a rule to the hit method that freezes an unfrozen Chaser.
- Add a rule to the hit method that unfreezes a frozen Chaser.
- Does the rule order matter? *Not in this case. Both rules are almost equally likely to happen.*

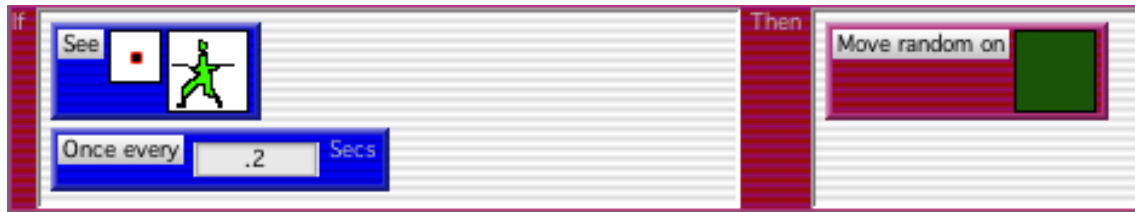


Test your new feature

- If the agents' behavior does not match the changes you have made, click on each agent's apply button.
- If your Chaser does not stay frozen, add an action to the ice arrow rule so that the ice arrow is erased as soon as it hits the Chaser. Otherwise, the arrow will freeze and then unfreeze the



- Do frozen Chasers move? Add a condition to the Chaser rule move rule so that only **unfrozen** Chasers can move.

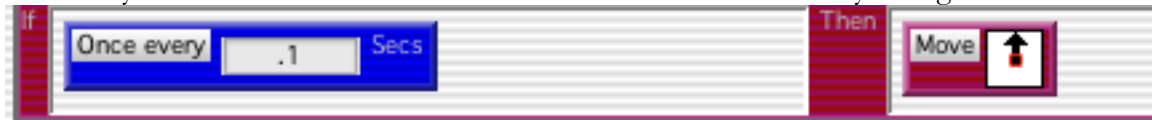


Journey (Continued)

- If the Traveler dies and ends the game when s/he is close to a frozen Chaser, check whether the Traveler is next to an unfrozen Chaser.



- If your ice arrow does not move across the worksheet, check whether your game is running so fast that the arrow movement is not visible and consider adding a timer condition to the if side of your ice arrow move rule so that the ice arrow moves slowly enough to be visible.



- Does the ice arrow freeze or unfreeze a chaser on the other side of a wall? Reorder your rules so that the rule that erases an ice arrow over a wall comes first and has priority over hitting a Chaser right above it in the square next to the wall. Then the ice arrow will be erased before it can do anything to a Chaser.
- Do your arrows stack up on the edge of the worksheet? Add walls along the upper edge of the worksheet to absorb arrows.

Student Handout 4C:

Ice Arrows 2.0 Challenge

Before you start this challenge:

Ice Arrows 1.0 must be completed and tested! The Traveler should shoot ice arrows upwards that freeze unfrozen Chasers and unfreeze frozen Chasers.

Design Challenge: The Traveler turns to face whichever direction s/he is moving: up, down, left or right. Make the Traveler fire ice arrows at the Chasers in whichever direction the Traveler is facing. Ice arrows will move in the direction that they are fired: up, down, left or right.

Gamelet Design Activity: read the challenge and identify the new depictions that must be added to each agent.

Create New Depictions

Ice Arrow: Add new depictions so that the ice arrow moves point-first in all 4 directions

- Select the ice arrow agent and click on the New Depiction button at the bottom of the gallery window and draw a new depiction. Or go to the Gallery menu (to the right of the word AgentSheets) and find the Duplicate Depiction option. Pick the 3 different rotations of your upwards-facing ice arrow off the list of choices.
- Make sure to make them large enough and different enough that you can identify them from a small picture.
- Make an up-facing arrow, a down-facing arrow, a left-facing arrow and a right-facing arrow.
- The ice arrow's depiction stores the ice arrow's **state**: which direction it moves.

