Creating “Journey”
In AgentCubes

You are a traveler on a journey to find a treasure. 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 by following your scent. When you collect the treasure, you win. If a chaser catches you, you lose.

Created by: Cathy Brand, University of Colorado, School of Education

This curriculum has been designed as part of the Scalable Games Design project. It was created using portions of prior work completed 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.
Lesson Objective:

- To create a game of the student’s own design where a traveler explores an unknown world.
- To be able to describe and use the computational thinking patterns described below.

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-controlled movement
    - Random movement
    - Ending the game

Computational Thinking Patterns:

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

Length of Activity:

- Three to Five 30-45 minute lessons 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 treasure
- Part 4: Switch the Traveler to First Person Navigation

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)

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 2
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 3
Polling and Broadcast

Teacher Instructions: Part 4 – Become the Traveler –
Implementing First Person Navigation
Student Handout 4: Part 4 – Become the Traveler –
Implementing First Person Navigation

Student Handout 5: Ice Arrow Challenge
Vocabulary/Definitions

Algorithm..................a set of instructions designed to perform a specific task.
Attribute...................a value assigned to an agent (such as scent)
Bird’s Eye View........ looking down on a World as if you were flying over it
Brackets....................method of setting information apart using “[“ and “]”
Broadcast.................. controller agents broadcast (or send out) messages
Chaser.....................the agent that chases the traveler
Collision....................an event wherein two agents run into each other.
Diffusion...................the process in which an attribute (in this case, scent) changes its value, being larger near its source and smaller farther way from its source
Increment..................to increase by one
Hill Climbing.............a local search technique, or algorithm, that attempts to find the best solution by testing each possible solution in turn until no better solution can be found. Here the algorithm searches for the strongest scent in the grid squares surrounding the current square.
Local Variable.............a variable (attribute) belonging to a specific agent
Method.....................a named set of rules evaluated by an agent in response to a message
Parentheses................method of setting information apart using “(“ and “)”
Polling......................the process of asking agents to update a simulation property and then taking some action based on the value of the simulation property
Propagated..................spreading a value (scent) through a grid of agents
Randomly...................to occur in non-systematic ways
Rule Order..................the order in which rules are placed for each agent
Simulation Property A named value that all agents can see and update
Traveler.....................the main character who is searching for treasure
General Teaching Strategies

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.

- Every effort has been made to create instructions with an eye toward guided discovery. Direct instruction has been used for those aspects where students are learning the code for the first time; however, materials have been provided to ensure that students are understanding the programming concepts, as opposed to simply copying code. Note that special materials have been designed for students who are new to AgentCubes.

- 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. Students may lose out on conceptual understanding with too much teacher support.

Guided Discovery Process

- **Model the process** rather than just giving students the answer. 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. Ask directing questions or give helpful suggestions, but **provide only minimal assistance** and only when needed to overcome obstacles.

- Don’t fear **group work**! 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

---

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.


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.

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

**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.

- Remember that conceptual understanding takes time, and it may be necessary to explain 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.

- 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.

- 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 the lessons 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 earlier than other students by having student 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.

  o **Lesson 1:**
    - Option 1: Pair a struggling student with an experienced student
    - Option 2: Provide struggling students with Handout 1A, which provides more directed instruction steps
    - Vocabulary for ELL Students: Traveler, Chaser, Randomly, Rule Order, Collision
    - 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. AgentCubes provides many 3D drawings that students may choose for their agents’ appearances.

  o **Lesson 2:**
    - Pairing the student with an experienced student should alleviate many problems.
    - Vocabulary for ELL Students: Algorithm, propagated, attribute, local variable, diffusion, method, parentheses, brackets

  o **Lesson 3:**
    - Note that this is a challenging lesson for students – plan extra time and additional help from more experienced students.
    - Pairing the student with an experienced student should alleviate many problems.
    - Vocabulary for ELL Students: polling, broadcast, increment

• Note that there are two versions of the student pages. The STANDARD version is designed for most students. It presumes basic knowledge of AgentCubes. The ALTERNATIVE version is designed for those students who may need more support. It provides explicit directions for the first part of the project, and step-by-step instructions for the challenges.
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 find treasure. You travel on the ground amid walls along with one or more chasers. The chasers move randomly on the ground. When you collect the treasure, you win. If a chaser catches you, 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.

