

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.

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.

Student Handout 1:
Part I - Basic Game

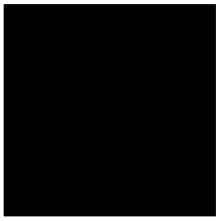
Initial Story: Create the quintessential arcade game of the 80's! Wind your way through a maze while eating pellets. Watch out for the ghosts!

Create these Agents and the world:

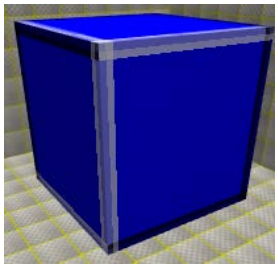
PacMan

Ghost with two depictions

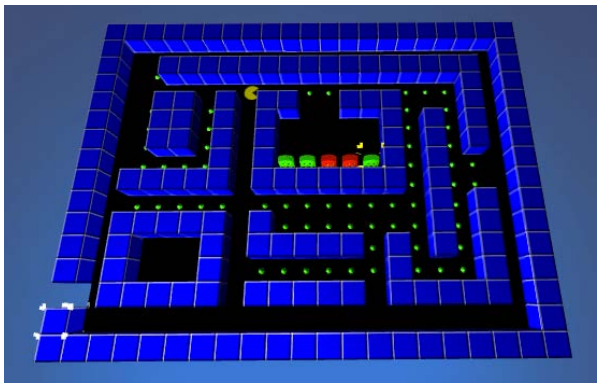
Pellet



Ground



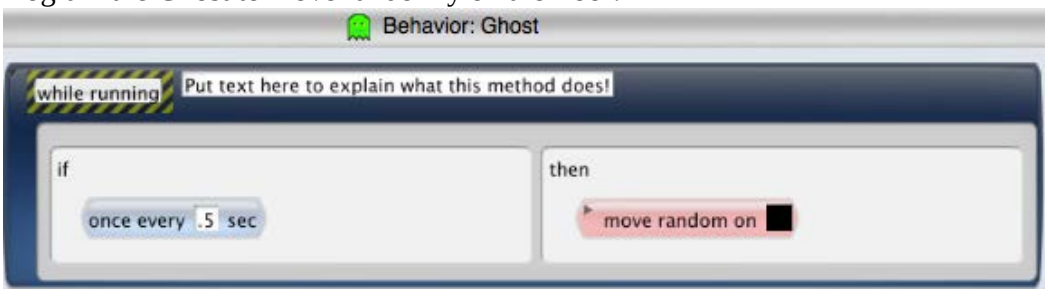
Wall



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:

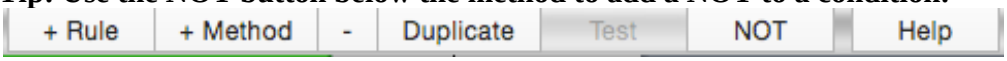
- Make four rules so your agent moves in the right directions when each arrow key is typed (cursor control).

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

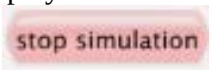
The event that occurs when two agents are next to one another

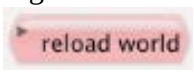
IF the PacMan and the Pellet COLLIDE, THEN the Pellet should ERASE

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 2

Part 2 – Making the Ghost Chase the PacMan

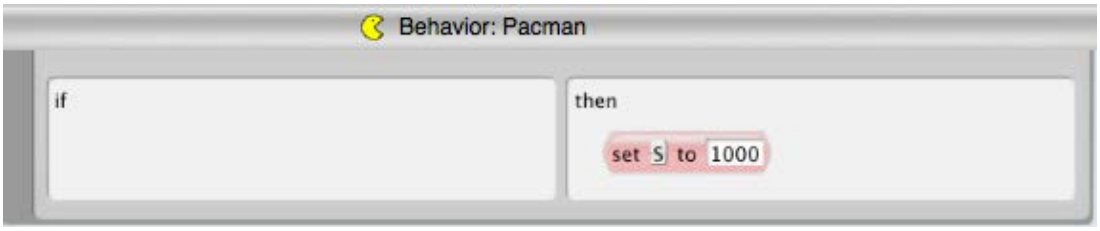
So far, your Ghost just moves randomly, either just on the floor, or on the floor and the pellets…he doesn't actually chase the PacMan, does he? That's about to change!

