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: Cathy Brand, 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 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:

- 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

**Teacher Instructions:** General Information

**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 Ghost Chase the PacMan

**Student Handout 2:** Part 2 – Making the Ghost Chase the PacMan

**Student Handout 2A:** Troubleshooting Guide for PacMan Part II
  Diffusion and Hill Climbing

**Teacher Instructions:** Part 3 – Making the game more sophisticated –
  Polling and Broadcast

**Student Handout 3:** Part 3 – Making the game sophisticated –
  Polling and Broadcast

**Student Handout 3A:** Troubleshooting Guide for PacMan Part II
  Polling and Broadcast

**Student Handout 4a:** Challenge: PacMan changes directions

**Student Handout 4b:** Challenge: PacMan moves continuously

**Student Handout 4c:** Challenge: Create Power Pellets

**Student Handout 4d:** Challenge: Move to next level
Vocabulary/Definitions

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

Attribute .............. a variable belonging to an agent (such as scent) also called a Local Variable by computer scientists.

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

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

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

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

Diffusion .............. the process in which an attribute’s value (in this game, scent) is calculated based on the scent values of the neighboring agents.

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

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

Method .............. a named set of rules evaluated by an agent in response to a message.

PacMan .............. the main character who eats the pellets as the user moves him around the world.

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.

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.

Simulation Property A named value that all agents can see and update.
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 where problems arise.

- 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 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

---

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.


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 AgentCubes) 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. Students may use provided images.
  
  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 AgentCubes. The ALTERNATIVE packet is for students with NO prior AgentCubes 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, Pac-Man is taken to the next stage. Enemy ghosts roam the maze, trying to catch Pac-Man. Pac-Man 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 Ghosts 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 AgentCubes. Use Handout 1A for those students.

*The student pages can be found on page 3 of the STANDARD packet and page 3 of the ALTERNATIVE packet.*

---

\(^2\) From http://en.wikipedia.org/wiki/Pac-Man
Student Handout 1:
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 world:

- PacMan
- Ghost with two depictions
- Pellet
- Ground
- Wall
- World with PacMan and Red and Green Ghosts

Create the following BEHAVIORS for your agents:

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

Step 2: PacMan:
- Make four rules so your agent moves in the right directions when each arrow key is typed (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 Do NOT See a Wall, THEN, I MOVE UP

Tip: Use the NOT button below the method to add a NOT to a condition:

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
Show a message so that the player knows why the game ended.

Very Important Tip: Put or in your Game over rule.
If you forget to do this, AgentCubes will do the game ending rule over and over until you are able to type the Return Key to click the OK on the dialog box and then immediately after use the mouse to click on the stop game button (the red square).
If you can not click on the stop game button before the dialog box reappears, you must shut down and restart your computer and then restart AgentCubes.

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?
- Did you show a message to tell the player why the game ended?
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 world:

- PacMan
- Ghost with two depictions
- Pellet
- Ground
- Wall
- World with PacMan and Red and Green Ghosts
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 AgentCubes 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 project button on the bottom right corner of the Project Chooser Window</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Step 2</th>
<th>Name the Game</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Type PacMan and click OK</td>
</tr>
</tbody>
</table>
Create the agents for PacMan

**Step 3**

Click on the +Agent button.

Choose Inflatable Icon, then Miscellaneous, then “packy” to get a picture of PacMan.

Type PacMan in the Agent Name box and click on the OK button.

**Step 4**

To edit the drawing of PacMan, double click on the picture next to the agent’s name.

Use the drawing tools to change the current image or click on the Clear button to erase the picture.

Use the pencil and the color picker (black square in the center of the row of drawing tools on the left of the 2D drawing) to draw a new picture.

Click on the color picker to choose a new color.

Use the bucket tool to fill in the outline with color.

Click on the + to inflate your drawing into a 3D image.

Use the Ceiling slider to flatten it.

Click the upright box to stand PacMan up.