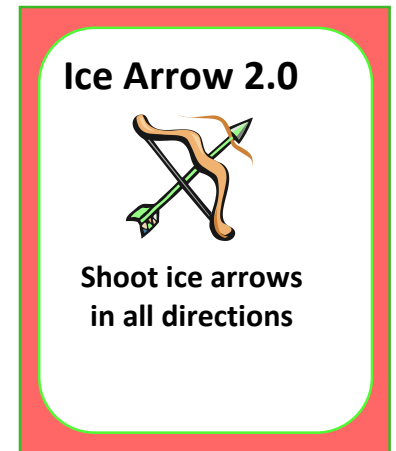
Traveler: Add new depictions to the Traveler so s/he faces in the direction s/he is moving:

- The direction that the Traveler faces determines which ice arrow will be generated.
- The Traveler's depiction stores the Traveler's **state**: which way s/he is facing.
- Go to the Gallery menu (to the right of the word AgentSheets) and find the Duplicate Depiction option. Find the 3 different rotations of your Traveler on the list of choices. You should have an up-facing Traveler, a down-facing Traveler, a left-facing Traveler and a right-facing Traveler when you are done.

Create New Rules for the Traveler:

Edit the Traveler's rules so the depiction changes each time the Traveler moves a different direction:

- Add an action to each of the Traveler's move rules so that the Traveler's depiction changes to match the direction of movement. For example, typing the left arrow key changes the Traveler's depiction to a left-facing depiction and the Traveler moves one square left.



Journey (Continued)

Change the Traveler's rule to fire the ice arrow:

- Use a method to decide which direction the arrow should be fired because now there are 4 possibilities to choose from.
- Remove the action from the rule with the key = spacebar condition and replace it with



which should be read as “make **me** do the method named FireArrow”. The dot means “me”.

Create the FireArrow method in the Traveler's rule window and add 4 rules to it:

- Click on the New Method button at the bottom of the Traveler's rule window.
- Edit the method name so it exactly matches the name in the Make action!
- Make the first rule by adding a condition that checks which way the Traveler is facing and then adding an action to generate a new ice arrow facing the same way.
- You may either generate the new ice arrow right on top of the Traveler by using the dot or make it appear next to the Traveler by editing the dot to be an arrow facing the same way as the Traveler.
- Duplicate this rule 3 times and edit the duplicated rules so that they generate new ice arrows in the other 3 directions.

Create New Rules for the Ice Arrow:

Change the Ice Arrow's movement rule so that it calls a method with rules that check which way the arrow should move:

- Change the original up-arrow move rule in the ice arrow while-running method by removing



the move action and adding this action: which means “make **me** do the method named **Fly**”.

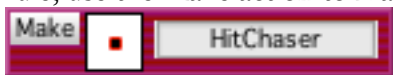
- Add a timer condition to the if side of this rule to control how rapidly the arrow moves. Make it move slowly enough to be visible!

Create the fly method and add 4 rules to it so that the ice arrow continues to move in the direction its point faces:

- Click on the New Method button at the bottom of the Ice Arrow's rule window.
- Edit the method name so it exactly matches the name in the Make action.
- Create the first move rule by adding a condition that checks what the ice arrow looks like and adding an action that makes it move in the matching direction. For example, the up-arrow should move up.
- Duplicate the first rule 3 times and edit each of these rules so that the ice arrow can move in the other 3 directions.

Edit the ice arrow rules so that each Ice Arrow depiction can freeze or unfreeze a Chaser:

- Change the hit rules in the while running method so that you have two rules which detect when an ice arrow is **near** an unfrozen Chaser and when it is **near** a frozen Chaser. In each rule, use the Make action to make the ice arrow do a new method, HitChaser:



Journey (Continued)

- The HitChaser method rules will decide which way the ice arrow is facing and where the Chaser is relative to the ice arrow. Once the HitChaser rule has checked the ice arrow depiction, it can send a Hit message to the Chaser in front of the ice arrow point so that it either freezes or unfreezes, depending on its state.
- For example, if the ice arrow is a downwards-facing ice arrow, it will send a Hit message to

the Chaser below it: 

- Make 3 more rules in the HitChaser method to detect the other 3 ice arrow depictions and send messages to Chasers above, left or right of the ice arrow.
- Why did we create the HitChaser method? The logic of what happens when an ice arrow is next to a Chaser is the same whether the Chaser is frozen or unfrozen so we can create a single set of rules in HitChaser that will cover all 4 ice arrow depiction possibilities. The 2 rules in the main ice arrow while-running method make sure that both frozen and unfrozen Chasers can be hit by ice arrows.


