Creating “PacMan”

Create the quintessential arcade game of the 80’s!
Wind your way through a maze while eating pellets.
Watch out for the ghosts!

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.

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:

- The students will apply the computational thinking patterns of user control, absorb, collaborative diffusion and Hill Climbing while creating a version of the 80’s arcade game, PacMan.

Prerequisite Skills:

- Students are presumed to have the following skills.
  - Create agents
  - Basic agent behavior including:
    - Key control
    - Random movement
    - Ending the game

Computational Thinking Patterns:

- User Control
- Collision
- Collaborative 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 PacMan and randomly moving ghosts
- Part 2: Make the Ghosts chase the PacMan
- Part 3: Enhance the game so that the game is over when the pellets are gone
# Table of Contents

<table>
<thead>
<tr>
<th>Teacher Instructions:</th>
<th>General Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Teacher Instructions:</td>
<td>Part 1 – Basic Game</td>
</tr>
<tr>
<td>Student Handout 1:</td>
<td>Part I - Basic Game (experienced student)</td>
</tr>
<tr>
<td>Student Handout 1A:</td>
<td>Part I - Basic Game (new student)</td>
</tr>
<tr>
<td>Student Handout 1B:</td>
<td>Agent Creation Short Cuts</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Teacher Instructions:</th>
<th>Part 2 – Making the Ghost Chase the PacMan</th>
</tr>
</thead>
<tbody>
<tr>
<td>Student Handout 2:</td>
<td>Part 2 – Making the Ghost Chase the PacMan</td>
</tr>
<tr>
<td>Student Handout 2A:</td>
<td>Troubleshooting Guide for PacMan Part II</td>
</tr>
<tr>
<td></td>
<td>Diffusion and Hill Climbing</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Teacher Instructions:</th>
<th>Part 3 – Making the game more sophisticated – Polling and Broadcast</th>
</tr>
</thead>
<tbody>
<tr>
<td>Student Handout 3:</td>
<td>Part 3 – Making the game sophisticated – Polling and Broadcast</td>
</tr>
<tr>
<td>Student Handout 3A:</td>
<td>Troubleshooting Guide for PacMan Part II Polling and Broadcast</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Student Handout 4a:</th>
<th>Challenge: PacMan changes directions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Student Handout 4b:</td>
<td>Challenge: PacMan moves continuously</td>
</tr>
<tr>
<td>Student Handout 4c:</td>
<td>Challenge: Create Power Pellets</td>
</tr>
<tr>
<td>Student Handout 4d:</td>
<td>Challenge: Move to next level</td>
</tr>
</tbody>
</table>
Algorithm ............. a set of instructions designed to perform a specific task.

Attribute ............. a variable held by an agent (such as scent) that is sometimes referred to as a Local Variable

Brackets ............... method of setting information apart using “[“ and “]”

Broadcast ............. controllers broadcast (or send out) a signal

Ghost ................. the agent that chases the PacMan

Collision .............. an event wherein two agents run into each other.

Diffusion .............. the process in which an attribute’s values (in this game, scent) change base on the location of the source

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

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

Method ................. a set of rules to follow in a specific situation

Parentheses ........... method of setting information apart using “(“ and “)”

Polling ................. the process of contacting and communicating with each agent

Propagate ............. the spreading of the scent

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

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

PacMan ............... the main character who eats the pellets as the user moves him around the worksheet.
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 transfer of programming concepts between related games and simulations.

- 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 for each curriculum guide, special materials have been designed for students who are new to this program.

- 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. Build the game on your own, before trying it with your students to enable you to see possible struggling points.

- Have students work through problems on their own. Ask guiding 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

---

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.


other programmers. Talking through coding problems enables students to think more critically about Computational Thinking Patterns, as well as the steps needed to solve a problem. Additionally, seeing how others solved an issue with code helps students realize that problems often have multiple solution strategies, and some that 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 or programming techniques that are relatively new necessarily include more student 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 some concepts multiple times, using different examples in different situations, 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 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 earlier than other students by having student materials ready in advance.