Pick front and back connected next to the word “surfaces” so PacMan has 2 sides.
| Step 5 | Create the Ghost.  
If you go to the Miscellaneous category under Inflatable Icon, there is a agent named “ghosty”.  
You can edit the drawing and change the color or draw your own ghost.  
|---|---|
|   | ![Ghost Image](image.png)  
Here is a PacMan Ghost like the ones in the original game.  
| Step 6 | Make a second shape for your Ghost  
Click on the agent name, then click on the +Shape button at the bottom of the window.  
Give your new shape a name and click OK  
The double click on the picture next to the name of the new shape and edit it.  
|   | ![Shape Images](image.png)  
Change the ghost color so this shape looks different from the original ghost.  
|
### Step 7: Create the Wall
- Click +Agent and name it Wall.
- Select Cube and then pick a cube that you like.

### Step 8: Create the Ground
- Click +Agent Name it Ground.
- Select Tile and pick one that you like.

### Step 9: Create the Pellet
- Click +Agent Name it Pellet.
- Pick an inflatable icon, erase the picture, draw a pellet, use + to inflate & ceiling to flatten it.

### Step 10: Make a Level 1 World
- Click on the ‘+’ next to World.
- When the New World box appears, type the name into the Name box.
- Leave the size numbers alone for now.
### Step 11
**Design your World**

Use the dotted rectangle tool to place a rectangle of ground tiles on the World. Then use the same tool to place a rectangle of pellets on top of the ground. Use the same tool to place lines of wall cubes.

Use the pencil to draw single agents (PacMan, the Ghosts).

The Ghosts are in the rectangular area with an opening. PacMan is in the lower left corner.

### Step 12
**When you like the way your World looks, click on the Save button next to the World name.**

Save your World every time you make a change that you wish to keep.

When agents get moved around after playing or testing, click on the Reset button on the top bar of AgentCubes to return to the saved version of your World!
# Time to Create Behaviors for your Agents

<table>
<thead>
<tr>
<th>Step 13</th>
<th>Create an agent behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Click on an agent and its behavior window will appear below the world. You are going to drop and drag the conditions (on the left) and the actions (on the right) to create the rules.</td>
</tr>
</tbody>
</table>

This rule says:

**IF** the user presses the up arrow, **AND** I Do NOT See a Wall in the up direction, **THEN** my PacMan will move UP

Tip: Click on the See condition, then the NOT button at the bottom of the window to make the NOT see condition.

Create 3 more rules to make PacMan move right, left and down.

NOTE: Click on the rule you made, then click on the +RULE button at the bottom of the window to create each new rule.

<table>
<thead>
<tr>
<th>Step 14</th>
<th>Program the PacMan to eat the pellets.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>Take a look at this code when the PacMan sees the Pellet above him:</td>
</tr>
</tbody>
</table>

It says...

**IF** The Up Arrow is pressed **AND** I see a Pellet above me

**THEN**... Erase the Pellet above me **AND** Move up

NOW add 3 rules for the other directions.
### Step 15

**Program the Ghosts to move randomly**

Click on the agent to add behaviors to that agent.

*How can you make the Ghost move faster or slower? Experiment with the number in the `once every` condition.*

*Should the ghost also move on the pellets? Or only on the floor? You decide and code accordingly!*

<table>
<thead>
<tr>
<th>Step 15</th>
<th>Program the Ghosts to move randomly</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Click on the agent to add behaviors to that agent.</td>
</tr>
<tr>
<td></td>
<td><img src="image1.png" alt="Image" /></td>
</tr>
<tr>
<td></td>
<td><strong>How can you make the Ghost move faster or slower? Experiment with the number in the <code>once every</code> condition.</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Should the ghost also move on the pellets? Or only on the floor? You decide and code accordingly!</strong></td>
</tr>
<tr>
<td></td>
<td><img src="image2.png" alt="Image" /></td>
</tr>
</tbody>
</table>

### Step 16

**Create rule to end the game when the PacMan is next to the Ghost**

Click on PacMan and add this rule.

*Note: You need a similar rule for each different Ghost shape.*

This rule says:

**IF I am NEXT TO at least 1 Green Ghost**

**THEN, show a message “The Green Ghost caught PacMan! Try again!”**

**AND** **Reload the World**

**Very Important Tip:** Put **or** in your *Game ending rule.*

If you forget to do this, AgentCubes will do the game ending rule over and over until you are able to type the Return Key to click the OK on the dialog box and then immediately after use the mouse to click on the stop game button (the red square).