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[1].

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:



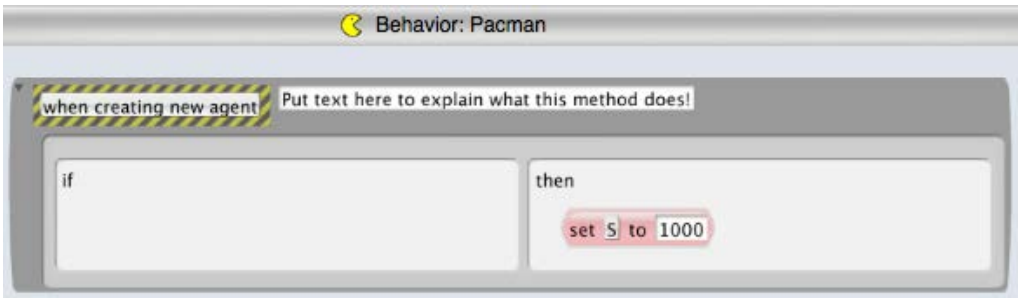
This rule says "Always set S to 1000". AgentCubes interprets an empty condition as true. This rule must appear AFTER all the rest of PacMan's rules, at the bottom of the while running method.

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

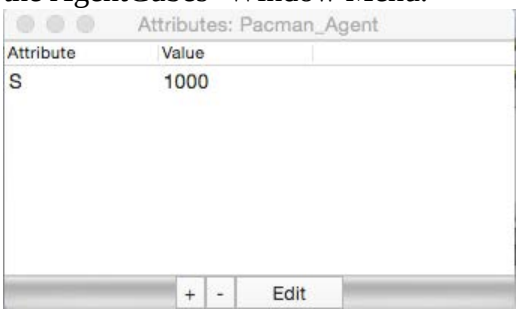
"on" in the new method's black and yellow striped tape and change the label from "on" to "when creating new agent".

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

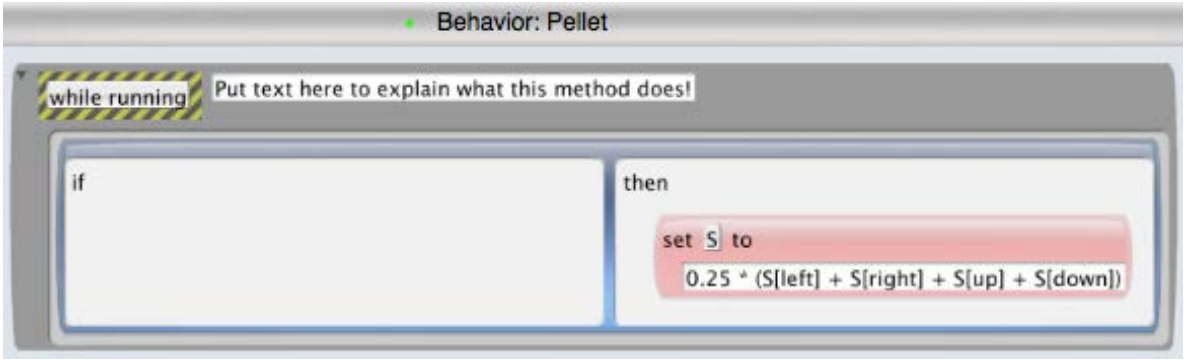
The pellet agent will have the behavior below; the single action is to calculate and store the average of the four surrounding agents' agent attributes. Remember, you named the agent attribute "S" (for scent).