• **Students who need more assistance:** Other students (especially those with no prior programming experience with AgentSheets) 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 approach 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 and student support.
    - Vocabulary for ELL Students: PacMan, Ghost, 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. Handout 1B provides block images of each agent as portrayed in this lesson.

  o **Lesson 2:**
    - 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

  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: Two student packets are available. The STANDARD packet is for students with some basic knowledge of AgentSheets. The ALTERNATIVE packet is for students with NO prior AgentSheets experience who may need more support. The ALTERNATIVE packet should NOT be used for most students as it significantly reduces the active thinking processes.
Teacher Instructions:

Part 1 – Basic Game

Teaching Suggestions

Task your students with creating a new game. The basic features\(^2\) of the game are as follows:

The player controls Pac-Man through a maze, eating pellets. When all pellets are eaten, PacMan is taken to the next stage. Enemy ghosts roam the maze, trying to catch PacMan. PacMan dies if an enemy touches Pac-Man.

Give students time to discuss the features of this game.

Consider these prompts:

- How is this game similar to Frogger? Different than Frogger?
- What are the skills that you learned from Frogger that you might need here?
- What agents will we need?
- How do you think you might be able to program the Ghost 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.

\(^2\) From http://en.eikipedia.org/wiki/Pac-Man

The student pages can be found on page 3 of the STANDARD packet and page 3 of the ALTERNATIVE packet.
Initial Story: Create the quintessential arcade game of the 80’s! Wind your way through a maze while eating pellets. Watch out for the ghosts!

Create these Agents and the worksheet:

- PacMan
- Ghost with two depictions
- pellets
- Background
- Wall
Create the following BEHAVIORS for your agents:

**Step 1: Ghost:**
Program the Ghost to move randomly on the floor.

![Image of Behavior Ghost](image)

**Step 2: PacMan:**
- Set up your agent to move with the arrows (cursor control).

**Step 3: Prevent your PacMan from going through walls**
Work with the person next to you to figure out how to prevent the PacMan from walking into a wall. Here is the code in words:

  IF the Up Arrow is pressed AND I See the FLOOR in the up direction, THEN, I MOVE UP

**Step 4: Enable your PacMan to ‘eat’ the pellets**
Work with the person next to you to figure out how to have the PacMan eat the pellets.

---

**Collision**

*The event that occurs when two agents are next to one another*

**IF the PacMan and the Pellet COLLIDE, THEN the Pellet should ERASE**

**If the PacMan and the Ghost COLLIDE, THEN the Game is OVER**
Step 5: Game Over when the PacMan is next to a Ghost
**Remember to RESET the Simulation when the Game is over.**

Step 5: Test your game
Play your game by pressing the green arrow.
- Does your PacMan move in all four directions?
- Does your PacMan stay on the floor (and not go through walls)
- Do the ghosts move randomly?
- Does your PacMan eat pellets?
- Does the game end when the PacMan is next to the Ghost?
Student Handout 1A:

Part I - Basic Game

Initial Story: Create the quintessential arcade game of the 80’s! Wind your way through a maze while eating pellets. Watch out for the ghosts!

Create these Agents and the worksheet:
You are about to create your first game. Because this is new to you, we will give you some parts of the code. Be careful – you will have to find the patterns to create the rest of the code. If you get stuck, check in with a nearby student to get some help!

Launch AgentSheets and Create a New Game

<table>
<thead>
<tr>
<th>Step 1</th>
<th>Create Game</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Click on the new game icon (far left)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Step 2</th>
<th>Name the Game</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Name it PacMan and click OK</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Step 3</th>
<th>Define Agent Size</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Do not change - Click OK</td>
</tr>
</tbody>
</table>
PacMan (Continued)

Launch AgentSheets and Create a New Game

<table>
<thead>
<tr>
<th>Step 4</th>
<th>Create Agent</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Click on New Agent</td>
</tr>
<tr>
<td></td>
<td>Name it PacMan</td>
</tr>
<tr>
<td></td>
<td>Click OK</td>
</tr>
</tbody>
</table>

**Step 4 Diagram:**

- Create Agent
- Click on New Agent
- Name it PacMan
- Click OK

<table>
<thead>
<tr>
<th>Step 5</th>
<th>Edit Agent</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Click EDIT DEPICTION</td>
</tr>
<tr>
<td></td>
<td>Click CLEAR to erase the current image.</td>
</tr>
</tbody>
</table>

**Step 5 Diagram:**

- Click EDIT DEPICTION
- Click CLEAR to erase the current image

<table>
<thead>
<tr>
<th>Step 6</th>
<th>Draw PacMan</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Click DONE</td>
</tr>
</tbody>
</table>

**Step 6 Diagram:**

- Draw PacMan
- Click DONE

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.
### Step 7
**Draw remaining agents**

- **Ghost with two depictions**
  - Create a ghost agent and design as the green ghost. Then click New Depiction and create the red ghost.