(Student handout is found on page 2 of the STANDARD student packet)

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

(Student handout is found on page 2 of the ALTERNATIVE student packet for those students needing additional support)
Student Handout 1:

Part I - Basic Game

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

Create these Agents:
Choose the Inflatable Icon – People category and pick 2 images for your agents:

Traveler  Chaser

Choose a Tile to be the ground. Pick a color that contrasts with your traveler and chaser.

Choose a Cube for making walls. Pick a color that contrasts with the ground.

Choose an image for your Treasure agent from the Inflatable icon – miscellaneous category.

Create your Level 1 World:
Create the following BEHAVIORS for your agents:

**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). Make sure that the game ends if the Traveler is caught by 1 or more Chasers. Create a similar rule if your traveler approaches the treasure. What happens if the win rule is below the move rules? Does your Traveler always win? Why not?

Be SURE to reload the World when the game 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!
Student Handout 1A: No Prior Experience with AC
Part I - Basic Game

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

Create these Agents:
Choose the Inflatable Icon – People category and pick 2 images for your agents:

![Traveler](image1.png)  ![Chaser](image2.png)

Choose a Tile to be the ground. Pick a color that contrasts with your traveler and chaser.

![Tile](image3.png)

Choose a Cube for making walls. Pick a color that contrasts with the ground.

![Cube](image4.png)

Choose an image for your Treasure agent from the Inflatable icon – miscellaneous category.

![Treasure](image5.png)
### Step 1: Create Game
Click on the New Project button at the bottom of the Project Chooser Window.

Or go to the AgentCubes File menu and click on New Project.

### Step 2: Name the Game
Name it Journey and click OK.

### Step 3: Create Agent
Click on +Agent button in the lower left corner of the AC window.

### Step 4: Choose Inflatable Icon
From the Shape list on the left, then People from the center list and finally Sally or another name from the list on the right.

Click OK to save your agent.
### Step 5
Draw your own traveler by clicking on the picture of a person in the list of agents.

Use the clear button to erase the person.

Or just use the pencil tool to add details to the person.

### Step 6
Click on Pencil tool for drawing.

Click on color well to choose a color.

### Step 7
Use the + button to inflate your drawing and make it 3D.

Check the upright box to stand your Traveler up.

Pick front and back connected next to surfaces and then move the distance slider to make a wider 3D image.
## Step 8: Create Wall, Ground, Chaser and Treasure agents.

Pick a Cube for the Wall and a Tile for the Ground. Pick inflatable agents for the Chaser and the Treasure.

### Step 9: Create a world

Click the + next to World in the top bar of the AgentCubes window.

### Step 10: Name Your New World “Level 1”.

Do not change the numbers for your first world. Click OK.

### Step 11: Try out the World tools

- Select tool for moving agents
- Pencil tool for drawing agents on World
- Tile tool for drawing groups of agents
- Eraser tool
- Trigger tool that calls a user-defined method

Use the tools to design your Level 1 World.

Do not place a Ground agent over another agent. This means if you place a Chaser on the World, do not draw the Ground over it without erasing the Chaser first.

Cover the World with a layer of Ground Tiles. Put a few Walls...
DRAFT 3-D Journey (Continued)

<table>
<thead>
<tr>
<th>Step 12</th>
<th>Try out the 3D movement tools</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rotate</td>
<td><img src="image" alt="Rotate icon" /></td>
</tr>
<tr>
<td>Pan</td>
<td><img src="image" alt="Pan icon" /></td>
</tr>
<tr>
<td>Zoom</td>
<td><img src="image" alt="Zoom icon" /></td>
</tr>
</tbody>
</table>

Move your World so it looks 3D.