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:

S = 0.25*(s[up]+s[down]+s[right]+s[left])

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

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



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

Helpful Tips

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

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.

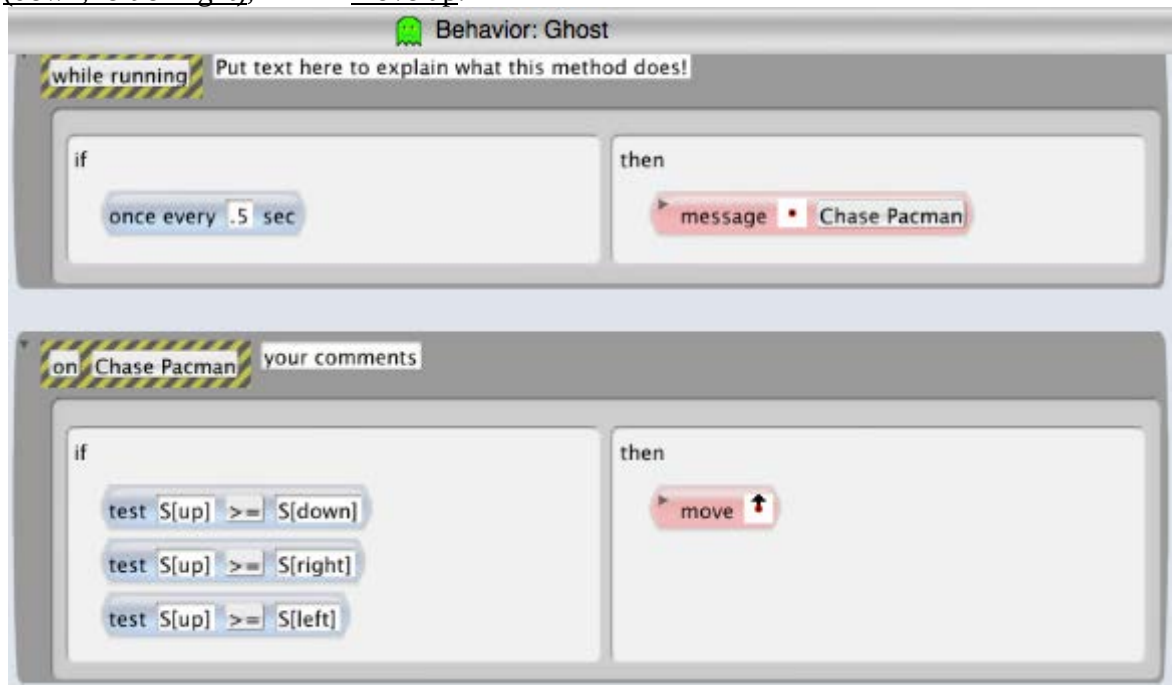
Step 3: For the Ghost to know which way to walk, he has to determine where the scent is the strongest. We call this HILL CLIMBING. If this were real life, he would smell up, smell down, smell left and smell right. Wherever the smell was strongest, he would walk in that direction. We need to program the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

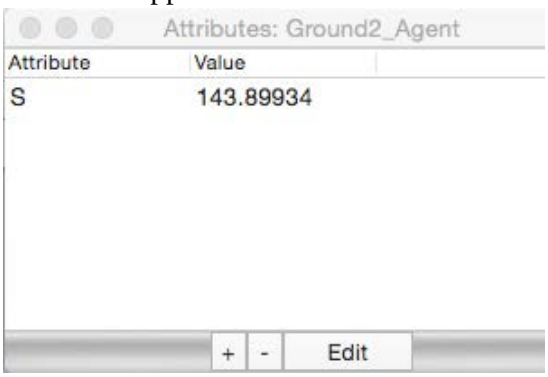
- PacMan's "set S to 1000" rule must be the **last** rule in the while running method.
- If you used the when creating new agent method to set S to 1000, erase and redraw PacMan, then **save** the world so the initial value of S is saved.
- In the Chaser's rules, the method name must be the same in the 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

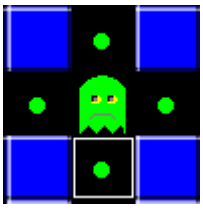
To determine what is happening in your game, it is helpful to look at the agent attributes.

