Creating "PacMan"

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

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

PacMan

## Vocabulary/Definitions

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

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

Brackets ....method of setting information apart using "[" and "]"

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

Ghost .... the agent that chases the PacMan

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

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

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

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

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

Parentheses..........method of setting information apart using "(" and ")"

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

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

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

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

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

Student Handout 1A:

Part I - Basic Game

Initial Story: Create the quintessential arcade game of the 80's! Wind your way through a maze while eating pellets. Watch out for the ghosts!
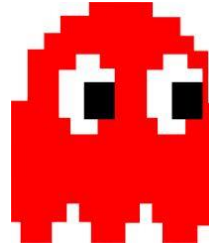
Create these Agents and the worksheet:
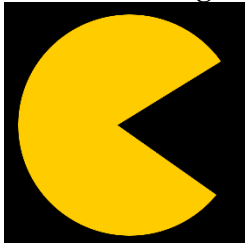
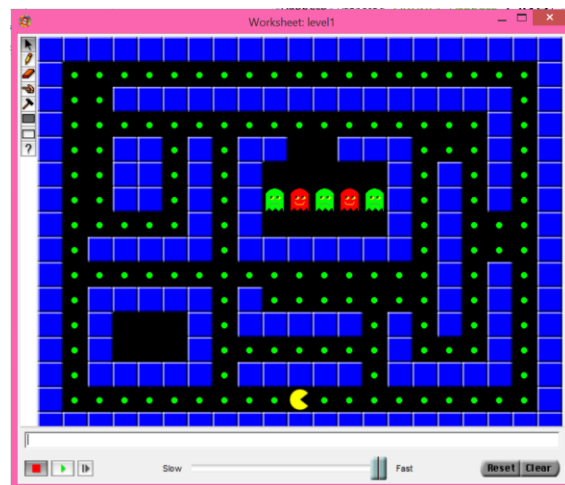**PacMan**   **Ghost with two depictions**   **pellet**
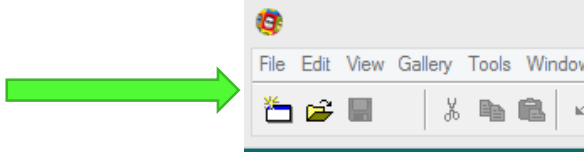
Background   **Wall**
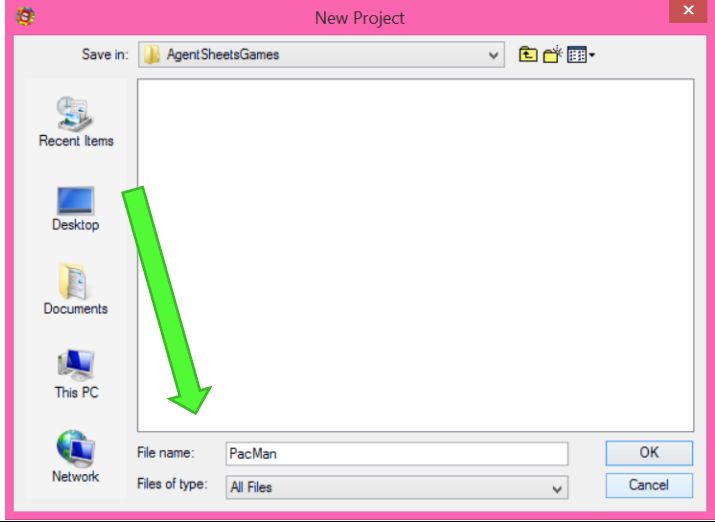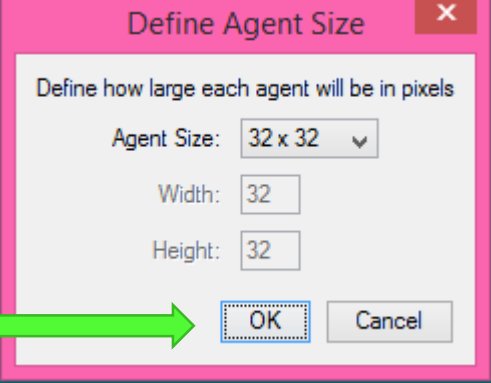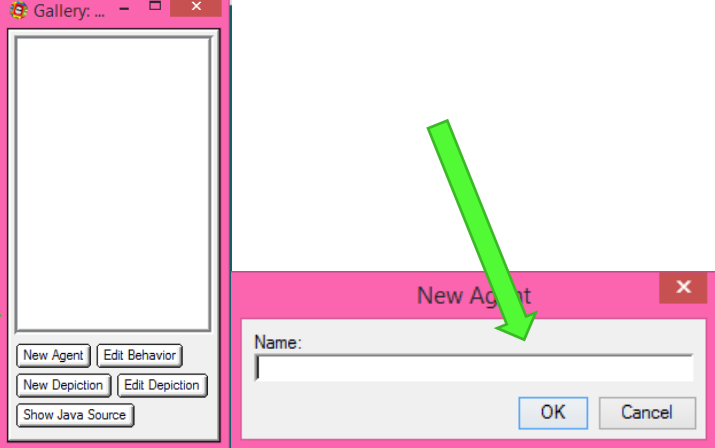
You are about to create your first game. Because this is new to you, we will give you some parts of the code. Be careful – you will have to find the patterns to create the rest of the code. If you get stuck, check in with a nearby student to get some help!