<table>
<thead>
<tr>
<th>Step 13</th>
<th>Use the save button to save your World when you like the way it looks!</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="World: + Level 1 Save" /></td>
<td>Save your World when it is set up to start a game. Then use the Reset button to return to the saved starting point for your game.</td>
</tr>
</tbody>
</table>

Your Level 1 World might look like this:

![Level 1 World](image)

<table>
<thead>
<tr>
<th>Step 14</th>
<th>Create behaviors for your Traveler.</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Click on the Traveler to see its behavior." /></td>
<td>We will use rules to give behaviors to our agents.</td>
</tr>
</tbody>
</table>

Rules are made up of an IF-THEN statement. In order to control the Traveler’s movements with the cursor keys, we need this rule “IF the Up key is hit, THEN the Traveler will move up.”

The Traveler needs 3 more movement rules so there is a rule for each direction (Up, Down, Left, Right).
Step 15
Create a behavior.
The Traveler’s rule 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.

Take a look at this rule…it says,

IF I click on the up arrow, THEN my traveler will move UP

Create the rules to have the traveler move up, right, left and down.

NOTE: Each rule has to be separate…use the +Rule button to create each new rule.

Step 16
Make a rule so the Traveler loses when the Chaser comes near.
The Traveler must lose the game if one or more Chasers catch her. How do we write 1 or more mathematically? Use “ >= 1”.

Use the reload World or stop simulation actions to end the game!

Step 17
Make a rule so the Traveler wins when s/he collects (erases) the Treasure.
Use the reload World action to take the World back to its saved state and let the player go right on playing.
Use the **stop simulation** action to stop the game. Then the player must click on the reload button on the top bar of the AgentCubes window to return the game to its saved state and then click on the green triangle to play again.

<table>
<thead>
<tr>
<th>Step 18</th>
<th>Prevent your Traveler from walking through walls.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Part a)</strong> Add the code shown.</td>
<td>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.</td>
</tr>
<tr>
<td><strong>Part b)</strong> Add code for the remaining directions.</td>
<td>Note an important programming point: The two conditions are in the same box...this is an AND statement. It reads like this:</td>
</tr>
<tr>
<td></td>
<td><em>IF</em> the up arrow is pressed AND the traveler does NOT see a wall above him</td>
</tr>
<tr>
<td></td>
<td><em>THEN</em> he moves up</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Step 19</th>
<th>Arrange your Traveler’s rules so that rules which do special or unusual things appear first, followed by rules which control more common behavior.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>What is the most important thing the Traveler does? Win or lose a game! Put these rules first by dragging them to the top of the Traveler’s while running method!</td>
</tr>
<tr>
<td></td>
<td>What does the Traveler do most of the time when you are playing the game? Move around the World. Drag these rules below the Win and Lose rules.</td>
</tr>
</tbody>
</table>
### Step 20

Program the *Chaser* to move randomly by putting this rule in the Chaser’s rules.

*Click on the agent to add behaviors to that agent*
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=XXXU0uKLo0

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 seek the Traveler agent using a computational thinking pattern called “hill climbing”.

Imagine that the Traveler agent emits a scent. Hill climbing is a search algorithm used by the Chaser agent 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”, which mimics the physical process of diffusion in which matter moves down a gradient from highest to lowest concentrations. In this program, the closer the ground agent is to the source of the scent (the Traveler agent), the greater the value of the scent in the ground agent.

See the Code Snippets Handout for complete rules for diffusion.

**Pass out Student Handout 2**

*(Student handout is found on page 4 of the STANDARD student packet and page 10 of the ALTERNATIVE student packet)*

---

**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 AgentCubes 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, method, parentheses, brackets
So far, your Chaser just moves randomly…he doesn’t actually chase the traveler, does he? That’s about to change!

We will make the Chaser pursue the Traveler agent using a search algorithm called “hill climbing.” An algorithm is a set of rules followed by an agent to achieve a goal. Imagine the traveler agent emits a scent. Using the hill climbing search algorithm, the Chaser finds the direction in which the scent is strongest and moves that direction, following the Traveler.

The scent will spread out or be propagated by the ground agents using a computational thinking pattern called “diffusion” which copies the physical process of diffusion by which molecules move from areas of highest concentration to areas of lowest concentration. In this game, the values spread out from the Traveler to all the ground agents in the World. The values are highest in the ground agents close to the Traveler and smallest in the ground agents far away from the Traveler.

