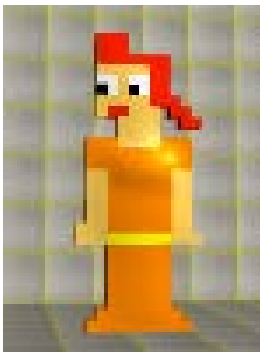


Being in the Game: Implementing First Person

Adding Challenge to a game like Journey or PacMan by switching viewpoints from bird's eye to first person so that the player only sees what the Traveler or PacMan sees rather than the entire game world.

Created by: Catharine Brand and Susan Miller, University of Colorado

This curriculum has been designed as part of the Scalable Games Design project.
It uses material from Fred Gluck's video tutorials and Susan Miller's SGD curricula.

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

Lesson Objective:

- **To add rules that let the player see the game from the point of view of the key-controlled character.**

Prerequisite Skills:

- **Students are presumed to have the following skills. Return to the Frogger Lesson Plans for detailed explanations on these skills.**
  - **Create agents**
  - **Basic agent behavior including:**
    - **Key-controlled movement**
    - **Random movement**
    - **Ending the game**

Computational Thinking Patterns:

- **First Person**

Useful links:

first person video tutorial:
http://sgd.cs.colorado.edu/wiki/First_Person_Navigation
introduction to modulo math operation:
http://betterexplained.com/articles/fun-with-modular-arithmetic/

Activity Description: divide over 2 to 3 classes

- **Part 1: Create a basic world with a key-controlled ladybug and 4 walls.**
- **Part 2: Learn how to name directions in AgentCubes. Initialize the ladybug's Direction attribute and modify the basic movement rules so that the ladybug updates her Direction attribute to match the direction she is moving and then turns to face the same direction.**
- **Part 3: Explore why movement should be different in first person: up arrow means move forward and other arrows turn agent. Learn what mod means and why it is important for first person movement.**
- **Part 4: Add the first person movement rules.**
- **Part 5: Test the first person movement rules.**
- **Part 6: Copy and paste first person rules into the Traveler or Pacman.**

Table of Contents

Vocabulary/Definitions

Agent Attribute .....a value assigned to an agent (such as scent)

Bird's Eye View..... looking down on a World as if you were flying over it

First Person View…. seeing a world from the point of view of a single agent

Local Variable .....a variable (attribute) belonging to a specific agent

Mod or Modulo ……this mathematical operation returns the remainder when one number is divided by another

General Teaching Strategies[1]

Basic Philosophy

- The educational goal of these lessons is to learn and apply Computational Thinking Patterns in the context of a familiar game. Emphasis on these Computational Thinking Patterns is essential for student understanding.
- These lessons are also designed to give students positive experiences with and perceptions of computer science. Research shows that students turn away from high school and college computer science courses if they perceive it as boring, unrelated to what matters to them, and hard. We hope to change that by providing a fun, relevant and accessible computer science experience where they can personalize their experience to make computer science about them.
- Guided discovery is the central tenant of our curriculum. Direct instruction is sometimes used for aspects where students are learning the code for the first time; however, materials have been provided to ensure that students understand the programming concepts, as opposed to simply copying code.
- Whenever possible, students should try to come up with the steps on their own or in small groups, when differentiation and more structure is needed, we have more structured materials available.

[1] This information is supported by research found in the following documents:

Basawapatna, A. R., Koh, K. H., & Repenning, A. (2010, June). Using scalable game design to teach computer science from middle school to graduate school. In Proceedings of the fifteenth annual conference on Innovation and technology in computer science education (pp. 224-228). ACM.

Google. (2014). Women Who Choose Computer Science — What Really Matters The Critical Role and Exposure. Retrieved from https://www.google.com/edu/resources/computerscience/research/

National Research Council. (2011). *Learning science through computer games and simulations*. (M. Hilton & M. Honey, Eds.). Washington, DC: The National Academies Press.

National Research Council. (2014). *STEM Integration in K-12 Education:: Status, Prospects, and an Agenda for Research*. (M. Honey, G. Pearson, & H. Schweingruber, Eds.). Washington, DC: The National Academies Press.