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.



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.

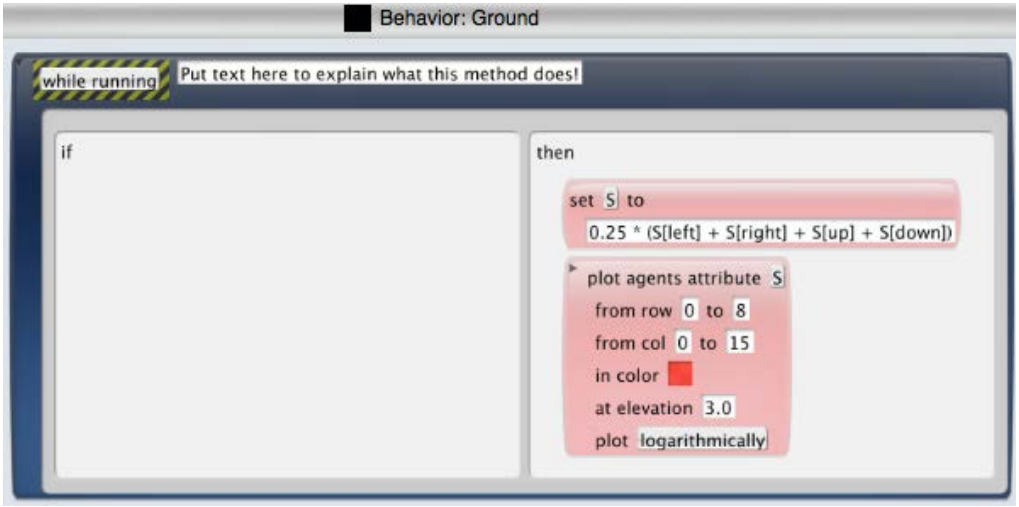
- Are the conditions correct?
- Are the arrows in the move actions correct?
- Check that PacMan has an S value by clicking on him and looking at the attributes dialog box. See previous page for suggestions about setting PacMan's S value.

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!

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.

Definition: Computer scientists call the process of making a decision by sending a message to multiple recipients and checking responses **polling**.

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")
   Set @Pellets to zero
2. Ask the pellets if they are still on the world
   Broadcast to Pellet agents to do "Count"
3. Use the count of the pellets to see if the game is done.
   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.

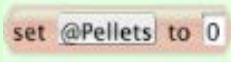
Step 5: Program the Pellet agent's Count method.

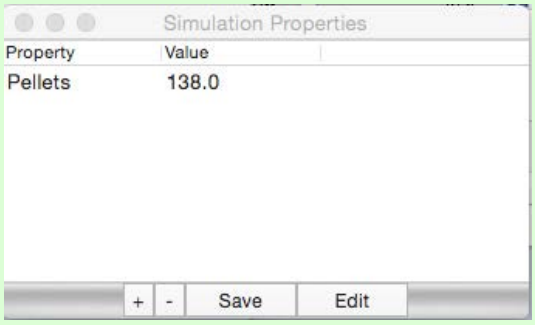
1. Make a new method for the Pellet agent.
2. Name it Count.
   The name must exactly match the name broadcast by the Controller.
3. Add a rule with an action that sets the value of "**@Pellets**" to "**@Pellets + 1**".
   This is how programmers add 1 to a number.

Try setting up these rules now!

How do Simulation Properties Work?