We will introduce the concept of an “agent attribute,” which is a piece of information that is stored within each occurrence of an agent. Computer scientists call this attribute a local variable because each agent has its own copy of it and each copy has its own value.

First, let’s make sure our traveler gives off a scent. To do this, we need to set the value of an attribute named “S” which stands for Scent. The S attribute is associated with the Traveler and is set in the Traveler’s last (lowest) rule in the while running method box.

This rule says to the Traveler, “Always give yourself a scent at level 1000.” In other words, if no rule above it has a true condition, this rule’s condition will evaluate to true and the Traveler’s local variable S (Scent) will be set to 1000.

**This rule should be AFTER all the other rules for the Traveler, at the end of the list so that the Traveler wins, loses or moves before setting S to 1000 because those actions are more important to the game.**
Step 2:
Now, since the scent is diffusing, or spreading out from the Traveler, we need to find the value of the scent in each ground agent. Imagine that the smells are coming in from the North, South, East and West of each ground agent. The value of the smell in any ground agent, then, is the average of the smells in the four surrounding agents. How will you program that?

The ground agent will have the behavior below; the single action is to calculate and store the average of the values of the four surrounding agents’ S 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 values of the attributes in the agents above, below, and on each side:

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

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.

Step 3:
For the Chaser to know which way to walk, it has to determine where the scent is the strongest. If this were real life, it would smell up, smell down, smell left and smell right. Wherever the smell was strongest, it would walk in that direction. We need to program the Chaser to do this.

We will create a METHOD that enables the Chaser to do a hill climbing search so that it moves in whichever direction the scent is strongest.
Take a look at the programming below. Here’s what it says…

The rule in the while running method says:
    ONCE EVERY 0.5 seconds, send me a message to do my Hill Climb method.

The first rule in the Hill Climb method says:
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?

Add this 5th rule at the bottom of the hill climbing method box:

With this rule added, the Chaser will always make a random move if the S values in all 4 directions are the same. For example, the S values may all be equal to 0 if the Traveler’s S value has not diffused all the way across the world yet.
Note: this rule must be the last rule in the Chaser’s hill climbing method!
Diffusion and Hill Climbing

To determine what is happening in your game, it is sometimes helpful to look at the agent attributes. Reset your game, then click run followed by a click on stop. Do not reset at this point. Since your game has run briefly, your Traveler now has a scent. You can see his diffused scent (the value of $s$) by clicking on the ground agents with the big arrow tool and then going to the AgentCubes Window menu and selecting the Show Agent Attributes option. The attributes window will appear with the value of $S$ for the agent that you clicked on.

Try checking the attributes of the four ground agents around the Chaser (up, down, left and right) and then single stepping the game using the black triangle next to the stop button. Does your Chaser moves in the direction you expected him to go?

If your game isn’t working, it’s time to do some troubleshooting. Check the following:

- The Traveler’s “set $S$ to 1000” rule must be the last rule in the while running method.
- In the Chaser’s rules, the method name must be the same in the message action and the black and yellow striped method name tag on the hill climbing method!
- Use of parentheses “(“ and brackets “[“ in the Ground agent rule must be correct. Look at the picture of the Ground agent’s equation 2 pages ago and compare it to the equation in your ground agent.
- Check your hill climbing rules again and make sure that the arrows in the actions point the correct direction and that the conditions for each rule are correct.
Teacher Instructions
Part 3 – Adding the Challenge of Multiple Treasures – 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 moving onto the treasure to win, the traveler will find there are many treasures to collect. Now, in order to win, the traveler will move around and pick up all of the treasures 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 treasures to be collected)
- What stops the game?
- What steps (code) will change?
- How would the World 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 treasures left on the worksheet. Remind students that they should take time to think through each programming step so they can use these skills later.

Hand out worksheet 3

(Student handout is found on page 8 of the STANDARD student packet and page 14 of the ALTERNATIVE student packet)
Student Handout 3
Part 3:
Adding Challenge to the Game – Polling and Broadcast

We will add another challenge to our Journey Game. Now the Traveler must “collect” – that is move on top of – multiple treasures in order to win. The game does not end until all of the treasures are collected.