If you can not click on the stop game button before the dialog box appears, you must shut down and restart your computer and then restart AgentCubes.
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 Computational Thinking Patterns 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.

These computational thinking patterns require AGENT ATTRIBUTES.

An Agent Attribute is a VARIABLE that can be used LOCALLY (by an individual agent). Note that AgentCubes 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.]

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

[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.]

This YouTube video explains how dogs use scent for search and rescue
https://www.youtube.com/watch?v=XXXU0uKLWo0

This YouTube video of a MythBusters® segment shows how quickly sharks will respond to fish blood in a pool…students
may be disappointed to see that the sharks do not respond to human blood with the same enthusiasm! 
https://www.youtube.com/watch?v=gU9CQT-snIo

A more scientific explanation of how to determine if sharks can smell blood in the water. 
https://www.youtube.com/watch?v=uqv9EmfkkGE

**Explain:** In Part I of this project, the 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 handout 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 (shown below), and it must be the last rule for PacMan. What would happen if this rule appeared above another rule?

The blank if condition is **always** true, so this rule will always be activated, blocking whatever rules are below it.

This rule should be read as follows (s stands for scent):
**IF ALWAYS THEN set s to 1000**

Your students will make this mistake and it offers an opportunity for a class discussion about how to read rules and the effect of a rule with an empty condition.

What is the effect of a rule with an empty condition and empty action?
A blank rule should be read: **IF ALWAYS DO NOTHING**
A blank rule is always true and always does nothing AND it blocks all the rules below it!

---

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. The value of the agent attribute $s$ belonging to any pellet or ground agent is the average of the values of the $s$ agent attributes of the four surrounding agents:

$$s = 0.25 \times (s[left] + s[right] + s[up] + s[down])$$

Every 0.5 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
AgentCubes 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 time to create the Ghost’s Hill Climbing rules before using the shortcut to ensure that students understand the Hill Climbing computational thinking pattern. The use of the Hill Climbing action (see rule below) makes future extensions in the game, such as having the ghosts run from the PacMan, easier.

A learner may find the message action a helpful reminder of what the hill climb action does.

![Behavior: Ghost]

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!)

![Behavior: Ghost]
You could then extend the learning by asking students to compare what happens when the ghosts search 4 directions around themselves versus searching 8 directions around themselves. Have the students do some tests to see which method catches PacMan more rapidly.

http://www.conwaylife.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 5 of the STANDARD packet and page 11 of the ALTERNATIVE packet
Troubleshooting:

Students can use the plot action to visualize the value of S in the ground agents and the pellets. The plot action will plot the values of S logarithmically in a 3D surface above the world. The peak on the plot represents the highest value of S in the world. The peak should be over PacMan since his S value is 1000. In order to see more clearly what is happening, start by making a test world that just has a layer of ground agents, a layer of pellets, PacMan and one ghost.

Add the plot action to the rule in the ground agent and in the pellets.

To make the plot action work

1. Enter “S” for the name of the agent attribute.
2. Edit the row and column numbers so that values from the entire world are plotted. Remember that the default world size is 9 rows and 16 columns. Computer scientists always count from 0 to (Number – 1), in this case, rows 0 to 8 and columns 0 to 15. If you changed the default size, you must count from 0 to (New Size -1).
3. Pick a color for the plot that will show up against your background.
4. Leave the elevation at 3.0.
5. Make sure to choose “logarithmically” as the plot type. This option works better than plotting linearly when some of the values are quite large and others are very close to zero.
6. Run the game and move PacMan around. The peak, which represents the high value, will follow him. The ghost will move towards the high value (the peak).
7. Add a couple of rows of wall agents. What happens to the shape of the plot when the walls are added?
8. The walls have no S value so the plot gets some wrinkles and valleys.
9. Watch the ghost go around the walls following the increasing S values towards PacMan!
10. Now run your PacMan world with the complete maze on it and see what happens to the plot of the S values!
Figure 1. Plot of the S values with peak over PacMan.