- **Pellet**

- **Background**

- **Wall**

### Step 8
**Make the workspace**
- Click File>>New Worksheet

### Step 9
**Make the worksheet bigger**
- Notice it is big, but not so big that it fills up the whole space.
Step 10 | Use the tools to place items on the worksheet.

Pencil: places items one at a time

Filled in Rectangle: Places items in an array (rectangle).

Step 11 | SAVE your WORKSHEET!!

It is important that you do not draw over an agent with the Ground agent. This means if you place a pellet on the worksheet, do not draw the Ground over it without erasing the pellet first.
## Time to Create Behaviors for your Agents

<table>
<thead>
<tr>
<th>Step</th>
<th>Create behaviors for your agents</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>Read the explanation and then Click Edit Behavior</td>
</tr>
</tbody>
</table>

The behaviors that we will give to our agents are called rules. Rules are made up of one or more IF-THEN statements. To control the PacMan using the cursor keys, one of the rules we need should be that “IF the Up key is hit, THEN the PacMan will move up.” We also want to add a condition that says the PacMan only walks on the floor. Overall we should have 4 rules, one for each direction (Up, Down, Left, Right).

![Diagram of rule creation](image)

**Take a look at this rule…it says,**

**IF the user presses the up arrow, AND I See the FLOOR in the up direction,**

**THEN my PacMan will move UP**

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

**NOTE:** Each rule has to be separate…use NEW RULE to create each new rule.
Step 13 | Program the PacMan to eat the pellets.  
---|---
First, take a moment to think about the PacMan and the Pellets. Does the PacMan really eat the Pellets? No, he’s just a picture and can’t eat anything. Instead, it’s an illusion. When the PacMan is next to the Pellets, then the Pellet disappears, making it look like the PacMan ate it.

Take a look at this code when the PacMan sees the Pellet above him:

![Behavior: Pacman]

It says…

**IF**

The Up Arrow is pressed

AND

I see a Pellet above me

**THEN**... AND

Erase the Pellet above me

AND

Move up

**NOW**… Add the 3 rules for the other directions

---

Step 14 | Program the **Ghost** to move randomly
---|---
Should the ghost also move on the pellets? Or only on the floor? You decide and code accordingly!

Click on the agent to add behaviors to that agent

---

Step 15 | Create rule to end the game when the PacMan is next to the Ghost
---|---
Click on PacMan and Edit Behavior

Add these rules

This rule says:

**IF** I am NEXT TO one or more GHOSTS

**THEN**, give a message “Game Over!” and Reset the Simulation

Don’t forget the last action – reset simulation!!!
Student Handout 1B: 
Agent Creation short-cuts
Teacher Instructions:

Part 2 – Making the Ghost Chase the PacMan

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

There are two CTPs that will be included in this lesson:

Diffusion: The PacMan leaves a scent on the floor and the pellets. The scent will spread along the floor and pellets…this is the same as saying the floor and pellets will DIFFUSE the scent.

Hill Climbing: This is a particular way of ‘SEEKING’ a scent. This code makes the Ghost check the scent in all four directions, and follow the strongest scent.

This process all includes AGENT ATTRIBUTES

An Agent Attribute is a VARIABLE that can be used LOCALLY (by an individual agent). Note that AgentSheets also has Simulation Properties, or variables that can be used GLOBALLY (seen by ALL agents).

Instruction:
Talk to your students about the Ghost.

Consider these prompts:

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

[Give students a minute or two to discuss this with the person next to them. Then solicit their ideas. Remember that allowing students to discuss new concepts will aid their conceptual understanding of those concepts.]

[Say to your students] Imagine the PacMan emits a scent that the Ghost could smell…would that make it easier for the Ghost to find him? [Give the students an example they can relate to…bacon cooking in the kitchen, the smell of fresh coffee, sharks seeking blood in an ocean, 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.

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

Imagine the PacMan agent emits a scent. 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 value. Hill climbing, which is an algorithm to find the direction in which the scent is strongest, is then used by the ghosts to chase the PacMan.

Consider using the Diffusion PowerPoint Presentation to support understanding of these concepts. Students have a worksheet that accompanies this lesson. The following screen shots of code are for the teacher, but this information should not be shared fully with the student unless needed.