Launch AgentSheets and Create a New Game
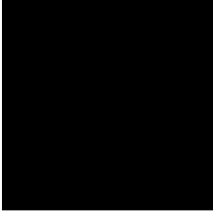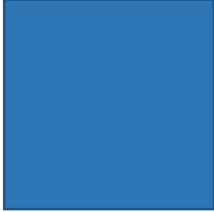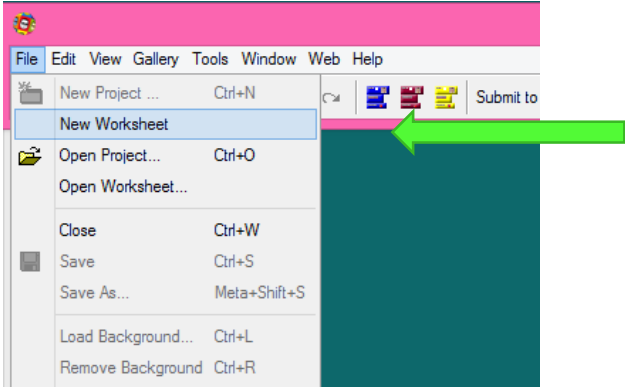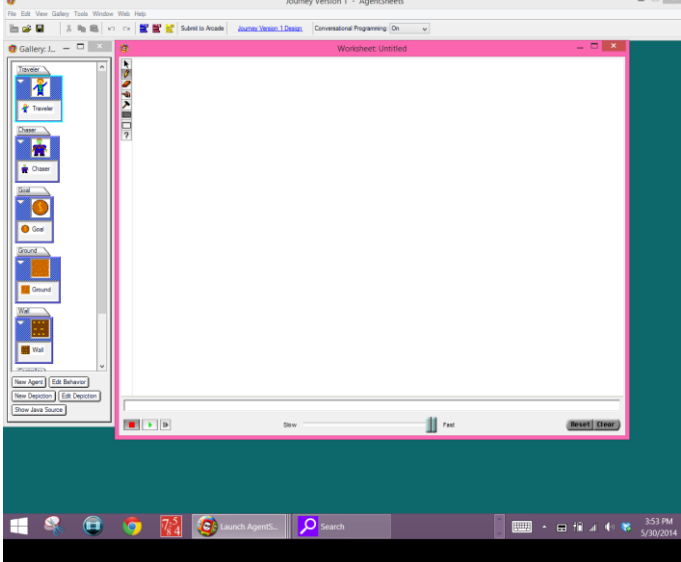
Step 1	Create Game Click on the new game icon (far left)	
Step 2	Name the Game Name it PacMan and click OK	
Step 3	Define Agent Size Do not change - Click OK	
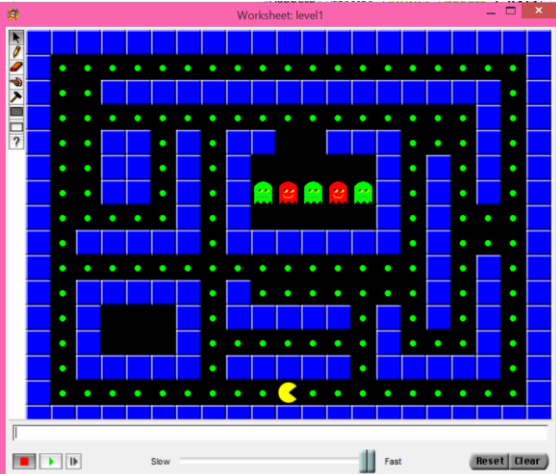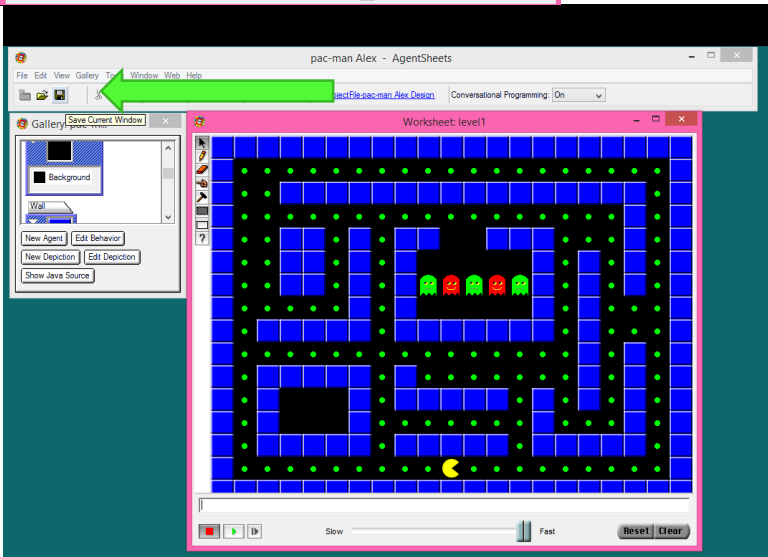
Launch AgentSheets and Create a New Game