Why is there a low spot over the ghost? Does the ghost have an S value?

Figure 2. Plot of the S values with some walls in the world.

PacMan and the ghost are under the peak. Why is there a valley over the wall?
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 “searching.” In this instance, we will use a specific method of searching 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 agent attribute a local variable.

Step 1:
First, let’s make sure our PacMan gives off a scent. To do this, we need to set PacMan’s attribute “S” to a value. S represents scent. There are several ways to do this. For example, we can create this rule at the end of the PacMan list of rules:

This rule says “Always set S to 1000”. AgentCubes interprets an empty condition as true. This rule must appear AFTER all the rest of PacMan’s rules, at the bottom of the while running method.

Challenge:
What will happen if you put this rule above PacMan’s movement rules? Will PacMan be able to move?

Another way to initialize PacMan’s S agent attribute is to set it when PacMan is drawn on the World. To do this, create a new Method by clicking on the +Method button. Click on the word “on” in the new method’s black and yellow striped tape and change the label from “on” to “when creating new agent”.

Student Handout 2

Part 2 – Making the Ghost Chase the PacMan
Your when creating new agent method should look as follows:

If you use this method to set PacMan’s S attribute, make sure that you erase and redraw PacMan and then SAVE the World.

**Important Note:** If you forget to save the world, PacMan may not have any value set for S when the world is reloaded.

**Checking the Value of PacMan’s S agent attribute:**
1. Select PacMan on the world.
2. Select **Show Agent Attributes** from the AgentCubes’ Window Menu.
3. You should see this window appear:
4. If S is not visible in the window, it did not get set to a value yet.
5. If you put the set S action in PacMan’s while running method, run the game briefly. S should appear in the Attributes window.
6. If you used the when creating new agent method, erase and redraw PacMan and then save the world. S should appear in the Attributes window.

**Step 2:**
Now, since the scent is diffusing, or spreading out, we need to find the average of the scent from the area around a ground agent or a pellet agent. 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 named the agent attribute “S” (for scent).

The “set” action sets each pellet agent’s attribute “S” to the average of the attributes in the agents above, below, and on each side:
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

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

NOW...diffuse the scent across the ground by adding a rule to the ground agents!

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

**Helpful Tips**

What do FIRE ALARMS have to do with coding?
A METHOD is a set of rules with a name...rules to follow in a specific situation. These are done when there is a specific call for them...much like the fire alarm means you follow different rules. You can create a METHOD by clicking the +Method button below an agent's behavior.

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.

The rule in the while running method says
“ONCE EVERY 0.5 seconds, follow the Chase PacMan procedure”.

The rule in the Chase PacMan 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 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, check the following:
- PacMan’s “set S to 1000” rule must be the last rule in the while running method.
- If you used the when creating new agent method to set S to 1000, erase and redraw PacMan, then save the world so the initial value of S is saved.
- 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 of the hill climbing method!
- Use of parentheses “(“ and brackets “[“ in the ground and pellet agent rules must be correct. Check the picture of the ground agent’s equation 2 pages ago and compare it to the equations in your ground and pellet agents.
- 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.
Student Handout:
Troubleshooting Guide for Diffusion and Hill Climbing –
Part 1: Tracking the Ghost One Step at a Time

To determine what is happening in your game, it is helpful to look at the agent attributes.
1. On your world, click run until the ghosts move out of the box, and then click stop.
2. Do not reload the world.
3. PacMan’s scent (the value of S) has diffused across the world.
4. To see how PacMan’s scent (the value of S) has diffused, click anywhere on the ground.
5. Then on click on the AgentCubes Window menu and select the Show Agent Attributes option.
6. A box will appear that lists the S attribute value for that agent.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>143.89934</td>
</tr>
</tbody>
</table>

7. Click around the world. Is S biggest close to PacMan and smaller far away from PacMan?

Check the attributes of the four boxes around the Ghost (up, down, left and right) and then single-step the game using this button so you can see if your Ghost is moving towards the agent with the largest S value.