To accomplish this, we introduce the concept of SIMULATION PROPERTIES, which are named values that can be used and checked by all agents in a project.

We will create a new agent, the “Controller” to manage the process of polling the treasure agents to determine when they are all “collected”; that is, when there are none left on the worksheet. To begin, we must change the behavior of the traveler agent so that it no longer declares the game is over when it moves on top of the treasure.

**Step 1:** Remove the win rule from the Traveler that makes the game end when she moves above the treasure.

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

A Simulation Property name is always preceded by an “@” when it appears in an action or a condition.

The @ differentiates simulation properties from agent attributes.
**Step 2: Create a Controller agent**

Create a Controller agent, using a colorful tile or any predefined shape or drawing your own. Use the pencil tool to place one Controller on your World.

**Step 3: Counting up the treasures 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 process is similar to the way polling will work in your program.

Once per second, the Controller will say, “Treasure agent count starts at zero” (like the classroom, no hands are up when the teacher asks who is still working).

When the treasure agents ‘hear’ the Controller ask (broadcast) the question, the treasure agents respond back (raise their hands).

The controller checks the treasure agent count. If this count is more than zero, nothing happens and the game continues. If the answer is zero (meaning that there are no remaining treasures on the board), the game ends.

**How does Polling work?**

In its “While Running” method, the Controller agent first sets the simulation property “@Treasures” to zero.

Then it broadcasts a signal “Count” to all treasure agents. Each treasure agent responds by adding one to the @Treasures simulation property.

Finally, the Controller calls upon the “Check Win” method. The player wins if no treasure agents are left in the world, which is determined by the @Treasures simulation property being zero.

**Definition:** Computer scientists call the process of making a decision by sending a message to multiple recipients and checking responses polling.
Create the rule in the while running method of the Controller:

Pick a time interval for the once every condition
Add these three actions to the same rule:

1. Set the value of the simulation property Treasures to zero. (this is like the teacher saying “hands down”). **Note you must write @Treasures in the set action!**
2. Use broadcast to ask all the treasure agents to evaluate the rules in their Count methods if they are still on the World.
3. Send me (the Controller) a message to evaluate the rules in my Check Win method to see if the Traveler has collected all the treasures and won the game.

The single rule in the Controller’s **while running** method should look like this:

![Controller's while running method](image)

Click on the +Method button below the Controller’s rules. This method box will appear in the Controller:

![Method box](image)

Click on the word “Untitled” in the upper left corner and choose the same name that you entered the message action from the rule in the Controller’s **while running** method.

![Edit method](image)

Make the rule for the Check Win method:

1. Drag in the **test** action and use it to check whether the value of the simulation property Treasures equals zero because all the treasure agents have been collected by the Traveler. **Note you must write @Treasures in the test action!**
2. If the treasure agents are all gone, do the win actions. Remember to **stop** the simulation or **reload** the World to end the game!
Treasure behavior changes: There are two behavior changes required for the treasure agent.

- The first step is to have the treasure be collected by the traveler. We can simulate this by **erasing** the treasure agent when the Traveler moves on top of it.

- The second behavior change for the treasure agent is to respond to the Controller’s broadcast by evaluating the rule in its **Count** method, which updates the Treasures simulation property.

  *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 **Count**, each remaining treasure agent will increase (or **increment**) the value of the Treasures simulation property. If no treasure agents remain in the World, then the value of the Treasures property will be zero, which the controller agent will detect and declare the game won.

Here is the behavior for the Treasure Agent:

The rule in the while running method erases the Treasure agent when a Traveler is on top of it.

The action in the Count method adds one to the value of the Treasures simulation property. **Note you must write @Treasures in the set action!**
Adding Levels to Your Game:
Now that you have made your project more like a real game with Chasers that really chase the Traveler and multiple treasures that must be collected to win, it is fun to make several Worlds so that your player can try to win multiple levels.

How can you make one World more challenging than another?
1. Thank about the arrangement of the Walls. Is it easier or harder for the Traveler to escape the Chasers in a more open maze with fewer walls?
2. Think about the number of Treasures. Is it easier or harder for the Traveler to win when s/he must collect a larger number of Treasures?
3. Think about the number of Chasers. What number of Chasers would make it harder but not absolutely impossible for your Traveler to win?