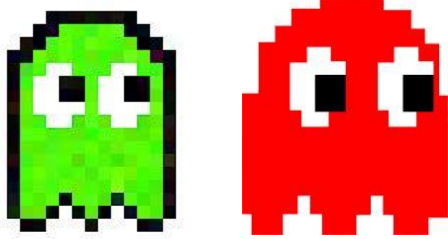| Step 4	**Create Agent** **Click on New Agent** **Name it PacMan** **Click OK**	
Step 5	**Edit Agent** **Click EDIT DEPICTION** **Click CLEAR to erase the current image.**	*Click on Color> Mask Color>> White to make the white background sections see-through*
Step 6	**Draw PacMan** **Click DONE**	Here is an example of one way to draw him. You can be creative. If you make a mistake, use the eraser or click CLEAR to clear the whole area.
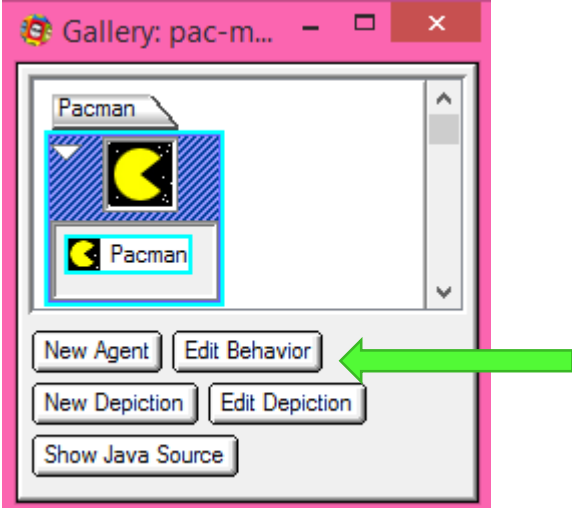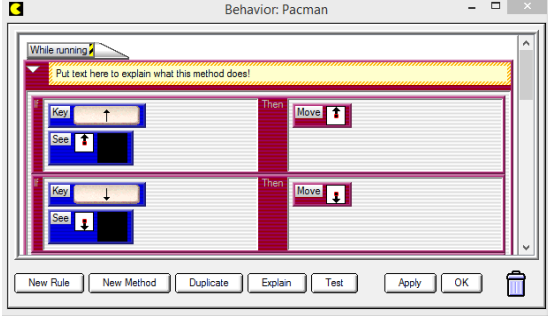
Step 7	Draw remaining agents	
		Ghost with two depictions **Pellet** (Create a ghost agent and design as the green ghost. Then click New Depiction and create the red ghost) **Background** **Wall**
Step 8	**Make the workspace** **Click File>>New Worksheet**	
Step 9	**Make the worksheet bigger** **Notice it is big, but not so big that it fills up the whole space.**	
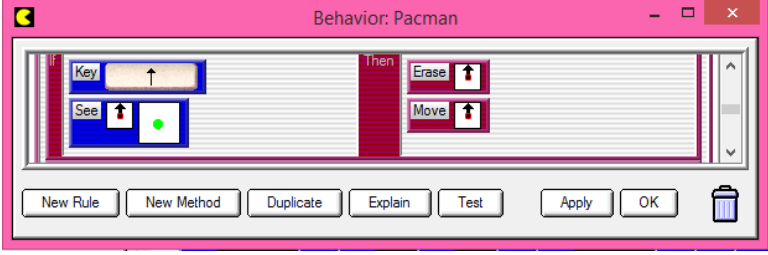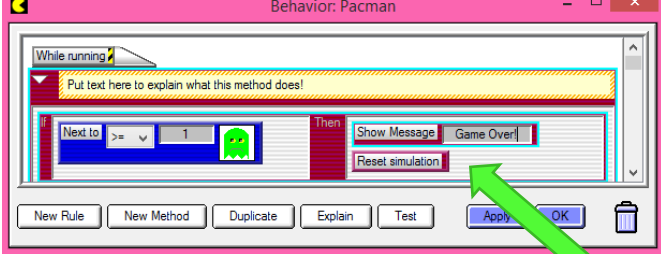
Step 10	Use the tools to place items on the worksheet. Pencil: places items one at a time Filled in Rectangle: Places items in an array (rectangle).	**It is important that you do not draw over an agent with the Ground agent. This means if you place a pellet on the worksheet, do not draw the Ground over it without erasing the pellet first.**
Step 11	**SAVE your WOKRSHEET!!**	

PacMan (Continued)

Time to Create Behaviors for your Agents

Step 11	Create behaviors for your agents Read the explanation and then Click Edit Behavior	The behaviors that we will give to our agents are called rules. Rules are made up of one or more IF-THEN statements. To control the PacMan using the cursor keys, one of the rules we need should be that "IF the Up key is hit, THEN the PacMan will move up." We also want to add a condition that says the PacMan only walks on the floor. Overall we should have 4 rules, one for each direction (Up, Down, Left, Right).
Step 12	Create a behavior When you first open this, it will be blank. You are going to drop and drag the conditions (on the left) and the actions (on the right) to create the rules.	**Take a look at this rule…it says,** **IF the user _presses the up arrow_,** **AND I See the FLOOR in the up direction,** **THEN my PacMan will move UP** **Create the rules to have the PacMan move up, right, left and down.** **NOTE: Each rule has to be separate…use NEW RULE to create each new rule.**

Step 13	Program the PacMan to eat the pellets.	First, take a moment to think about the PacMan and the Pellets. Does the PacMan really eat the Pellets? No, he's just a picture and can't eat anything. Instead, it's an illusion. When the PacMan is next to the Pellets, then the Pellet disappears, making it look like the PacMan ate it.
		Take a look at this code when the PacMan sees the Pellet above him:
		It says…
		IF…
		The Up Arrow is pressed Erase the Pellet above me
		AND THEN… AND
		I see a Pellet above me Move up
		NOW…Add the 3 rules for the other directions
Step 14	Program the *Ghost* to move randomly	*Click on the agent to add behaviors to that agent*
		Should the ghost also move on the pellets? Or only on the floor? You decide and code accordingly!
Step 15	Create rule to end the game when the PacMan is next to the Ghost Click on PacMan and Edit Behavior Add these rules	This rule says: IF I am <u>NEXT TO one or more GHOSTS</u> THEN, <u>give a message</u> "Game Over!" and <u>Reset the Simulation</u> Don't forget the last action – reset simulation!!!