Repenning, A., & Ioannidou, A. (2008, March). Broadening participation through scalable game design. In *ACM SIGCSE Bulletin* (Vol. 40, No. 1, pp. 305-309). ACM.

Taylor, H. G., & Mounfield, L. C. (1994). Exploration of the Relationship between Prior Computing Experience and Gender on Success in College Computer Science. *Journal of Educational Computing Research*, *11*(4), 291–306.

- Student materials are available for each portion of the game design. These materials are intended to be used in addition to teacher materials, which provide prompts and discussion points.

- Students may become frustrated with too little teacher support, THIS IS OK! A little frustration and moving at a slower pace is well worth the deeper conceptual understanding that comes with guided discovery.

Guided Discovery Process

- **Model the process** rather than just giving students the answer. As a teacher, focus on explanations and discussions of WHY something works or doesn't work and let the students figure out HOW to make it work.

- Building the game on your own, before trying it with your class will enable you to see which steps may challenge or confuse your students.

- Have students work through problems independently or in small groups. Ask directing questions or give helpful suggestions, but **provide only minimal assistance** and only when needed to overcome obstacles.

- **Group work is your friend**! It is common for computer programmers to talk through problems with one another, and to use code snippets found from other programs and other programmers. Talking through coding problems enables students to think more critically about Computational Thinking Patterns, as well as the steps needed to solve a problem.

- Additionally, seeing how others solved an issue with code helps students realize that problems often have multiple solution strategies, and that some solutions might be more effective than others. Also group work lets them see that they are not alone and that others have similar and different questions, struggles, inspirations and perspectives.

- Recognize that programming is largely a process of **trial and error**, particularly when students are first learning. It is helpful to encourage this mindset with your students.

- "What have you tried so far? Why didn't it work?" is a great way to start any troubleshooting discussion.

Building Blocks

- Each project is designed to build on the prior one. Very little student support is provided where expertise has already been created. Conversely, material that is new has more support.
- Be sure to talk through the building blocks (especially for PacMan in the area of diffusion and hill climbing) as these Computational Thinking Patterns will appear often in future games and simulations.
- Encourage discussion and reflection on these Computational Thinking patterns. Small group or whole class discussion relating Computational Thinking patters to the outside world can be super productive.
- Remember that conceptual understanding takes time, and it may be necessary to review these concepts multiple times, using different examples, so that all students can be successful.

Support Learning

- Research shows that game design is associated with engaged students, and engaged students show higher levels on conceptual understanding. Allowing students to personalize their games aids in this engagement and motivation. Plus, it makes grading and reviewing games more fun for you.
- Coding may be difficult for some students, and all students are likely to be frustrated at times when the code does not produce the expected results. **Praise students** for sticking with the troubleshooting process and encourage them to share what they learned with others.
- Consider students who are ahead to the role of “code ambassadors” to walk around and help their peers with coding questions.
- Be sure to communicate that **the process is more important than the answer**, and that coding of a project often takes time. Do not place pressure on your students to ‘hurry up’ and resort to giving them the code. The process of figuring it out on his/her own will result in much stronger conceptual understanding.

Differentiated Instruction

Note that there are many vocabulary words in this lesson that may be new for your students. Take time to define those words. Using the words in context often will reinforce their meaning for the students.

- **Students who need a challenge:** Some students with more fluency in programming may finish this very quickly – be prepared for by having challenge activity materials ready in advance.

- **Students who need more assistance:** Other students may struggle a bit more.
    - This lesson provides an opportunity to review how degrees match positions on the unit circle and the meaning of the modulo operation.
    - Pairing the student with an experienced student should alleviate many problems.
    - Vocabulary for ELL Students: bird's eye view, first person view, navigation, modulo, rotate

Teacher Instructions: Introduction

This lesson style is different than many of the other lessons within Scalable Game Design. Students will learn to change their game using first person navigation. This is a rather tricky concept, and one that is not likely to be discovered by students. The lesson will begin with some activities by students to help build conceptual understanding, but will move to providing the students with the necessary code. We will model good practices around using pre-existing code, and figuring out what the code does.