1. Simulation properties may be added, saved or deleted in the Simulation Properties window.
2. Open this window using the Show Simulation Properties option of the AgentCubes Window menu.
3. Make the Pellets simulation property by clicking on the + button at the bottom of the simulation properties window and typing the name.
4. The value of a simulation property can be changed or checked by any of the agents in the game or world.
5. All conditions and actions which check or change the value of a simulation property must place an "@" before the simulation property name.

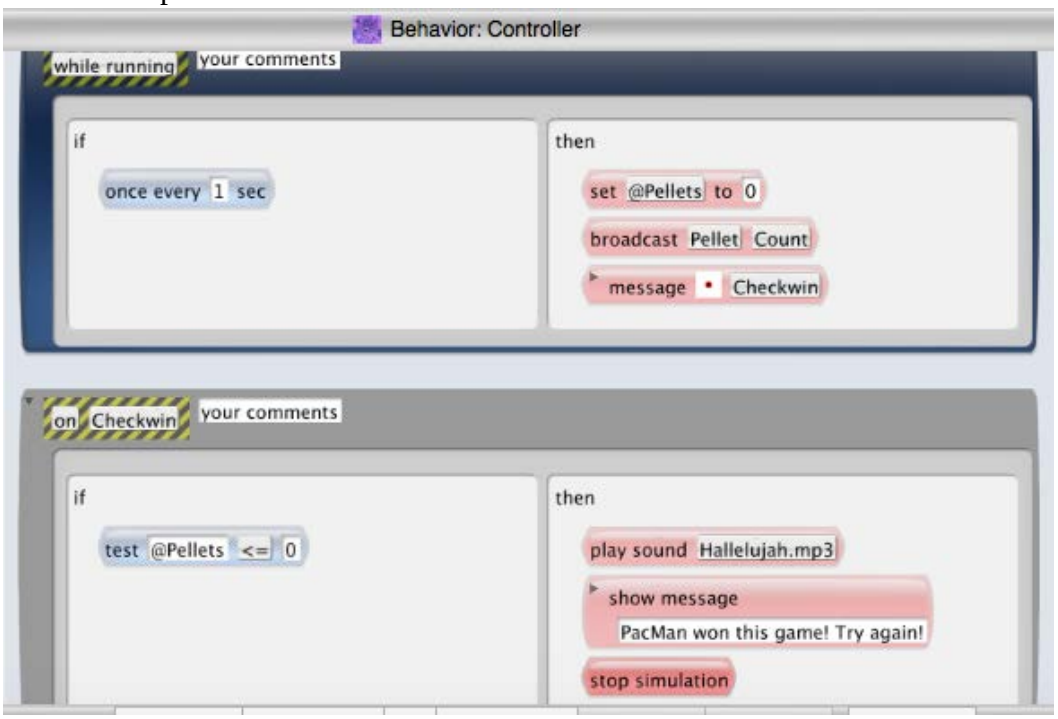




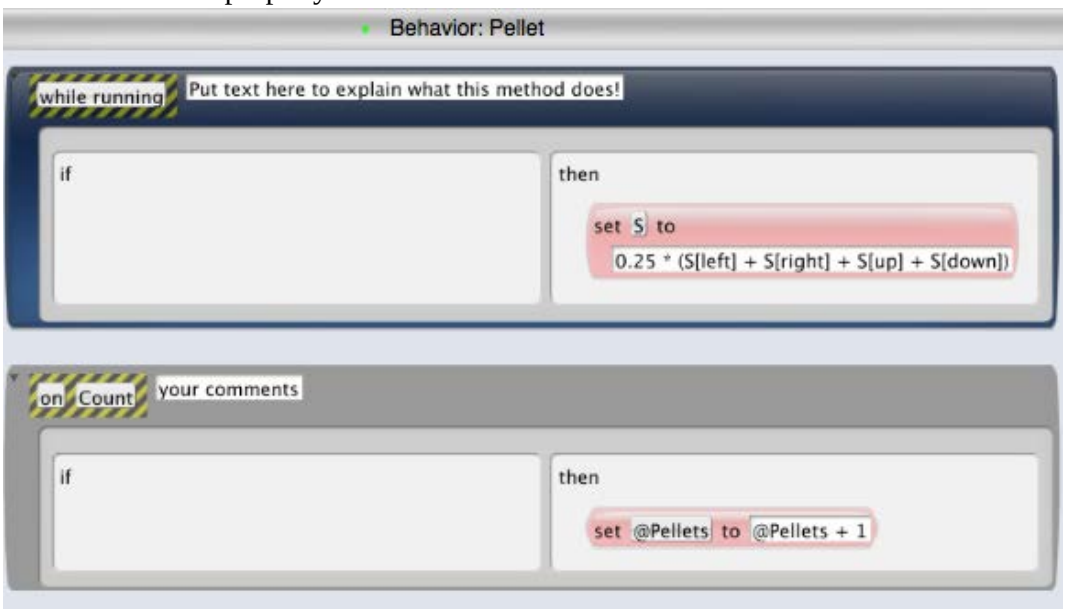
Count is not part of the continually running "While Running" method. It must be a separate method since it only runs when called by the controller agent.