Student Handout 1B:
     Agent Creation short-cuts
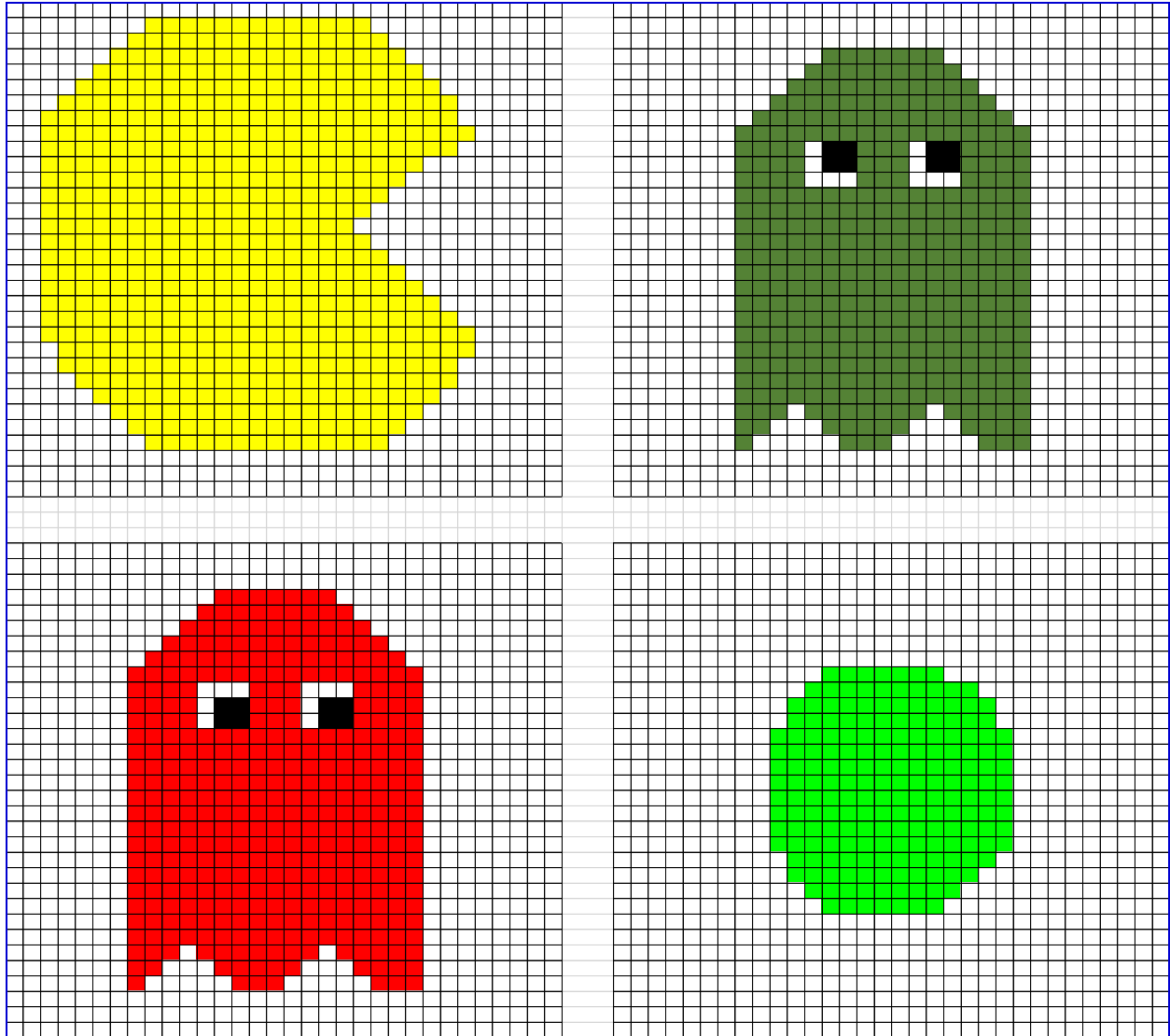
Student Handout 2
Part 2 – Making the Ghost Chase the PacMan

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 "seeking." In this instance, we will use a specific method of seeking called Hill Climbing. Imagine the PacMan agent emits a scent. Hill climbing is a procedure or **algorithm** to find the direction in which the scent is strongest.

The scent will spread out, or be **propagated,** by the ground agents using a computational thinking pattern called "diffusion." Diffusion is a fundamental process (physical, biological, and social) by which objects move from areas of highest concentration to areas of lowest concentrations. The closer to the source of the scent, the greater its value[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 attribute a **local variable**.

Step 1:
First, let's make sure our PacMan gives off a scent. To do this, we need to set an attribute "s" (We have given the arbitrary name of the agent attribute "s" for scent) for the PacMan. There are several ways to do this. For example, we can create this set at the end of the PacMan list of rules:
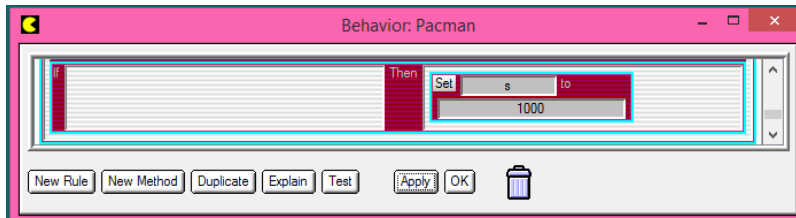
This rule says to the PacMan, "If I am not doing anything else, I will emit a scent at level 1000 wherever I am." **This rule should be AFTER all the other rules for the PacMan, at the end of the list.**

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

---

[1] Note that this method simulates the beginning of the diffusion process – however, in the actual diffusion process, the smell would become evenly distributed eventually, which will not occur in this model.

Step 2:

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

Diffuse the scent with the pellets

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

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

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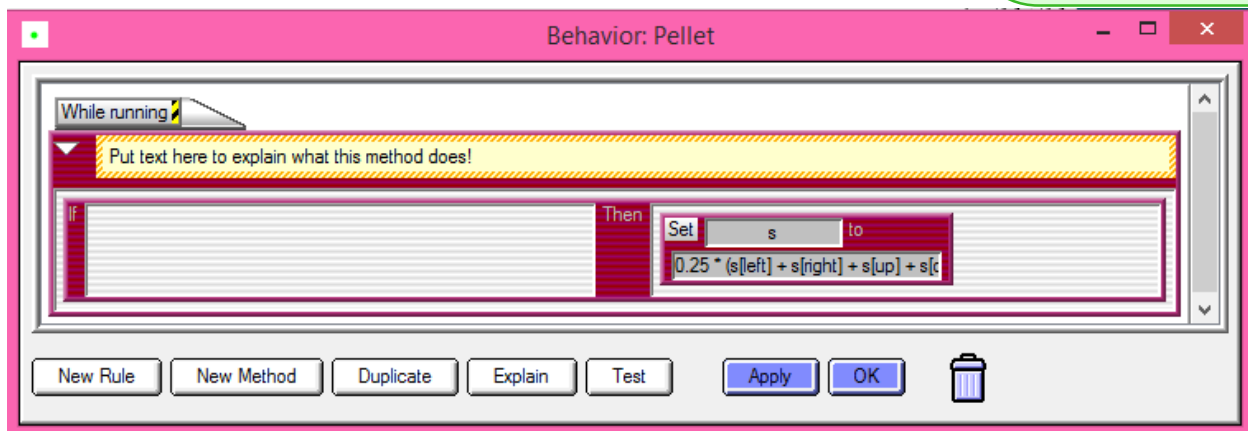
OR