Background:

Up to this point, we have played games in **bird's eye view**, looking down at the game world from above. From the bird's eye view, it is easy to see how to get around obstacles, and where the dangers and the rewards of a game are located. Below is a journey world in bird's eye view.

Gamers usually play games in first person because it can be exciting and challenging to make their way through the game world, seeing only what their character would see. Instead of looking down on an entire maze, the player sees only the walls that surround his character. Chasers jump out suddenly from behind a wall. A treasure appears unexpectedly as the gamer turns the corner. Below is a snapshot of the same world with all the agents in the same positions in **first person view**, looking at a Chaser in the maze over the Traveler's head. Note that the direction which appears to be "up" in first person is actually "down" in bird's eye view!

While it is easy to switch into the first person point of view by selecting an agent with the big arrow tool and clicking on the camera button  located on the top bar of the AgentCubes window, the simple movement rules with arrow keys do not work as expected.

Have students try this out. Open Journey, PacMan or a similar game. Have students switch to First Person View by clicking on the main character, and then on the camera button. What do they see? Probe them to see why they think this is happening. What changes do they think need to be made to fix the problem?

In these simple rules, the up arrow key always moves the character towards the top of the AgentCubes window although the direction that appears to be "up" in first person view may actually be the left, right or lower side of the AgentCubes window. The right and left arrow keys make the agent move sideways like a crab and the down key makes the agent move backwards. It would be more realistic if the left, right and down arrow keys turned the agent while the up arrow key moved the agent forward in whichever way it is facing. In order to make the agent's movements match what the player sees from the first person perspective, we must create new movement rules for our keyboard-controlled agent when the first person button has been clicked.

Have students try this out. Hand out the student worksheet (Student handout A, found on page 3 of Student Handbook). Students can either modify an existing game or create a new one. The students will create a new world that enables them to easily track the direction they are heading, and will learn how to create rules to move in first person view.

Note: Much of this lesson is done without student handouts. The teacher's role will be to prompt students to think about how to correctly maneuver in First Person, as well as to determine why the given code works as needed.

STUDENT Handout A:

Create a world for First Person Navigation

Creating a simple world for exploring first person movement

You will create a world to learn how First Person Navigation works in AgentCubes Online. To do this, you will create a new world that is occupied by a ladybug. We will add the first person rules to a ladybug because it has a head end, which makes it much easier to see that it turns and moves correctly in response to each arrow key. Testing our rules is easier if we work in a simplified world where we can instantly see which way our agent is facing. You can choose to add these rules to an existing game that you want to change into First Person, or create a brand new game as an example.

- Make a new level called First Person in the game in which you would like to use the first person perspective.
- Select PacMan or the Traveler or Frogger or any main character who is keyboard-controlled with the big arrow tool, click on the +Shape button to make a 2nd shape and name it ladybug.
    - Choose Inflatable Icon, the Animals category and then Akako, the ladybug.
- **Or** create a new agent named Ladybug and choose the Akako shape.

Create the following 6 agents:

1. An agent with a clear front and back (we strongly encourage you to use Ladybug and choose the Akako shape)
2. a tile named “Ground”
3. Select a cube for the north wall by choosing the Letters-Bit font category and then picking n.png.
4. Select a cube for the west wall by choosing the Letters-Bit font category and then picking w.png.
5. Select a cube for the south wall by choosing the Letters-Bit font category and then picking s.png.
6. Select a cube for the east wall by choosing the Letters-Bit font category and then picking e.png.

Create your first person world:

1. Cover it with a layer of Ground tiles.
2. Put the Ladybug in the center of the World with her head facing the top of the World.
3. Draw a line of north blocks across the top of the World.
4. Draw a line of west blocks down the left side of the World.
5. Draw a line of south blocks across the bottom of the World.
6. Draw a line of east blocks up the right side of the World.

Your First Person World should look like this:

Exploring first person movement with bird's eye perspective rules

- Set up the Ladybug with the basic key-controlled movement rules so the 4 arrow key rules move it up, down, left, and right.

- Select the Ladybug with the big arrow key, then click on the camera to switch to first person. Use the Rotate  , pan  and zoom  tools to adjust the camera so that it is looking over the top of the agent's head, as if the agent were wearing a head-mounted GoPro camera. This lets the viewer see the world from the agent's point of view.