If the Ghost moves the wrong way or does not move, go back and check your rules in the Chase PacMan method. Compare your rules with a friend’s rules.
- Are the conditions correct?
- Are the arrows in the move actions correct?
- Check that PacMan has an S value by clicking on him and looking at the attributes dialog box. See previous page for suggestions about setting PacMan’s S value.
Trouble Shooting Guide for Diffusion and Hill Climbing – Part 2: Visualizing the Values of S in Your World

Use the plot action to visualize the value of S in the ground agents and the pellets. The plot action will plot the values of S in a 3D surface above the world. The peak on the plot represents the highest value of S in the world. Can you predict which agent has the highest S value?

In order to see more clearly what is happening, start by making a test world that just has a layer of ground agents, a layer of pellets, PacMan and one ghost.

Add the plot action to the rule in the ground agent and in the pellets.

To make the plot action work
1. Enter “S” for the name of the agent attribute.
2. Edit the row and column numbers so that values from the entire world are plotted. Remember that the default world size is 9 rows and 16 columns. Computer scientists always count from 0 to (Number – 1), in this case, rows 0 to 8 and columns 0 to 15. If you changed the default size, you must count from 0 to (New Size -1).
3. Pick a color for the plot that will show up against your background.
4. Leave the elevation at 3.0.
5. Make sure to choose “logarithmically” as the plot type. This option works better than plotting linearly when some of the values are quite large and others are very close to zero.
6. Run the game and move PacMan around. The peak, which represents the high value, will follow him. The ghost will move towards the high value (the peak).
7. Why is there a hole in the plot over the Ghost? Does the Ghost have an S agent attribute?
8. Add a couple of rows of wall agents. What happens to the shape of the plot when the walls are added?
9. The walls have no S value so the plot gets some wrinkles and valleys.
10. Watch the ghost go around the walls following the increasing S values towards PacMan!
11. Now run your PacMan world with the complete maze on it and see what happens to the plot of the S values!
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. A teacher does this by asking students to raise their hand if they are still working. If no one raises a 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 world. Remind the students that they should take time to think through each programming step so they can use these skills later.

Hand out handout 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 17 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. Polling will tell us when all the pellets are gone and PacMan has won.

Polling uses a simulation property, also called a global variable by computer scientists, which is a piece of information that all agents in the simulation or game may check or set if they have the correct rules. A controller agent does the polling by sending out a message at intervals to all the agents that must be counted. These agents respond by adding one to the simulation property. The controller determines when all the pellet agents are gone and PacMan has won.

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 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.
Step 1: Create the Pellets simulation property as described in the green box below.

Step 2: Create the Controller agent.
- Use +Agent to make a Controller agent and choose any image.
- Place the Controller agent on top of a wall in your PacMan world.

Step 3: Add a rule to the Controller agent’s while running method.
1. Set the number of pellets to zero. (this is like the teacher saying “hands down”)
   \textit{Set @Pellets to zero}
2. Ask the pellets if they are still on the world
   \textit{Broadcast to Pellet agents to do “Count”}
3. Use the count of the pellets to see if the game is done.
   \textit{Send a message to myself (to do) Checkwin}

Step 4: Program the Controller agent’s Checkwin method.
- If there are no pellets left, tell the player that PacMan won and stop the game.

Step 5: Program the Pellet agent’s Count method.
1. Make a new method for the Pellet agent.
2. Name it Count.
   The name must exactly match the name broadcast by the Controller.
3. Add a rule with an action that sets the value of “@Pellets” to “@Pellets + 1”.
   This is how programmers add 1 to a number.

Try setting up these rules now!

\textbf{Helpful Tips:} \textit{Count is not part of the continually running “While Running” method. It must be a separate method since it only runs when called by the controller agent.}
Check your program:

Here is the Controller agent behavior with the rule in the while running method that makes the pellet agents count themselves and the new Checkwin method that ends the game if PacMan has eaten all the pellets:

Here is the Pellet behavior with the new Count method that allows each Pellet agent to add 1 to the Pellets simulation property:
Student Handout:

Troubleshooting Guide for PacMan Part III

Polling and Broadcast

Common Problems:

1. Is your Controller agent saved on the world?
2. Did you type in @pellets whenever the simulation property was checked or changed?
3. Do you refer to the correct agents in each step?