Testing

- If the agents' behavior does not match the changes you have made, click on each agent's apply button.
- Test that your Traveler can fire arrows in all 4 directions.
- Do ice arrows build up on the edges of your worksheet? Rearrange your walls to absorb them.
- If you have an arrow that does not move, check that the depiction in the New action in the Traveler's rules matches the depiction in the Move action in the ice arrow's rules.
- If you get error messages from other agents on the worksheet saying that they do not know how to respond to a "Hit" message, you must change the HitChaser rules to test that the ice arrow is about to hit a Chaser and not some other agent. Add a condition that checks for the Chaser agent by name rather than image to **each** HitChaser rule. Here is the condition that



must be added to the upwards-facing ice arrow rule:

- If your Chaser is not frozen by a direct hit, the problem may be that the ice arrow is next to the Chaser and unfreezes it immediately after freezing it. How can you guarantee that an ice arrow will not cause the Chaser to change rapidly back to unfrozen? Add an "Erase me"

action, , to each of the HitChaser rules so that the ice arrow sends a Hit message to the Chaser and instantly disappears.

- Now test that your Traveler can shoot in all 4 directions and can freeze and unfreeze a Chaser with all 4 ice arrows. You may need to move the Traveler and the Chaser into position in order to test each direction.

Student Handout 4D:

Ice Arrows 2.0 Challenge Tutorial

Before your start this challenge:

Ice Arrows 1.0 must be completed and tested! The Traveler should shoot ice arrows upwards that freeze unfrozen Chasers and unfreeze frozen Chasers.

Create New Depictions

Ice Arrow: Add new depictions so that the ice arrow moves point-first in all 4 directions

- Select the ice arrow agent and click on the New Depiction button at the bottom of the gallery window and draw a new depiction. Or go to the Gallery menu (to the right of the word AgentSheets) and find the Duplicate Depiction option. Pick the 3 different rotations of your upwards-facing ice arrow off the list of choices.
- Make sure to make them large enough and different enough that you can identify them from a small picture.
- Make an up-facing arrow, a down-facing arrow, a left-facing arrow and a right-facing arrow.
- The ice arrow's depiction stores the ice arrow's **state**: which direction it moves.

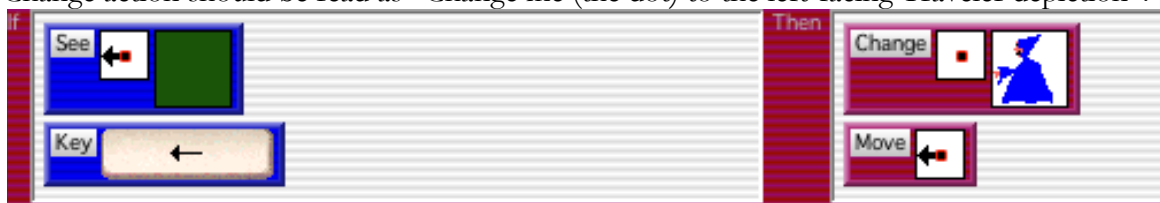
Traveler: Add new depictions to the Traveler so s/he faces in the direction s/he is moving:

- The direction that the Traveler faces determines which ice arrow will be generated.
- The Traveler's depiction stores the Traveler's **state**: which way s/he is facing.
- Go to the Gallery menu (to the right of the word AgentSheets) and find the Duplicate Depiction option. Find the 3 different rotations of your Traveler on the list of choices. You should have an up-facing Traveler, a down-facing Traveler, a left-facing Traveler and a right-facing Traveler when you are done.

Create New Rules for the Traveler:

Edit the Traveler's rules so the depiction changes each time the Traveler moves a different direction:

- Add an action to each of the Traveler's move rules so that the Traveler's depiction changes to match the direction of movement. For example, typing the left arrow key changes the Traveler's depiction to a left-facing depiction and the Traveler moves one square left. The Change action should be read as "Change me (the dot) to the left-facing Traveler depiction".