Here is what the first person world should look like when the Ladybug is facing up and the camera has been set to look over her head:

Run the game. In what ways is the movement realistic. In what ways is it not?

Teacher Instructions: Rotate the Agents

Have students run their game. Students will find that the basic move rules do not let the Ladybug move in a realistic way.

Ask the students:

Describe what the Ladybug is doing.

They might tell you that:

- The Ladybug can not turn around to reverse direction but must move backwards with no view of dangers behind it.
- The Ladybug also moves sideways like a crab rather than turning and then moving forward when the left and right arrows are typed.

Will the Ladybug will be able to move easily through the maze in Journey or Pacman in first person?

How could we make the Ladybug's movements more realistic?

The Ladybug would move in a more realistic way if it turned to face the direction it is moving.

Ask students to look at the list of basic actions to see whether there is anything useful there.

- They should notice the **rotate to** and **rotate by** actions.

These actions, 

and 

, require directions expressed as degrees on a circle.

At this point, suggest to students that they imagine skiers doing tricks. What does it mean to 'Do a 180'? What does it mean to do a 360? It's often helpful to have students stand up and (safely!) try this out. What would it look like to 'do a 90'? If I 'do a 180', does it matter which direction I start in? How can I be certain that we will all end up going the same direction? Be sure students understand the idea of degrees in a circle before moving on!

Which action would make it easy to turn the Ladybug to face in the direction of movement?

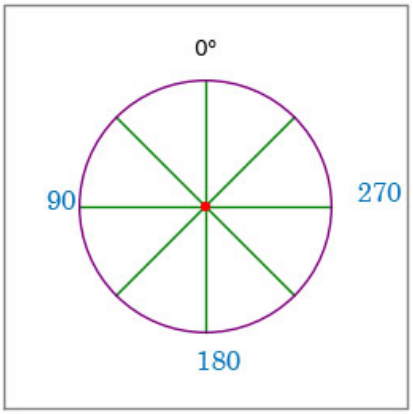
Students will likely suggest that the "Rotate To" or "Rotate By" action will rotate the agent. Have students add the Rotate To action to the arrow move rules, and provide time for students to figure out how to create rules that produce the desired results. Have students add the Rotate By action to the arrow move rules, and provide time for students to figure out how to create rules that produce the desired results. Note that the students will have to figure out:

1. The difference between Rotate By and Rotate To
2. Which number rotates the agent in the correct manner 

3. The proper order of the rules.

Here are the 4 basic movement rules with the rotate to action added. **Note**: the Ladybug rotates then moves because it looks more realistic!

Have students create their own circle and identify the location of 0, 90, 180, and 270 degrees.



Directions for rotate to and rotate by actions:
0 degrees is **North or up**
90 degrees is **West or left**
180 degrees is **South or down**
270 degrees is **East or right**

Testing First Person with the New Bird's Eye Move Rules

Have students test the Ladybug's movement in **both** bird's eye and first person.

Ask students to explain what is working well, and what is not with their new code.

Students are likely to notice that although it looks as if the Ladybug should be moving straight forward, the player must keep typing the right arrow to keep the agent moving right.

Ask the students what would make movement in first person more realistic?
How do people move?

Usually we turn and then walk forward.

How do we want our agents to move?

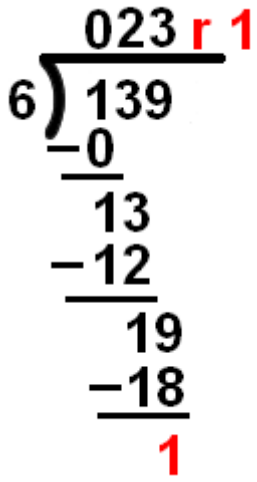
Ideally, the left, right and down arrows would turn the agent, while the up arrow would move the agent forward in whatever direction it faces. Then it would be much easier to move the agent through a maze and around obstacles rapidly.