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?**

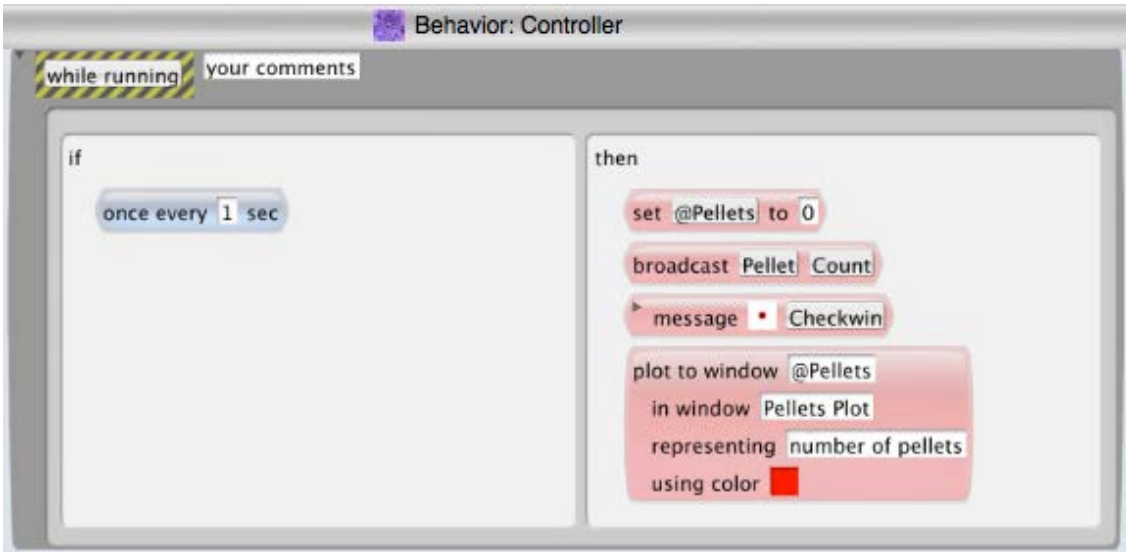
Another Approach to Troubleshooting:

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:

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:



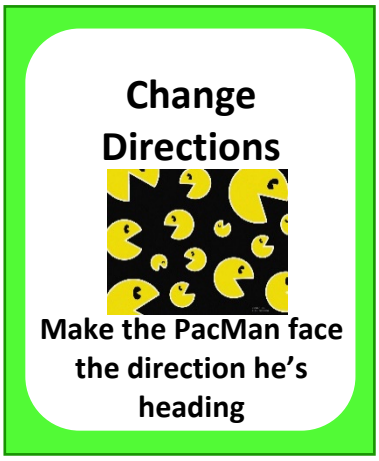
In the **plot to window** action, you must name the simulation property to be plotted (Pellets), name the window where it will appear (Pellets Plot), say what it represents (number of pellets) and pick the color of the line that will appear on the graph.
Note that you must put "@" before the Pellets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

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.