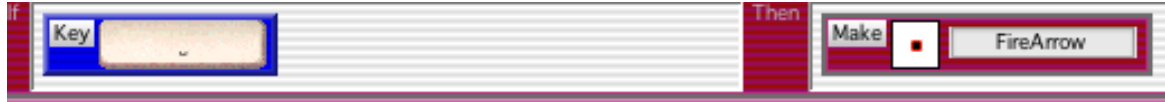
Journey (Continued)

Change the Traveler's rule to fire the ice arrow:

- Use a method to decide which direction the arrow should be fired because now there are 4 possibilities to choose from.
- Remove the action from the rule with the key = spacebar condition and replace it with

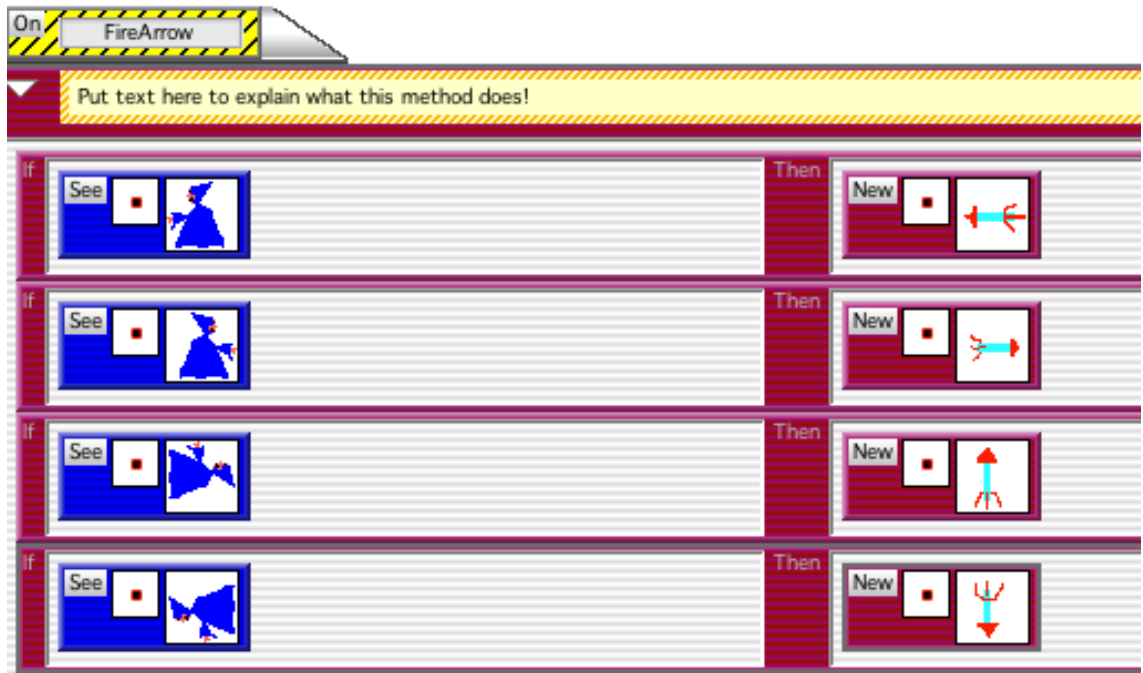


which should be read as “make **me** do the method named FireArrow”. The dot means “me”. The rule looks as follows:



Create the FireArrow method in the Traveler's rule window and add 4 rules to it:

- Click on the New Method button at the bottom of the Traveler's rule window.
- Edit the method name so it exactly matches the name in the Make action!
- Make the first rule by adding a condition that checks which way the Traveler is facing and then adding an action to generate a new ice arrow facing the same way.
- You may either generate the new ice arrow right on top of the Traveler by using the dot or make it appear next to the Traveler by editing the dot to be an arrow facing the same way as the Traveler.
- Duplicate this rule 3 times and edit the duplicated rules so that they generate new ice arrows in the other 3 directions.



Create New Rules for the Ice Arrow:

Change the Ice Arrow's movement rule so that it calls a method with rules that check which way the arrow should move:

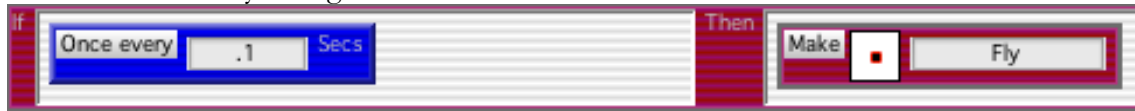
Journey (Continued)

- Change the original up-arrow move rule in the ice arrow while-running method by removing