Teacher Instructions: Using Modulo

Tell students to imagine that every time I click the ➔, my agent turns 90 degrees to the right. If I click the ➔ four times in a row, which direction is he facing? What if I click it 10 times in a row? What if I click it 15 times in a row? Continue to try more numbers until they find a pattern. They should recognize that every four clicks is 360 degrees, so they only need the number of clicks past the multiple of four. This begins a conceptual understanding of modulo.

Discuss this with your class:
Definition: Modulo is the <u>remainder</u> when dividing

Example:
Look at the example to the right. For 139÷6, the answer is 23, and the remainder is 1.



If I were to use the modulo, or 'mod' operation, it would look like this:
<u>139 mod 6</u> would return a value of 1

Let's look at another example. What is the answer for this problem?

<u>5 mod 3</u>

Remember, this means 'tell me the remainder when I divide 5 by 3.

That's right, 2 is the result of the modulo operation.

We use mod every day when we look at a clock, which has hours marked from 1 to 12.

Try these problems out with your class:
What time will the flight will depart if you get to the airport at 10 am and have to wait 4 hours for a flight? 10 + 4 = 14 but standard clock time does not go past 12 so we do 14 mod 12 and know that the flight will take off at 2 pm.

If I leave for a road trip at 10am, and I have to drive 17 hours, what time will it be?

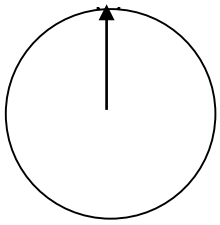
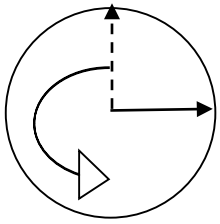
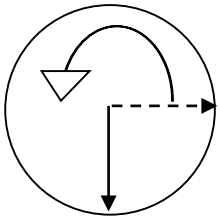
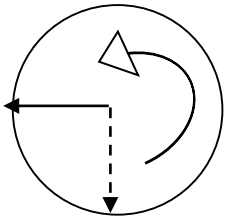
I woke up at 2pm, but I should have gotten up 6 hours ago? What time was I supposed to wake up?

Going back to our direction, remind students that the 'rotate to' button tells the ladybug to move to a specific number of degrees. So, if the → button is pushed three times, it adds 270 degrees to the current direction three times.

Have students try this out. Hand out the student handout B (found on page 5 of Student Handbook). Talk through the first example together. Then, have students work in pairs to solve the other three problems.

Student Handout B: Connecting Ladybug Movement to Modulo Math

Let's say the Ladybug is facing up (0°), and the player clicks the → button (270 degrees) three times, which way is the ladybug facing?

Starts here	Rotates 270°	Rotates 270°	Rotates 270°
Starts at 0°	+ 270°	+ 270°	+ 270° = 810°

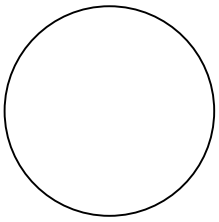
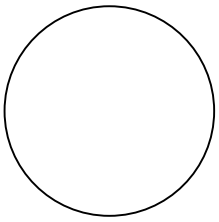
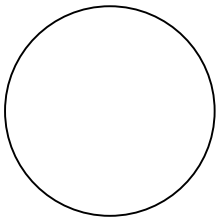
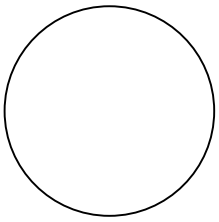
According to the math, the ladybug travels to 810°, but if we look at the picture, we can see the ladybug is facing 90°.

Mathematically, we can show that also, using the mod function

810 ÷ 360 = 2, Remainder 90

90 is the ending direction of the lady bug.

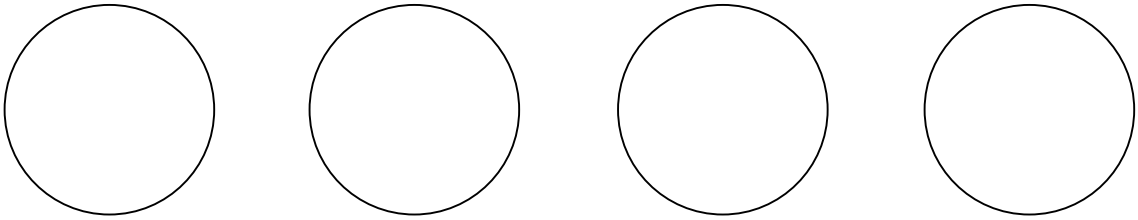
If my ladybug is facing down (180°), and the player clicks the ← button (90 degrees) three times, which way is the ladybug facing?