How do you make your Traveler move automatically from one level to another?
- AgentCubes has a condition that checks which World the agent is in right now and an action that lets the agent switch Worlds. The example Worlds were named “Level 1” and “Level 2” when they were created.

Where would you put rules that use these actions?
- Think about when the player should switch levels: not in the middle of exploring a World but after winning a particular level.

Go to the Controller and replace the single win rule in Check Win with these 2 rules (or more if you have more than two Worlds):

Test your Worlds on your friends! How many Levels can they win?
Student Handout:
Troubleshooting Guide for Journey Part III
Polling and Broadcast

Make a quick check on how many Treasures are in the World:
Go to the AgentCubes Windows menu and select “Show simulation Properties”. This window will appear:

![Simulation Properties Window]

The correct number of Treasures will not appear in this window until you have single-stepped (click on the black triangle next to the stop and go buttons) or briefly run the game. If your programming is correct, the value of Treasures will decrease by 1 each time your Traveler collects (erases) a Treasure. Stop the game and click on the word “Treasures” to check the current value of the Treasures simulation property. When the value of Treasures is equal to 0, you should win the game.

More detailed troubleshooting: To determine what is happening in your game, it is helpful to look at how the simulation property changes over time. Add the plot to window action to the rule in the Controller’s while running method. Fill it out as it appears below:

![Behavior: Controller]

In the plot to window action, you must name the simulation property to be plotted (Treasures), name the window where it will appear (Treasure Plot), say what it represents (number of Treasure Agents) and pick the color of the line that will appear on the graph. Note that you must put “@” before the Treasures in the plot to window action!
Look under the AgentCubes Windows menu and select the Treasure Plot window. Move the Treasure Plot window somewhere where you can watch it while you run the game. In this window, you will see 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 number of treasures never goes above 0, there is a problem with the method Count or the broadcast. If the number of treasures goes to zero but the game doesn’t end, there is a problem with the game ending rule in 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 AgentCubes.
   b. Examples are erasing the treasure agents.

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 treasures and winning the game.

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

1) **Diffusion**: emitting the scent (smell) of an agent. We used an agent attribute (S) on the agent with the smell, and we diffuse the smell by diffusing the attribute using the average of the 4 smells around it; $s = (s[left] + s[right] + s[up] + s[down]) \times 0.25$.

2) **Hill Climbing**: following the strongest 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 move towards the highest value of the scent “s” around him.

3) **Polling**: is when an agent “shouts out” (broadcasts) to all agents of a certain type requesting them to execute a specific method in response and perhaps change a global variable so that the originator agent can make a decision and take action. Example is the broadcast by the Controller of the method “Count” to the treasure agents in order to discover whether the game should end.

C) Other concepts we covered in AgentCubes 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.
4) Agents communicating through messages.
5) Switching Worlds in the win rules to implement multiple levels.
Ice Arrows Challenge

Before your 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 chase the Traveler using a hill climbing search. 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 must shoot ice arrows up in all four directions (up, down, left and right).
- 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

- Make it be an inflatable icon so you can draw your own picture.
- The picture for the ice arrow agent may face in any of the four directions.
- The point of the arrow should be a different color from the tail so that you can easily recognize which arrow you are seeing in the tiny pictures in the conditions.
- After you have drawn the first arrow shape, select the ice arrow agent and click on the +Shape button at the lower left corner of the AgentCubes window so you can draw an additional shape for the basic ice arrow.
- Draw 4 ice arrow shapes that so that the shapes face upwards, downwards, left and right.
- The ice arrow’s shape stores its Direction. We can tell which way an ice arrow should move by checking its shape. The image saves the direction instead of an agent attribute.

Create a second shape for the Chaser: a frozen Chaser

- Select your chaser agent and click on the +Shape to create the frozen Chaser
- Make sure the frozen Chaser looks different enough that you can see it in the tiny pictures in the conditions.
- The Chaser’s picture stores its state: frozen or unfrozen.
Traveler Design Challenges
How do you know which ice arrow the Traveler should shoot?
The Traveler should shoot an ice arrow in the direction that s/he is facing.