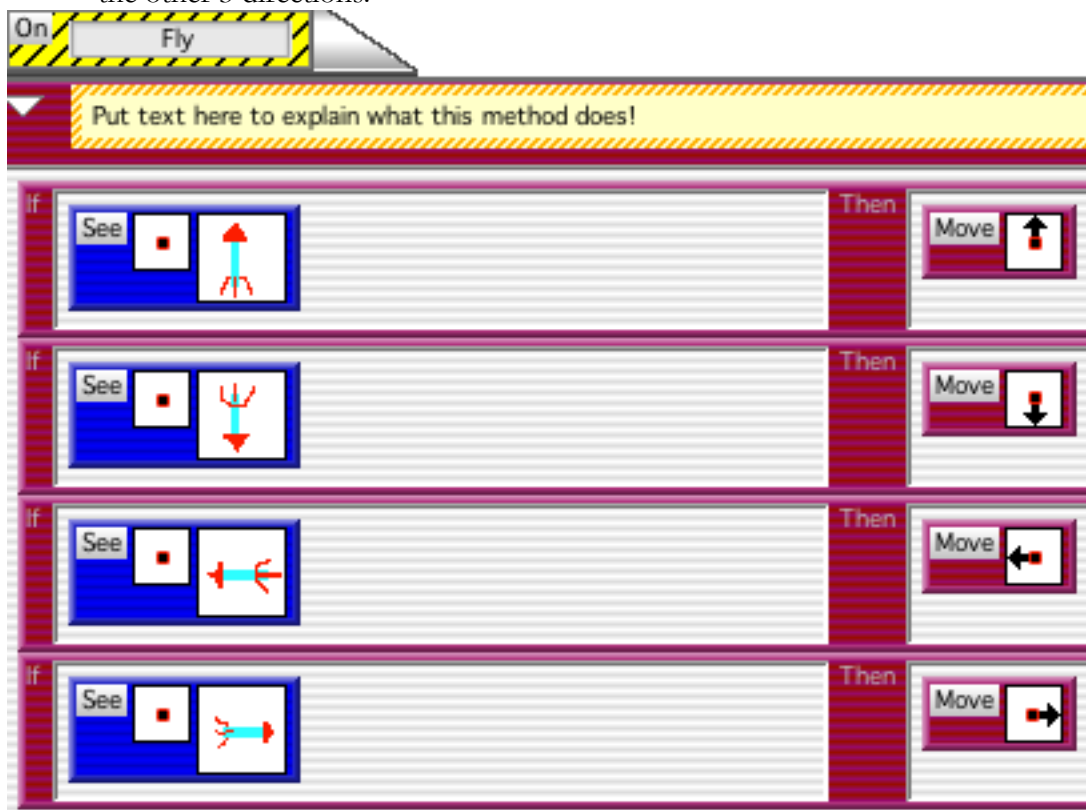
the move action and adding this action: which means “make **me** do the method named **Fly**”.

- Add a timer condition to the if side of this rule to control how rapidly the arrow moves. Make it move slowly enough to be visible! The rule should look as follows:



Create the fly method and add 4 rules to it so that the ice arrow continues to move in the direction its point faces:

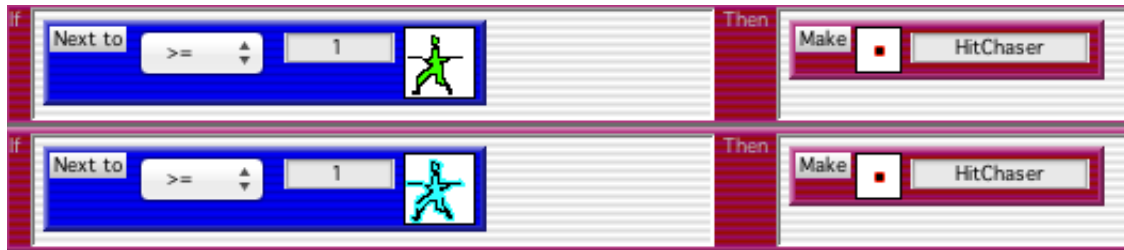
- Click on the New Method button at the bottom of the Ice Arrow’s rule window.
- Edit the method name so it exactly matches the name in the Make action.
- Create the first move rule by adding a condition that checks what the ice arrow looks like and adding an action that makes it move in the matching direction. For example, the up-arrow should move up.
- Duplicate the first rule 3 times and edit each of these rules so that the ice arrow can move in the other 3 directions.