Starts here	Rotates _____°	Rotates _____°	Rotates _____°
Starts at 180°	+ _____°	+ _____°	+ _____° = _____°

_____÷ 360 = ____, Remainder _____ Does your remainder match your picture?

If my ladybug is facing up (0°), and the player clicks the 🡻 button (180 degrees) two times, which way is the ladybug facing?

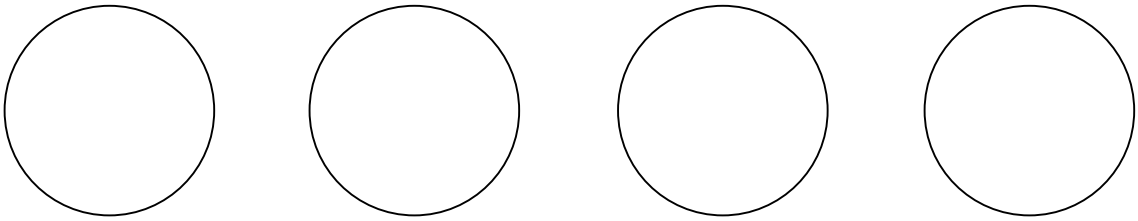


Starts here Rotates _____° Rotates _____° Rotates _____°

Starts at 0° + _____° + _____° + _____° = _____°

_____÷ 360 = ____, Remainder _____ Does your remainder match your picture?

If my ladybug is facing left (90°), and the player clicks the ➔ button (270 degrees) three times, which way is the ladybug facing?



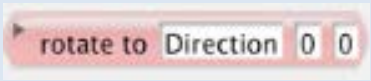
Starts here Rotates _____° Rotates _____° Rotates _____°

Starts at 90° + _____° + _____° + _____° = _____°

_____÷ 360 = ____, Remainder _____ Does your remainder match your picture?

We must use the mod 360 operation because the player may type one arrow key lots of times, making the Ladybug spin in circles and adding the value of the turn to Direction over and over. AgentCubes cannot rotate an agent to a direction that is 360 or larger!

Using Modulo 360 keeps the Direction Variable in the set of values {0,..., 359} so that



always turns the agent to a real position.

To use the mod operation, we would write:
Direction%360 if we want the remainder when we divide the direction by 360.

Talk with a partner. How does this change your ideas about creating rules for first person navigation?

Teacher Handout: Create the Rules for the First Person Navigation Method Using Pre-Existing Code

By this point, students know they need to rotate their agent, they want to be able to move in the direction their agent is facing by pushing the up arrow, and they likely recognize that using modulo has something to do with it. You may want to provide them some time to try and figure out how to solve this challenge on their own. If the struggle becomes unproductive, move on to this next section.

Sometimes we want to know how to do something but don't know where to start. Even with a basic understanding of what we want to accomplish (ideally, the left, right and down arrows would turn the agent, while the up arrow would move the agent forward in whatever direction it faces) and knowing something about how one might go about doing this (using modulo) may not be enough to get us started. When that happens, we might use someone else's code. Yet it's never a good idea to just insert someone else's code into your project without trying to understand it. In this portion of the lesson, we will give students the code, and then ask them questions to help them make sense of the code.

> ***Provide students with the Student Handout C (found on page 8 of Student Handbook). Have students work with a partner to answer the questions. Additional questions (italicized bold underlined) are included below to prompt deeper conceptualization. A troubleshooting guide is also provided if needed (Page 12).***

You may want to work through each step as a class, providing time for students to add the code to their game, and then working first in pairs, and later sharing with the class their thoughts about how the code works.

A full explanation of the code is provided on the remaining pages for the teacher.

Explanation for Creating First Person Move Rules

Remember that we want our game to be playable if it is in EITHER Bird's Eye OR First Person Navigation. So we will create rules that work for both situations.

