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: Jeffrey Bush and Cathy Brand
University of Colorado, School of Education

This curriculum has been designed as part of the Scalable Games Design project. It was created using portions of prior work completed by Susan Miller

This material is based upon work supported by the National Science Foundation under Grant No. DRL-1312129 and CNS-1138526. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.
PacMan

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 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-predictable 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.
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 or clone our blank game with the agents already designed: https://www.agentcubesonline.com/project/902095

Turn to your partner and discuss:
- Do you want ground under the walls?
- Why would it matter?
- What could go wrong?

Create the following BEHAVIORS for your agents:

Step 1: Ghost:
Program the Ghost to move randomly on the ground and the pellets. Here is the first rule that allows the ghost to move on the ground. What should the second rule look like? Be mindful of what will happen if both commands have the same time for “once every”. (See Appendix I: Guidance on Ghost’s Random Movement for more help. It is found on the last page of your lesson.)
Step 2: PacMan:
- Make four rules so your agent moves in the right directions when each arrow key is typed just like the frog in Frogger (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. First come up with an if…then sentence to describe the behavior, then write/test the code.

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

Scaffolding Tip: if they need help, encourage students to use if…and…then… sentences

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 run into 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 Online 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 cannot click on the stop game button before the dialog box reappears, you must shut down and restart your computer and then restart AgentCubes Online.

Step 5: Test your game
PacMan (Continued)

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?
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.

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: The best way to initialize PacMan’s S agent attribute is to set it when PacMan is drawn on the world because then PacMan’s attribute will always start at the same value.

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. Double click on PacMan with the big arrow tool.

![Attributes window]

2. You should see this window appear:

3. If S is not visible in the window, it did not get set to a value yet.
4. Erase and redraw PacMan and then save the world. Then 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 using the pellet agents**

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:

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

**Why do we multiply by 0.25?**

When you find the average of a set of numbers, you add them up and divide by the number of numbers.

In this case, dividing by 4 is the same as multiplying by \( \frac{1}{4} \) which equals 0.25.
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.

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 three more rules to the Chase Pacman method 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:

- 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.
More realistic Ghost catching
Have you noticed how your Ghost will “catch” PacMan when it is next to him, or even diagonal from him. How could you now change your lose condition for when the ghosts catch PacMan to make the Ghosts only “catch” him when they are above him? Give it a try.

Why didn’t this work before you added Hill Climbing? Discuss with a partner.

Shortcut for Hill Climbing
AgentCubes Online has an even better way to handle the process of Hill Climbing. 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.

There is a single action, hill climb, that replaces all the rules in the Chase PacMan method.

It is possible to eliminate the Chase PacMan method by simply putting the hill climb action in the rule in the ghost’s while running method.
Test out the options in the hill climb action.
What happens if the ghosts search in 8 directions?
Can PacMan escape?
Would it help to use fewer ghosts?
After you have run your game several times, choose whether your ghosts will search in 4 or 8 directions and decide on the number of ghosts in your game.
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. Running the game briefly made PacMan’s scent diffuse across the world.
3. Check PacMan’s s attribute by double clicking on him with the big arrow tool.

4. This window will appear:
5. Make sure that you erased and redrew PacMan on the world and then SAVED the world.
6. To see how PacMan’s scent (the value of S) has diffused, double click with the big arrow tool anywhere on the ground or on a pellet.
7. A window will appear that lists the S attribute value for that agent.
8. Click around the world. Is S biggest close to PacMan and smaller far away from PacMan? Before you do so, MAKE SURE YOU TEMPORARILY REMOVE THE “once every 0.5” seconds” so you get more immediate feedback.

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?
- If you used the hill climb action, check that you changed “value” in the hill climb action to S.
Now put the “once every 0.5 seconds” back in so the ghosts don’t move too fast

9. You can also try erasing all but one of the ghosts to see if it moves towards PacMan. This helps because the Ghosts do not propagate the scent.

Critical thinking question: Why don’t you want the Ghosts to propagate the scent? What would happen if they do?

Part 2: Using a plot action to visualize S values:
We 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 in a 3D surface above the world. The peak on the plot represents the highest value of S in the world. What agent should be under the peak?
Make a new rule at the bottom of PacMan’s while-running method and add the plot action:

To make the plot action work
1. This rule must be the last rule in PacMan’s while-running method!
2. Enter “S” for the name of the agent attribute.
3. Edit the row and column numbers so that values from the entire world are plotted.
   Remember that the world size for PacMan is 16 rows and 21 columns. Computer scientists always count from 0 to (Number – 1), in this case, rows 0 to 15 and columns 0 to 20. If you changed the default size, you must count from 0 to (New Size -1).
4. Pick a color for the plot that will show up against your background.
5. Change the elevation to 1.0.
6. 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.
7. Make a test world that just has a layer of ground agents, a layer of pellets, PacMan and one ghost.
8. Use the Rotate tool to tilt your world so that it is nearly horizontal and you can see the agents below the plot surface.

9. 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).