Another Approach to Troubleshooting:

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

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pellets</td>
<td>114.0</td>
</tr>
</tbody>
</table>

The correct number of Pellets 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 Pellets will decrease by 1 each time PacMan eats (erases) a Pellet. When the value of Pellets is equal to 0, PacMan 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:

![Controller Rule](image)

In the plot to window action, you must name the simulation property to be plotted (Pellets), name the window where it will appear (Pellets Plot), say what it represents (number of pellets) and pick the color of the line that will appear on the graph. **Note that you must put “@” before the Pellets in the plot to window box because you are checking the value of the simulation property Pellets!**

Look under the AgentCubes Windows menu and select the Pellets Plot window. Move the Pellets 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.

![Pellets Plot Graph](image)

This information will help you determine where a mistake may be. For example, if the number of pellets never goes above 0, there is a problem with the method Count or the broadcast. If the number of pellets 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 - 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 AgentCubes.
      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 (S for smell) on the agent which should be chased, and we diffuse the smell by setting the attribute on the background using the average of the 4 smells around it,

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

   2) **Hill Climbing**: following the highest value of the scent S. It only works if there is diffusion done with it, so they go hand in hand. Example is the method we created in the Ghost to move towards the agent next to him with the highest value of the scent “S”.

   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 AgentCubes.
      b. Example is the broadcast by the Controller of the method “Count” to the pellets so they will count themselves.

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.
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 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 world 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 3 new shapes!

Steps:

- Select PacMan by clicking on him, then click on the +Shape button and give this shape a name.
- Double click on the picture next to the new name and the drawing panel will appear.
- Clear the picture and draw PacMan facing a different direction.
- Once you have the four different shapes, change your move rules so that PacMan faces the direction that he is moving.
- Use the change action with a dot in the middle because the means “change me to” so the agent is able to change its shape.
- TEST your program to confirm that the PacMan’s shape changes when he changes directions as he moves.
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 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 world should have walls that the Ghost and PacMan cannot cross.

You must have 4 different shapes for 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 right arrow is pressed AND I do not see a wall to the right, change to the right-facing depiction. Once every 0.2 second, make me (the PacMan) do MoveDirection.

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

If I see myself heading right AND I do not see a wall in the right direction, I will move right.

The effect of the rule in MoveDirection is to make PacMan keep moving whichever way he is facing as long as there are no walls in the way.
There is still much to code:

**Step 1:**
Create code for all the other directions.

**Step 2:**
**Test your program.** (Hint: be sure your PacMan still leaves his scent everywhere.) **Click** on the PacMan with the big arrow tool to select him 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 Right arrow was not pressed or a wall was in the way.

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

The method **MoveDirection** is green, which means that either or both conditions are true. The PacMan does sees his right-facing shape AND does not see a wall, making the rule TRUE, so PacMan will move one step to the right.
Student Handout 4c:

Power Pellet Challenge

Before your start this challenge:

You must have a complete basic PacMan game with a PacMan who wins if 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 world 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 world.
• Power Pellets provide PacMan 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 or a new shape for the blue ghost?

• When the ghost chases the PacMan, PacMan has a scent of 1000. What happens if he has a scent of -1000? How can you set that new scent?

• How can you limit the time that PacMan’s scent is -1000? Could you create a timer agent that starts counting when it receives a message from PacMan that he ate a Power Pellet? The timer agent should send a message back to PacMan when it is done counting and it’s time for PacMan’s scent to return to 1000.

• 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 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 world 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 world?
- When would a new level appear?
- What code needs to change to make the new level appear?

You might have a rule like this:

```
if test 0Pellets = 0
then
  play sound Hallelujah.mp3
  show message PacMan won!
  stop simulation
```

How could you use this condition and this action to let the player move from a world named “Level 1” to a world named “Level 2”?

**Very Important Note:** Add another rule that stops the simulation if the player has won Level 2 **so the game ends!**
### ISTE Standards specific to the implementation of PacMan (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

---

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 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.