PacMan is the source of the scent wherever he moves. (There are no conditions for this rule, and it should be the last rule for PacMan)

3 Available on the Scalable Game Design Wiki. Use Quick Links>>Teacher>>Lesson Plans
Both the Pellet and the Background diffuse the scent so that the Ghosts can identify the direction of the strongest scent:

Set \( s \) to \( 0.25 \times (s_{\text{left}} + s_{\text{right}} + s_{\text{up}} + s_{\text{down}}) \)

Every 0.3 seconds, the Ghost will follow the scent by determining which direction is strongest, up, down, left or right. (Note, only the first of four rules is shown below).
Shortcut for Hill Climbing
AgentSheets has an even better way to handle the process of Hill Climbing which should be shared with students AFTER they understand the process of Hill Climbing. **It is worth the process of programming it entirely before using the shortcut to ensure that students.** The use of this code makes future extensions in the game (such as having the ghosts run from the PacMan easier).

This code is a shortcut for checking in all four directions (defined as Von Neumann’s Neighborhood…students could choose to check in eight directions using Moore’s Neighborhood!

You could then **extend** the learning by asking students to compare what happens using the two different programming methods.

http://www.countrylife.com/wiki/Von_Neumann_neighbourhood
There are many resources available on the Scalable Game Design Wiki to support the computational thinking patterns of Diffusion and Hill Climbing. Take time before the lesson to check these out and determine how they might support your students in better understanding these concepts.

http://sgd.cs.colorado.edu/wiki/Collaborative_Diffusion

Pass out Student Handout 2
The student pages can be found on page 7 of the STANDARD packet and page 11 of the ALTERNATIVE packet.

Troubleshooting:

Students can also add the MAP action to change the floor color and visibly see diffusion happen.
So far, your Ghost just moves randomly, either just on the floor, or on the floor and the pellets...he doesn’t actually chase the PacMan, does he? That’s about to change!

The Ghost will intelligently seek the PacMan agent using a computational thinking pattern called “seeking.” In this instance, we will use a specific method of seeking called Hill Climbing. Imagine the PacMan 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 process (physical, biological, and social) by which objects move from areas of highest concentration to areas of lowest concentrations. The closer to the source of the scent, the greater its value.

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 PacMan 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 PacMan. There are several ways to do this. For example, we can create this set at the end of the PacMan list of rules:

![Behavior: Pacman](image)

This rule says to the PacMan, “If I am not doing anything else, I will emit a scent at level 1000 wherever I am.” This rule should be AFTER all the other rules for the PacMan, at the end of the list.

*Looking to challenge yourself a bit more – here are some ideas to ponder…*
We may want to use a different means of setting this attribute to 1000. One option is to set the value of “s” only when we place the PacMan on the worksheet. The advantage to this will appear later when we use the PacMan as a controller. To do this, we would use a “when creating” trigger for this rule.
Other potential solutions:
- use the SET action and save the workshop to set the scent to a value
- use a onceEvery (1.0) timer condition.
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?

Diffuse the scent with the pellets

The pellet 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:

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

OR

\[
S = (s_{\text{up}} + s_{\text{down}} + s_{\text{right}} + s_{\text{left}})/4
\]

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

NOW…diffuse the scent across the ground!

Match both the parentheses “(” and the brackets “[” as shown in the equation.
Step 3:

For the Ghost to know which way to walk, he has to determine where the scent is the strongest. We call this HILL CLIMBING. 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 Ghost to do this.

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

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 Ghost 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 Ghost chases the PacMan! 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
Troubleshooting Guide for PacMan 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 until the ghosts move out of the box, and then click stop. Do not reset at this point. Your PacMan has now emitted his scent. You can see his scent (the value of s) by clicking anywhere on the ground and then on clicking 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 Ghost. By checking the attributes of the four boxes around the Ghost (up, down, left and right) and then running the game again, you can see if your Ghost 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 more sophisticated – 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:

In our game, the game ends when PacMan is next to a ghost. Now we need to find a way to win the game when all of the pellets are gone.

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 pellets to be collected)
• What stops the game?
• What steps (code) will 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 pellets 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
No code is provided in the teacher instructions as all code is provided for students.
The student pages can be found on page 11 of the STANDARD packet and page 15 of the ALTERNATIVE packet.
Student Handout 3
Part 3:
Making the game more sophisticated – Polling and Broadcast

In this enhancement to the PacMan project, the PacMan must “eat” all of the pellets in order to win. The game does not end until all of the pellets are gone.