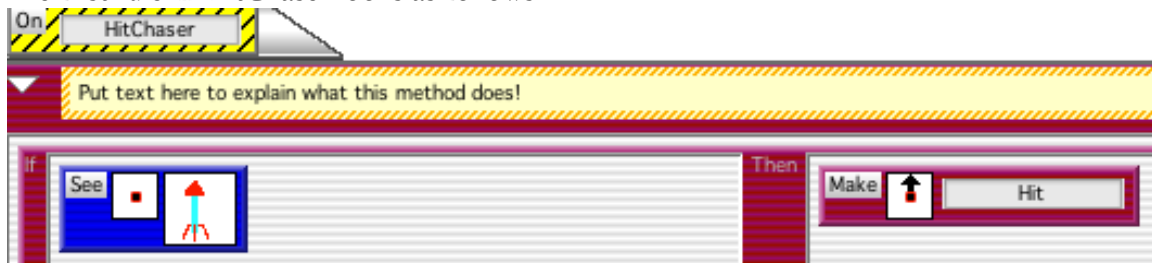
Edit the ice arrow rules so that each Ice Arrow depiction can freeze or unfreeze a Chaser:

- Change the hit rules in the while running method so that you have two rules which detect when an ice arrow is near an unfrozen Chaser and when it is near a frozen Chaser. In each rule, use the Make action to make the ice arrow do a new method, HitChaser. The rules

should look as follows:



- The HitChaser method rules will decide which way the ice arrow is facing and where the Chaser is relative to the ice arrow. Once the HitChaser rule has checked the ice arrow depiction, it can send a Hit message to the Chaser in front of the ice arrow point so that it either freezes or unfreezes, depending on its state.
- The first rule in HitChaser looks as follows:



- Make 3 more rules in the HitChaser method to detect the other 3 ice arrow depictions and send messages to Chasers below, left or right of the ice arrow.
- Why did we create the HitChaser method? The logic of what happens when an ice arrow is next to a Chaser is the same whether the Chaser is frozen or unfrozen so we can create a single set of rules in HitChaser that will cover all 4 ice arrow depiction possibilities. The 2 rules in the main ice arrow while-running method make sure that both frozen and unfrozen Chasers can be hit by ice arrows.

Testing

- If the agents' behavior does not match the changes you have made, click on each agent's apply button.
- Test that your Traveler can fire arrows in all 4 directions.
- Do ice arrows build up on the edges of your worksheet? Rearrange your walls to absorb them.
- If you have an arrow that does not move, check that the depiction in the New action in the Traveler's rules matches the depiction in the Move action in the ice arrow's rules.
- If you get error messages from other agents on the worksheet saying that they do not know how to respond to a "Hit" message, you must change the HitChaser rules to test that the ice arrow is about to hit a Chaser and not some other agent. Add a condition that checks for the Chaser agent by name rather than image to **each** HitChaser rule. Here is the upwards-facing

Journey (Continued)

ice arrow rule:



- If your Chaser is not frozen by a direct hit, the problem may be that the ice arrow is next to the Chaser and unfreezes it immediately after freezing it. How can you guarantee that an ice arrow will not cause the Chaser to change rapidly back to unfrozen? Add an “Erase me”



action, to each of the HitChaser rules so that the ice arrow sends a Hit message to the Chaser and instantly disappears. Here is a completed rule from HitChaser:



- Now test that your Traveler can shoot in all 4 directions and can freeze and unfreeze a Chaser with all 4 ice arrows. You may need to move the Traveler and the Chaser into position in order to test each direction.

Standards specific to the implementation of JOURNEY (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 games
- Design and develop computational science models

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

- ★ Design original games
- 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 games and simulations

Research and Information Fluency

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

Plan strategies to guide inquiry.

- Explore web sites and identify interesting connections

Journey (Continued)

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

Find relevant related web-based information, compute derivative 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 game design activities
- * Mentor other students

Demonstrate personal responsibility for lifelong learning.

- * Explore options of going beyond expected learning goals

Exhibit leadership for digital citizenship.

Journey (Continued)

- * 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 game and simulations using Photoshop (art), AgentSheets (programming), and

- * Excel (data analysis).

Troubleshoot systems and applications.

- * Debug games 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.