$$S = (s[up]+s[down]+s[right]+s[left])/4$$

> **Why do we multiply by 0.25?**
> When you find the average of a set of numbers, you add them up and divide by the number of numbers.
>
> In this case, dividing by 4 is the same as multiplying by 0.25

NOW…diffuse the scent across the ground!

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

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 "New Method" at the bottom of the behavior box for the Ghost.

Step 3:

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

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

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

ONCE EVERY 0.5 seconds, follow the Navigate procedure.

IF <u>the smell **above you** is greater than or equal to any of the other smells in different directions (down, left or right)</u>, THEN <u>move up</u>.
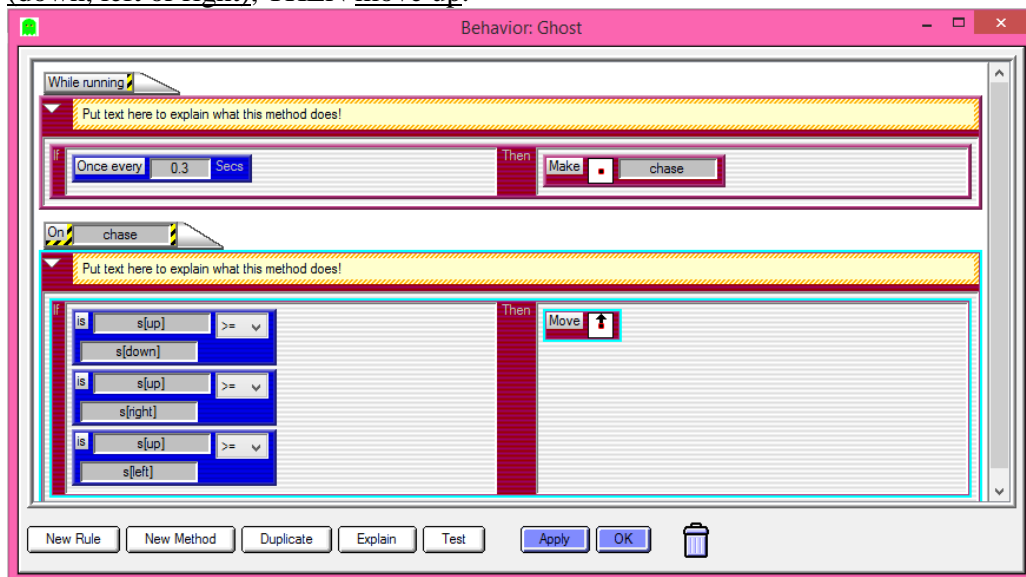
Now, add the rest of the rules so that the Ghost knows what to do if the smell down (s[down]) is greater…What if the smell to the left is greater? What about the smell to the right?

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

Check the following:
- Location of the rules
- Use of Method
- Use of parentheses and brackets

Student Handout:

Troubleshooting Guide for PacMan Part II

Diffusion and Hill Climbing

Step 1:

To determine what is happening in your game, it is sometimes helpful to look at the agent attributes. On your worksheet, click run until the ghosts move out of the box, and then click stop. Do not reset at this point. Your PacMan has now emitted his scent. You can see his scent (the value of s) by clicking anywhere on the ground and then on clicking on Tools>>Agent Attributes.
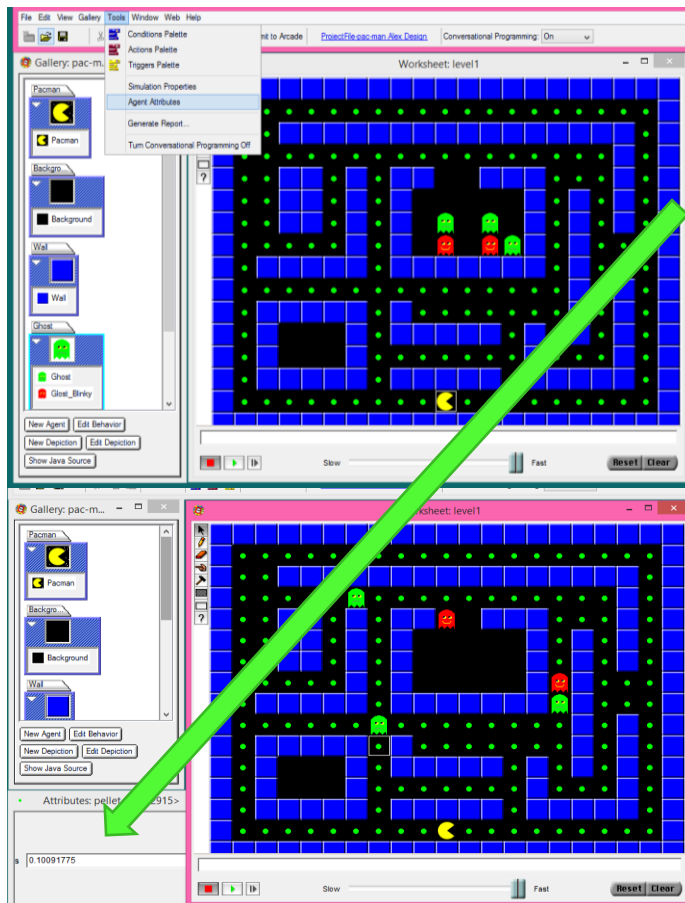
A box will appear that lists the attribute value for that agent. You can see it in the box below.