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.

You can choose to use the PacMan as the “Controller” or make a separate Controller to manage the process of polling the pellets to determine when they are all “eaten”; that is, when there are none left on the worksheet.

Step 1: Counting up the pellets 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, “Pellet count starts at zero” (like the classroom, no hands are up when the teacher asks who is still working). When the pellets ‘hear’ the Controller ask (broadcast) the question, the pellets respond back (raise their
PacMan (Continued)

hands). The controller counts the pellets. If the answer is more than zero, nothing happens and the game continues. If the answer is zero (meaning that there are no remaining pellets on the board), the game ends.

First, we need to create a simulation property called ‘pellets’. This property is the count of the hands. To do this, go to Tools>>Simulation Properties>>New. Type in pellets 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 pellets, we type @pellets.
There are three parts to the Controller behavior.

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

Set @pellets to zero

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

Broadcast Pellets check_in

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

Make myself check-won

How do Simulation Properties Work?
In the “While Running” method, the control first sets the simulation property “@pellets” to zero. Then it broadcasts a signal to all pellets. 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 pellets remaining, which is determined by the @pellets simulation property being zero. If any pellets are left, we will see that this simulation property will be greater than zero.
**Pellet behavior changes:** The controller has told the pellets to “check_in” but they don’t know how to do that. We have to program them to check in when asked.

- The rule will tell the pellet agent to respond to the “poll” (broadcast) called *check_in* from the Controller, to update the @pellets simulation property.

  *This 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 pellet agent will increase (or *increment*) the @pellets simulation property. If no pellet agents remain, then the @pellets property will be zero, which the controller agent will detect and declare the game won.
Student Handout:

Troubleshooting Guide for PacMan Part III

Polling and Broadcast

Common Problems:

1. Is your Controller agent on the worksheet?
2. Did you type in \texttt{@pellets} 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 ‘pellets’. Change the plot to graph between 0 and the total number of pellets 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 pellets never goes above 0, there is a problem with the method ‘check\_in’ or the broadcast. If the pellets 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 - PacMan

A) The main computational thinking patterns we reviewed were:
   1) **User Control**: intentionally moving an agent.
      a. Using keyboard keys to move an agent.
      b. Example is moving the PacMan.
   2) **Absorb**: deleting agents on the screen.
      a. Use the “Erase” action in AgentSheets.
      b. Examples are erasing the pellets.
   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 eating pellets and losing the game when the ghosts touch the PacMan.

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,

      \[ s = \frac{(s[left]+s[right]+s[up]+s[down])}{4}. \]

   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 Ghost 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 AgentSheets.
      b. Example is the broadcast to the Controller - the method check_in” to check in with the pellets to see if they are still there.

C) Other concepts we covered in AgentSheets 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:

PacMan Changes Direction Challenge

Before your start this challenge:

You must have a complete basic PacMan game with a PacMan who wins if s/he eats all the pellets and Ghosts who either move randomly or chase the PacMan. The PacMan loses if a Ghost gets too close. The worksheet should have walls that the Ghost and PacMan cannot cross.

Description of the Challenge:

• PacMan will turn in the direction he’s heading.

What to consider:

Do you need a new agent?
Do you need a new rule?

You might be thinking you need new agents…BUT WAIT! Since each PacMan will follow the same rules, you don’t need a new agent, but rather a new DEPICTION…

Right click on the depiction of the original agent. Check out the options under “duplicate depiction” and “run duplication script” to figure out how to create three other directions of the PacMan. You may want to rename them to help you stay organized. When you have done it correctly, it will look like this ➔.

Once you have the different depictions, you will want to change your code so that WHEN THE PACMAN MOVES, it CHANGES into the new DEPICTION.

TEST your program to confirm that the depiction changes when the PacMan changes directions.
Student Handout 4b:

**PacMan Moves Continuously**

**Challenge**

Before your start this challenge:

You must have a complete basic PacMan game with a PacMan who wins if s/he eats all the pellets and Ghosts who either move randomly or chase the PacMan. The PacMan loses if a Ghost gets too close. The worksheet should have walls that the Ghost and PacMan cannot cross.

You must have different depictions of the PacMan so that he faces the direction he heads.

**Description of the Challenge:**

- PacMan will continuously move in the direction he’s heading.