How do you know which way the Traveler is facing?
The Traveler must have an Agent Attribute (or local variable) called Direction which keeps track of which way the Traveler is facing.
Initialize the Direction Agent Attribute in a when-creating-new-agent method.
When an arrow key is typed, set the Direction attribute and rotate the Traveler to face in that direction of the arrow key before the Traveler moves.

How can you avoid putting 4 rules for the 4 different ice arrows in the Traveler’s while running method?
When the space bar is typed, make the Traveler send itself a message to do a method called Shoot Arrow that will shoot an arrow in the direction the Traveler is facing.

How are Directions named in AgentCubes?
Directions must be named using degrees on a circle as in the picture below.

How do you know which ice arrow the Traveler should shoot?
The Traveler should shoot an ice arrow in the direction that s/he is facing.

Traveler Rules
How to Create and Initialize the Traveler’s Direction variable
1. Click on the +Method button
2. Click on the word on in the method’s black and yellow tag and change it to “when-creating-new-method”
3. Drag in the set action and make a new variable name, Direction.
4. Set Direction to 0.
5. Drag in the rotate action and set all 3 numbers to 0.

Up arrow key = move in 0 degrees direction
Left arrow key = move in 90 degrees direction
Down arrow key = move in 180 degrees direction
Right arrow key = move in 270 degrees direction
6. Erase the Traveler and redraw the Traveler on the World, then save it. The Traveler’s when-creating-new-agent method should look like this:

```
when creating new agent

if
then
  set Direction to 0
  rotate to 0 0 0

DRAFT 3-D Journey (Continued)

Test the value of the Traveler’s Direction attribute

1. Choose Show Agent Attributes from the AgentCubes Window Menu
2. Click on the Traveler with the big Arrow tool.
3. You should see this:

![Attributes: Traveler_Agent](image)

Update the Traveler’s move rules

Add the `set` and `rotate to` actions to the Traveler’s move rules so that the Traveler’s Direction attribute changes as the player types the different arrow keys. Here is a the move right rule. **Edit the other 3 move rules.**

```
if
then
  set Direction to 270
  rotate to 270 0 0
  move

Note: the number for the direction goes in the first box of the rotate to action!

Test the Traveler’s move rules to make sure the value of the Direction attribute matches the Traveler’s movements

1. Choose Show Agent Attributes from the AgentCubes Window Menu
2. Click on the Traveler with the big Arrow tool.
3. Type the right arrow key. Does Direction change to 270 when the Traveler moves right? Does the Traveler rotate to face right?
4. Check the other arrow keys. **Remember move up is 0, move left is 90 and move down is 180.**

**Making the Traveler Shoot Arrows**

Here is the rule from the Traveler’s while running method that generates ice arrows:

```plaintext
if key = space
then message Shoot Arrow
```

Where should this rule appear in the while running method box?

- Above or below the win rule?
- Above or below the move rules?

**Remember: special cases and less common events appear above default behavior like moving!**

Here is the first rule in the Traveler’s Shoot Arrow method:

```plaintext
if Direction = 0
then new 
```

**Direction** refers to the Traveler’s agent attribute Direction, which keeps track of which way the Traveler is facing. **Remember 0 is up, 90 is left, 180 is down and 270 is right.**

Make 3 more rules like this one for the remaining 3 directions.

**Test your code:**

- Does your Traveler generate ice arrows in all 4 directions?
  
  If not, make sure that your ice arrow pictures and the arrows in the new actions match the directions.

**Ice Arrow Rules**

**Ice Arrow Design Challenges**

**How do you know which direction the ice arrow should move?**

Ice Arrows should move whichever direction they are pointing.

**How can you avoid putting 4 rules for the 4 different ice arrows in the Ice Arrow’s while running method?**

Make the Ice Arrow send itself a message to do a method called Fly that will make an ice arrow move in the direction it points.
Here is the rule that belongs in the Ice Arrow’s while running method:

```java
while running
    if once every .01 sec
    then message • Fly
```

Here is one of the 4 rules from the Ice Arrow’s Fly method:

```java
on Fly
    if see • 1
    then move ↑
```

Add 3 more rules for the other 3 directions to the Fly method.