In this box is the value of the scent to the left of this Ghost. By checking the attributes of the four boxes around the Ghost (up, down, left and right) and then running the game again, you can see if your Ghost is doing what you expected him to do.
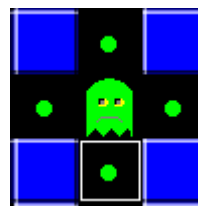
If he isn't, go back and check your rules and methods. Some things to consider:

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

PacMan

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

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

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

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

Step 1: Counting up the pellets to see if you won

Imagine this conversation…

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

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

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

That's what this programming will look like. The Controller will say, "Pellet count starts at zero" (like the classroom, no hands are up when the teacher asks who is still working). When the pellets 'hear' the Controller ask (broadcast) the question, the pellets respond back (raise their

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

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

To refer to this property, we use the symbol @. (This is similar to how we use the hashtag to tag posts, like #simulation.) Therefore, when we refer to the pellets, we type @pellets.
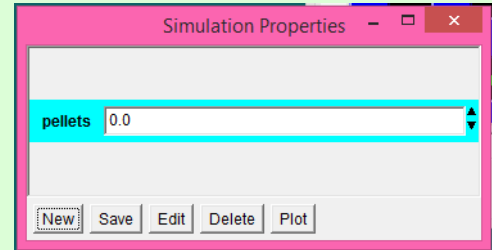
There are three parts to the Controller behavior.

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

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

Part 3: Use the count of the pellets to see if the game is done.
Make myself check-won

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

Your behavior for the PacMan or Controller should look like this:
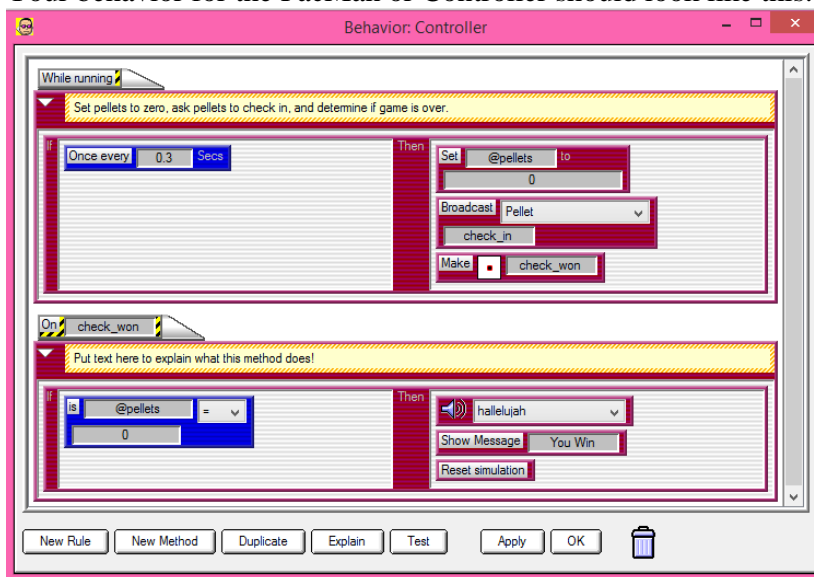
Pellet behavior changes: The controller has told the pellets to "check_in" but they don't know how to do that. We have to program them to check in when asked.

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

Helpful Tips

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

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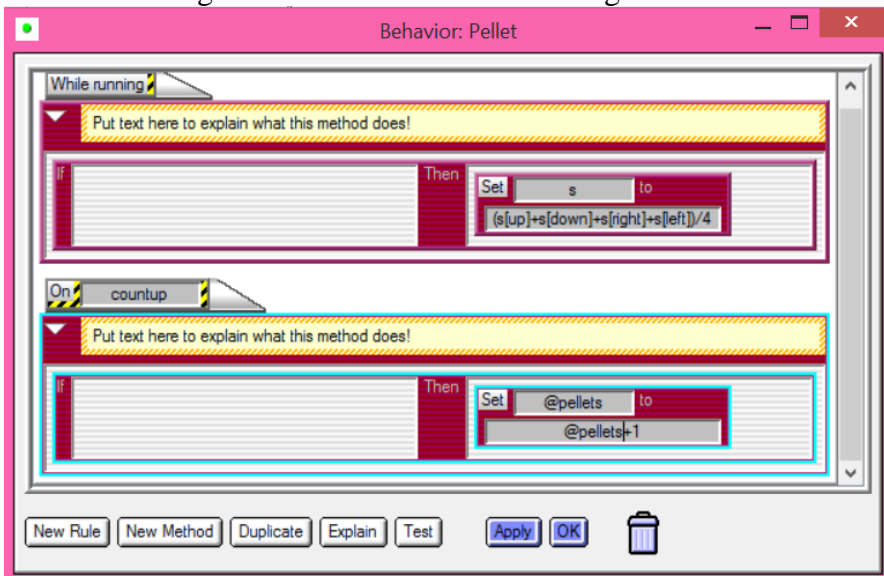
Student Handout:

Troubleshooting Guide for PacMan Part III

Polling and Broadcast

Common Problems:

1. **Is your Controller agent on the worksheet?**
2. **Did you type in @pellets where you needed to?**
3. **Do you refer to the correct agents in each step?**
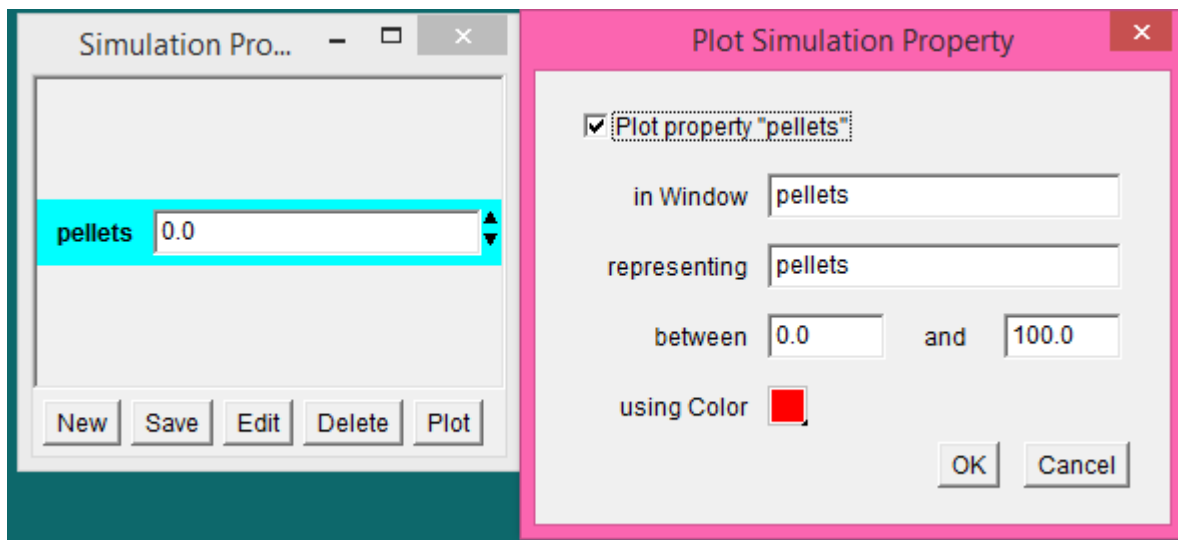
More detailed troubleshooting:

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

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

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

PacMan

End of Unit Review Sheet - PacMan

A) The main computational thinking patterns we <u>reviewed</u> were:

   1) **User Control**: intentionally moving an agent.
- a. Using keyboard keys to move an agent.
- b. Example is moving the PacMan.

   2) **Absorb**: deleting agents on the screen.
- a. Use the "Erase" action in AgentSheets.
- b. Examples are erasing the pellets.

   3) **Collision**: when 2 agents collide (run into each other).
- a. Use the "See" condition
- b. Use the "Stacked" condition, OR
- c. Use the "Next to" condition.
- d. Examples are the eating pellets and losing the game when the ghosts touch the PacMan.

B) The main NEW computational thinking patterns we <u>learned</u> were:

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

$$s = (s[left]+s[right]+s[up]+s[down])/4.$$

   2) **Hill Climbing**: following the highest scent. It only works if there is diffusion done with it, so they go hand in hand. Example is the method we created on the Ghost to follow the highest value of the scent "s" around him.

   3) **Broadcasting**: is when we "shout out" to all agents of a certain type requesting them to execute a specific method.

- a. Use the "broadcast" action in AgentSheets.

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

C) Other concepts we covered in AgentSheets are:

   1) Troubleshooting the simulation, and considering rule order.

   2) Using sounds and messages in the game.

   3) Timing our actions using the "Once every" condition.

PacMan

Student Handout 4a:

PacMan Changes Direction

Challenge

Before your start this challenge:

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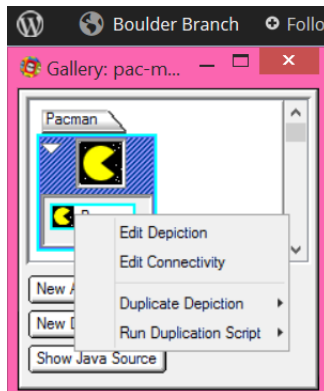
Change Directions

Make the PacMan face the direction he's heading

Description of the Challenge:

- PacMan will turn in the direction he's heading.

What to consider:
    Do you need a new agent?
    Do you need a new rule?

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

Right click on the depiction of the original agent. Check out the options under "duplicate depiction" and "run duplication script" to figure out how to create three other directions of the PacMan. You may want to rename them to help you stay organized. When you have done it correctly, it will look like this →.
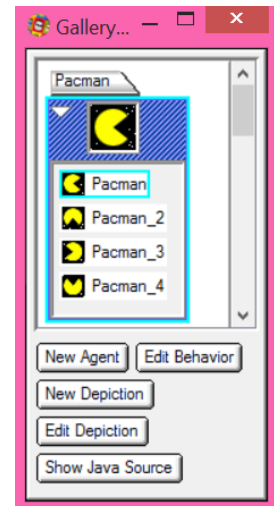
Once you have the different depictions, you will want to change your code so that WHEN THE PACMAN MOVES, it CHANGES into the new DEPICTION.

TEST your program to confirm that the depiction changes when the PacMan changes directions.

Student Handout 4b:

PacMan Moves Continuously

Challenge

Before your start this challenge:

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

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

Description of the Challenge:

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

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

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

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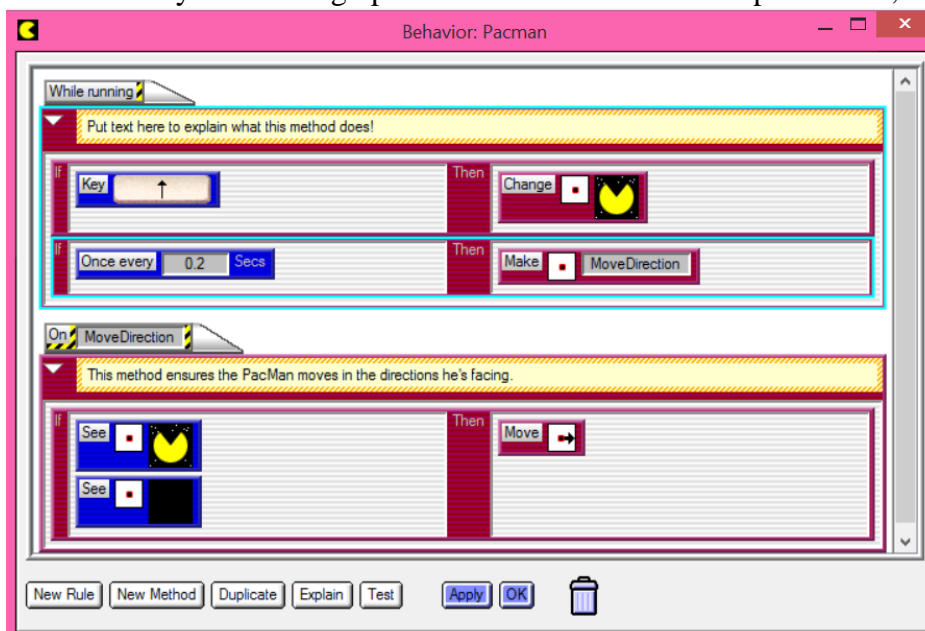
There is still much to code:

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

Step 2:
Create code for all the other directions.

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

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

The method MoveDirection is red, which means that either/both are true. The PacMan does not see the Up Depiction AND/OR does not see the floor, making the rule FALSE.
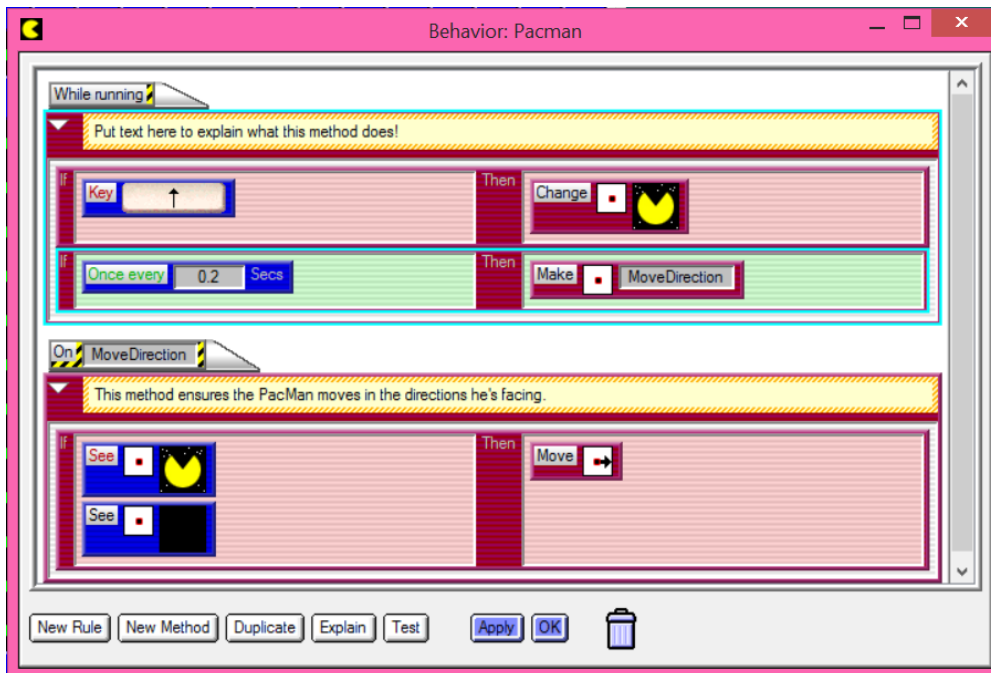
Student Handout 4c:

Power Pellet

Challenge

Before your start this challenge:

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

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

Description of the Challenge:

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

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

To help you think this through…

- You will need a new agent (Power Pellet)

- Do you need a new agent for the blue ghost?

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

- Hint: Use the **hill climbing** action rather than all the code for sniffing.
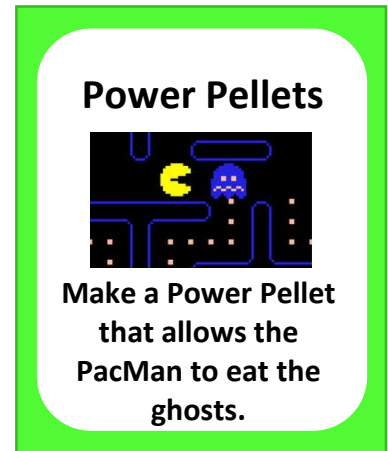
Student Handout 4d:

Next Level

Challenge

Before your start this challenge:

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

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

Next Level

Next Level!

Make a second level for your PacMan!

Description of the Challenge:

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

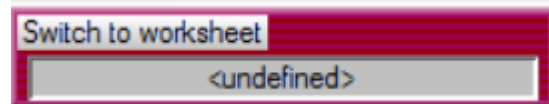This challenge gets you started, but won't give you all the code.

To help you think this through…

- Do you need a new agent? A new worksheet?
- When would a new level appear?
- What code needs to change to make the new level appear?
  You might have code like this:

Then
Show Message | You Win! Great .
Wait | 0.5
Reset simulation
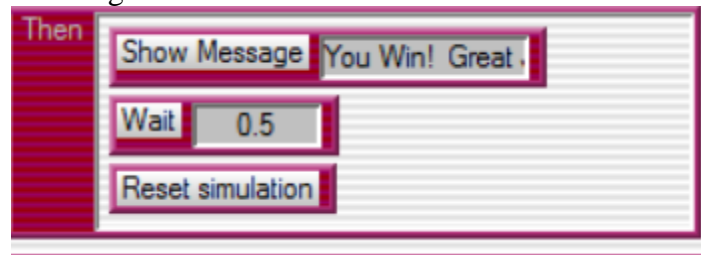
How could you incorporate this action instead?

Switch to worksheet
<undefined>

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