This challenge gets you started, but won’t give you all the code. Review the code below: It says, when the up arrow is pressed, change to the up depiction. Once every 0.2 second, make me (the PacMan) MoveDirection.

When the MoveDirection method is called, the PacMan does the following:

If I see myself heading up AND I see the floor in the up direction, I will move up.
There is still much to code:

Step 1:
Add code to tell the PacMan what to do when he sees an up depiction and a pellet in the up direction.

Step 2:
Create code for all the other directions.

Step 3:
Test your program. (Hint: be sure your PacMan still leaves his scent everywhere.)
Click on the PacMan and run the program. Use the colors to decide which rules are true or false. In this case, the first rule is red, which means the Up arrow was not pressed.

The next rule is green, which means every 0.2 seconds, the PacMan is being told to MoveDirection.

The method MoveDirection is red, which means that either/both are true. The PacMan does not see the Up Depiction AND/OR does not see the floor, making the rule FALSE.
Power Pellet Challenge

Before your start this challenge:

You must have a complete basic PacMan game with a PacMan who wins if s/he eats all the pellets and Ghosts who either move randomly or chase the PacMan. The PacMan loses if a Ghost gets too close. The worksheet should have walls that the Ghost and PacMan cannot cross.

You must have different depictions of the PacMan so that he faces the direction he heads, and he must move continuously.

Description of the Challenge:

- Power Pellets are added to the worksheet
- Power Pellets provide Pac-Man with the temporary ability to eat the enemies. The enemies turn deep blue, and reverse direction

This challenge gets you started, but won’t give you all the code.

To help you think this through…

- You will need a new agent (Power Pellet)
- Do you need a new agent for the blue ghost?
- When the ghost chases the PacMan, he has a scent of 1000. What happens if he has a scent of -1000? How can you set that new scent? How can you time-limit that value?
- Hint: Use the hill climbing action rather than all the code for sniffing.
Student Handout 4d:

Next Level Challenge

Before your start this challenge:

You must have a complete basic PacMan game with a PacMan who wins if s/he eats all the pellets and Ghosts who either move randomly or chase the PacMan. The PacMan loses if a Ghost gets too close. The worksheet should have walls that the Ghost and PacMan cannot cross.

You must have different depictions of the PacMan so that he faces the direction he heads, and he must move continuously.

Description of the Challenge:

- When the game ends, a new level appears, even harder than before!

This challenge gets you started, but won’t give you all the code.

To help you think this through…

- Do you need a new agent? A new worksheet?
- When would a new level appear?
- What code needs to change to make the new level appear?

You might have code like this:

How could you incorporate this action instead?

HINT: You will need to BROWSE to name the new level properly. Click in the box, do not just type in the name.
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 complex 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

4 ISTE Standards for Students (ISTE Standards•S) are the “standards for evaluating the skills and knowledge students need to learn effectively and live productively in an increasingly global and digital world.” http://www.iste.org/standards/standards-for-students
Students apply digital tools to gather, evaluate, and use information. Students:

**Plan strategies to guide inquiry.**
- Explore web sites and identify interesting connections

**Locate, organize, analyze, evaluate, synthesize, and ethically use information from a variety of sources and media.**
- Find relevant related web-based information, compute derivate information

**Evaluate and select information sources and digital tools based on the appropriateness to specific tasks.**
- Understand validity of information, e.g. Scientific journal information vs. Personal blogs

**Process data and report results.**
- Write programs to access numerical information, define functions to process data and create output based on voice or plotting to represent data.

**Critical Thinking, Problem Solving, and Decision Making**

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

- **Identify and define authentic problems and significant questions for investigation.**
  - Define research questions and explore approach of exploration

- **Plan and manage activities to develop a solution or complete a project.**
  - Outline sequence of exploratory steps
  - Experience complete bottom-up and top-down design processes
  - Employ algorithmic thinking for creating programs to solve problems

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

- **Use multiple processes and diverse perspectives to explore alternative solutions.**
  - Experience and understand design trade-offs, e.g. Bottom-up vs. Top-down

**Digital Citizenship**

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

- **Advocate and practice safe, legal, and responsible use of information and technology.**
  - Learn how to use tools to locate resources, e.g., images with google image search, but understand copyright issues

- **Exhibit a positive attitude toward using technology that supports collaboration, learning, and productivity.**
  - Stay in the flow, where design challenges match design skills
  - Experience success through scaffolded 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), 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.