**Test your code:**
- Do your ice arrows move in all 4 directions?
  - If not, make sure that your ice arrow picture and the move arrow match in each rule.

**How do you make the point of the ice arrow hit the Chaser?**
Only the point of the arrow can freeze or unfreeze the Chaser. It does not look convincing if the Chaser is hit by the side of an arrow. We need another method called HitChaser to see whether the arrow point will actually hit the Chaser.

**What rules call HitChaser?**
Use the `next to` condition to test whether the arrow is near a **frozen** or **unfrozen** Chaser, then use the HitChaser method to find out whether the Chaser is in front of the arrow point.

Here is the rule that calls HitChaser when the ice arrow is next to an **unfrozen** Chaser:

```java
while running
    if next to > 1
    then message • HitChaser
```

Does this rule belong above or below the rule that makes the ice arrow do it’s Fly method?
- It’s a special case and special cases should always be higher than other rules!
- **Make a second rule that will call HitChaser when the ice arrow is next to a **frozen** Chaser.**
What do Ice Arrow’s HitChaser rules do?

1. Test the shape of the arrow.
2. Test whether there is a Chaser on the grid square that the arrow points at.
3. Send a message to the Chaser that it has been Hit.

Here is a rule from the HitChaser method:

Make 3 more rules like this for the other 3 ice arrow shapes. Then make the same 4 rules for the Frozen Chaser, so that it unfreezes if hit by an arrow.

Chaser Rules

How does the Chaser Freeze and Unfreeze?
The Hit method freezes an unfrozen Chaser and unfreezes a frozen Chaser.

Here is the rule from the Chaser’s Hit method that freezes a Chaser:

Note: The blue Chaser is frozen.

Make a second rule that will unfreeze a frozen Chaser.
Test your code:

- What happens when the Traveler shoots an ice arrow at the Chaser? You may need to temporarily delete the action in the Chaser’s rules that makes it hill climb in search of the Traveler. It is much easier to test your ice arrows if the Chaser holds still!
- Does the ice arrow sit in front of the Traveler while the Traveler blinks back and forth between frozen and unfrozen shapes?
- The problem here is that the ice arrow keeps sending Hit messages to the Chaser so that the Chaser freezes and unfreezes over and over.
- How do you make the ice arrow stop sending messages to the Chaser?
- Erase it!
- Change the rules in the Ice Arrow’s HitChaser method so that the ice arrow erases itself after it sends the Hit message to the Chaser.

Here is what the rules in HitChaser should look like now:

Test your code: does the Chaser stay frozen now until a 2nd ice arrow hits it?

One last step:

Do your ice arrows jump over walls?
Add rules to the Ice Arrow’s Fly method so that ice arrows erase themselves if their points run into a wall.

Here is one of the rules:

Where should the rules about walls appear in the Ice Arrow’s Fly method?

Special cases come before default behaviors!
1. Running into a wall happens less often so it is a special case and belongs above the regular fly rules.
2. The Ice Arrow’s default behavior is to move forward so the call to the Fly method should be the last rule.

Where does this rule go? Look →

Make 3 more rules like this for the other 3 ice arrow shapes.
Test your program!

- Do your ice arrows stack up on the edges of your world?
- Put walls around the edges to absorb the ice arrows.
- Or create special ground agents that absorb ice arrows and put them around the edges of your world. If you choose this option, you will need 4 more rules in the Ice Arrow Fly method that look just like the wall rules but use the special ground agent instead of the wall. Put these rules just below the wall rules.
- Test that your Traveler is able to fire ice arrows in all 4 directions.
- The ice arrow should shoot in the Direction the Traveler is facing. If necessary, select the Traveler with the big arrow tool, then click on the Show Agent Attributes option in the AgentCubes Window menu to check which way the Traveler is facing and make sure that the ice arrow is fired the same direction. If there is a problem, go back through this tutorial and check that your rules exactly match the pictures.
ISTA Standards² 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

² ISTE Standards for Students (ISTA 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
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 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.

- 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), AgentCubes (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.