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:

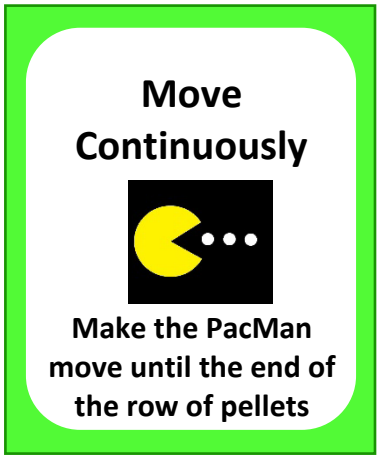
- Select PacMan by clicking on him, then click on the +Shape button and give this shape a name.
- Double click on the picture next to the new name and the drawing panel will appear.
- Clear the picture and draw PacMan facing a different direction.
- Once you have the four different shapes, change your 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.

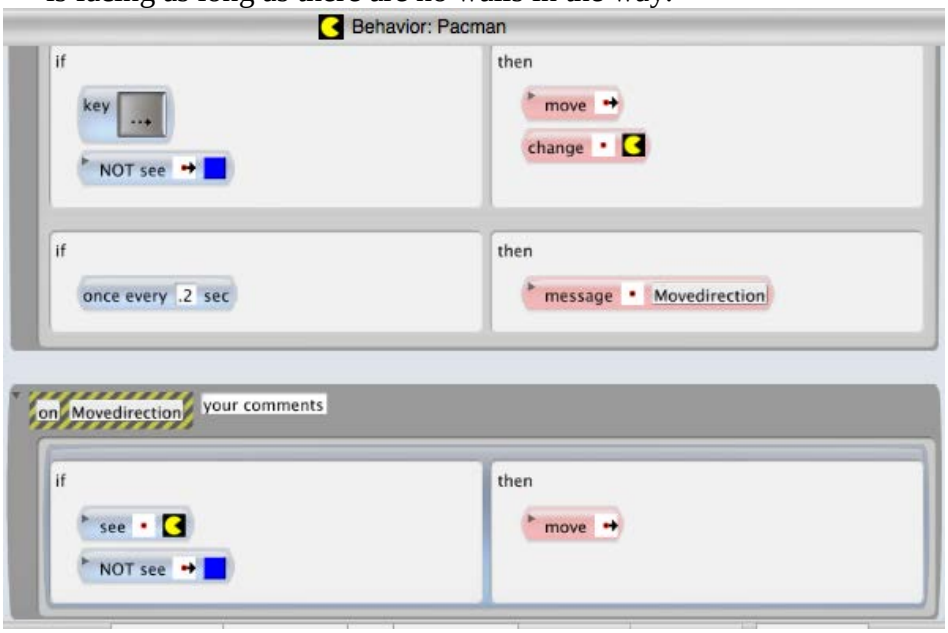
Description of the Challenge:

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**.**

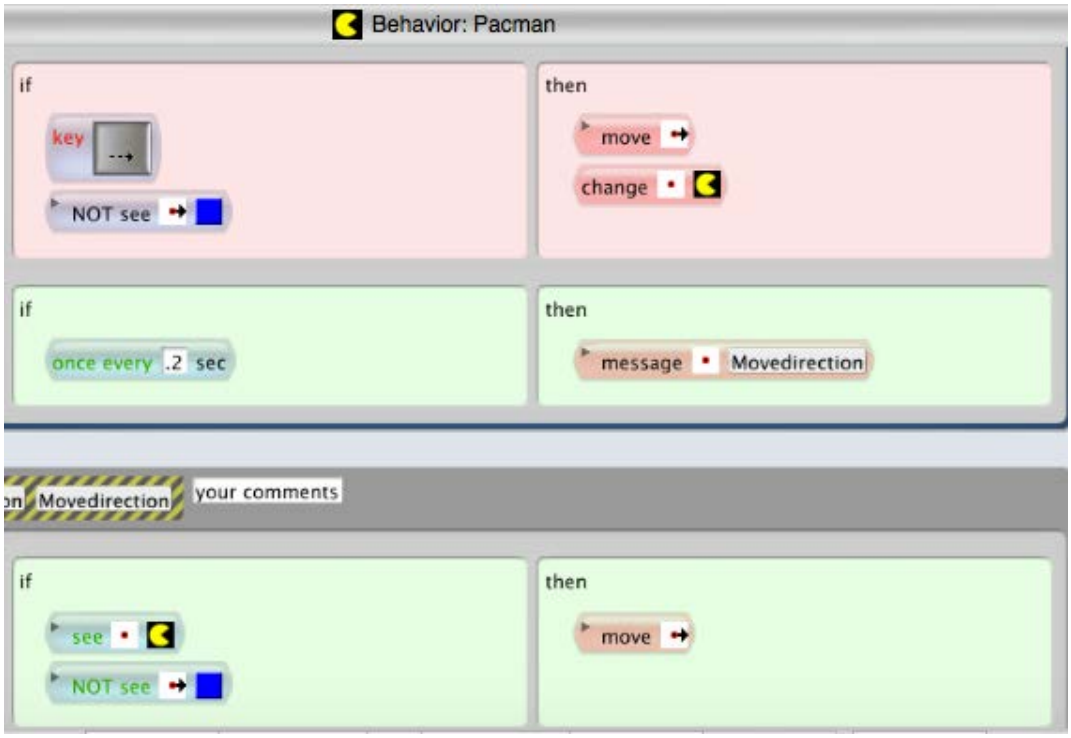
Step 2:

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.

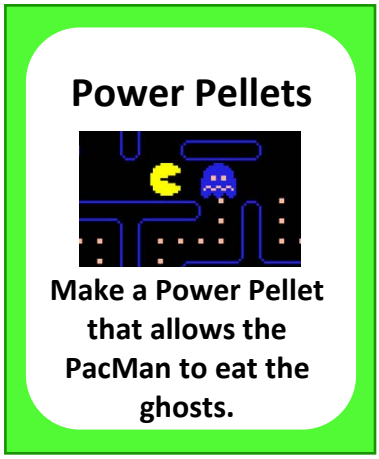
The method MoveDirection is green, which means that either or both conditions are true. The PacMan does sees his right-facing shape AND does not see a wall, making the rule TRUE, so PacMan will move one step to the right.



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:

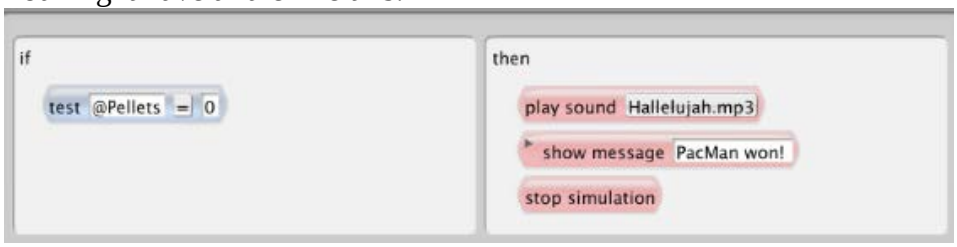
- When the game ends, a new level appears, even harder than before!

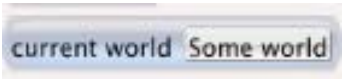
This challenge gets you started, but won't give you all the code.

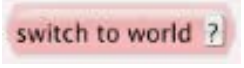
To help you think this through…

- 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:



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**!