10. Add a couple of rows of wall agents. What happens to the shape of the plot when the walls are added?

11. The walls have no S value so the plot gets some wrinkles and valleys.

12. Watch the ghost go around the walls following the increasing S values towards PacMan!

13. Now run your PacMan world with the complete maze on it and see what happens to the plot of the S values!

14. You may find that reloading the world does not erase the plot surface. Reload your browser window or click on the AgentCubes Online logo in the upper right corner of the AgentCubes Online window and click the edit button for your project again.

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?
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).

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

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 and SAVE the 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”)
   \[ \text{Set } \text{@Pellets to zero} \]
2. Ask the pellets if they are still on the world
   \[ \text{Broadcast to Pellet agents to do “Count”} \]
3. Check the number of pellets to see if the game is done.
   \[ \text{Send a Checkwin message to myself.} \]

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.

How do Simulation Properties Work?

1. Simulation properties may be added, saved or deleted in the Simulation Properties window.
2. Open this window by clicking on the gear button on the top right side of the AgentCubes Online window and choosing “Show Simulation Properties”.
3. Make the Pellets simulation property by clicking on the + button at the bottom of the simulation properties window and typing the name “Pellets”.
4. Click on the Save button so that the new simulation property is saved!
5. The value of a simulation property can be changed or checked by any of the agents in the game or world.
6. All conditions and actions which check or change the value of a simulation property must place an “@” before the simulation property name.
Try setting up these rules now!

**Helpful Tips**

*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:

![Controller behavior](attachment:image1.png)

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

![Pellet behavior](attachment:image2.png)
The Bigger Picture: Communication between Agents

Polling introduces a technique that allows agents to create a complex behavior by cooperating: a particular set of conditions cause one type of agent to send a message to another type of agent to do a named method that contains a special set of rules. Count was the special method in the polling example.

This type of communication between different types of agents can be used to create interesting games. For example, if PacMan eats a power pill, then PacMan can broadcast a message to all ghosts to “Get_Scared”. The Get_Scared method can change the ghosts’ appearance so that they look different as they run away instead of chasing PacMan. Or the Traveler in Journey can fire ice arrows at Chasers. When an ice arrow hits a Chaser, it sends the Chaser agent a message that makes it freeze if it is unfrozen. Frozen chasers cannot move so the Traveler can collect the treasures without being caught by the Chaser.

However, when the goal is just to count up the number of agents and stick the value in a simulation property, AgentCubes Online has a simpler method for the controller to do this:

These two actions have been deleted from the controller’s while running method:

```
set @Pellets to 0
broadcast Pellet count
```

And replaced by this action:

```
set @Pellets to agents_of_type("Pellet")
```

The set action contains a specialized communication between the controller and the Pellet agents, the `agents_of_type("Pellet")` message, which makes the pellets count themselves without the need for us to code a separate count method.

You learned polling so that you would understand how to make different types of agents communicate. But there is usually more than one solution to a programming problem so now you have seen an alternate way to keep track of the number of any kind of agent.
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:

Click on the gear button on the upper right side of the AgentCubes Online window and select “Show simulation Properties”. This window will appear:

![Simulation Properties](image)

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!**

The Pellets Plot window will appear as soon as the run button is clicked. 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.
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 Online.
      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,

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

   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 Online.
      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 Online 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:

**Challenge 1: PacMan Changes Direction**

**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.

*Keep in mind there are multiple ways to solve this. Before reading ahead, try solving this challenge your own way. Do you need new/different rules? New agents/shapes?*

**Option 1:** Each PacMan follows the same rules so you need 3 new shapes instead of a new agent

**Steps:**

- Select PacMan by clicking on him, then click on +Shape
- Clear the picture and draw PacMan facing a different direction.
- 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.

**Option 2:** Instead of making new shapes, use the “rotate to” command

**Steps:**

- Edit the move behavior of PacMan to include a “rotate to” command
- The first of the three numbers in “rotate to” is the rotation you want, a rotation of 180 degrees will cause PacMan to turn in the opposite direction.

Once you are done, TEST your program to confirm that the PacMan’s shape changes when he changes directions as he moves.
Student Handout 4b:

Challenge 2: PacMan Moves Continuously

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 “move continuously”.

When the move continuously 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 move continuously 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 see 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:

Challenge 3: Power Pellet

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:

Challenge 4: Next Level

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!
Appendix 1: Guidance on Ghost’s Random Movement

Making the ghosts move randomly on both pellets and ground is a challenge and your students will come up with many different solutions that do not work.

Your students may make rules that look like these two examples:

What happens with the rules above if there are pellets on the ground? Can the ghost move?

What happens in this case if the ghost is on the ground? Can the ghost move?

It helps to understand how timer events work. A “once every” condition can be interpreted like this: “Has this amount of time passed since the last time this condition was true?” Once the set amount of time has passed, the rule will remain true until the “then” condition (move random in this case) has been preformed. Once that happens, the condition is no longer true and it goes back to the start of the method. This means it will never get to the second “once every” condition.

Both of these sets of rules will work correctly and allow Pac-Man to move on pellets and ground if the once-every condition has a different time for the second rule