Therefore, we need a separate set of move rules for first person perspective which assign different actions to the arrow keys.

In order to switch to the First Person move rules, we will put this rule **<u>above</u>** the Bird's Eye Perspective move rules in the Ladybug's while running method:

Ask students:

<u>What would happen if this rule appeared below the Bird's Eye Perspective move rules?</u>

One of the regular move rules would be true whenever an arrow key was typed before AgentCubes checked whether the game was in First Person.

This rule will only become true if it is above the regular move rules!

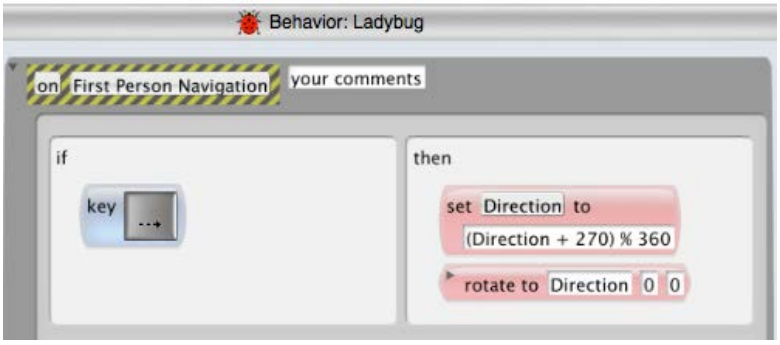
Note: Any rules that apply in both bird's eye and first person modes must be above this rule.

For example, win and lose rules must appear above this rule or else the game will never end if the player has an agent in first person.

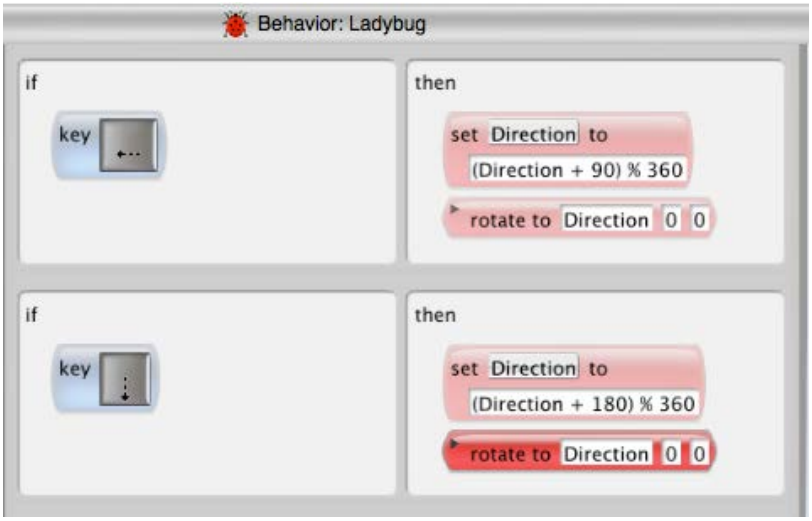
Keeping Track of the Agent's Direction

While playing a game in first person, the user can turn the agent many times using the left, right and down keys but when the up key is typed, the agent must move in the direction it faces. Since we will keep track of the current direction the agent is facing in an agent attribute (local variable) called Direction, we must make the Direction always stay between 0 and 359 degrees because the **rotate to** action will not work if the number of degrees is 360 or greater. Therefore, we must use the **modulo** mathematical operation (written %) to keep our Direction variable smaller than 360.

The rule for the right arrow in the First Person Navigation method will look like this:

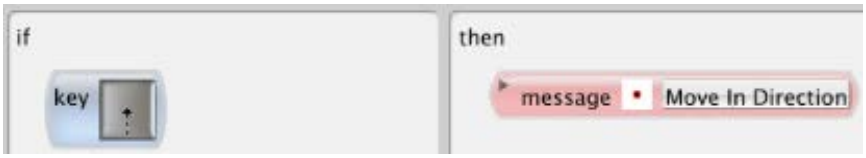


The First Person Navigation method rules for the left and down arrow keys will look very similar to the rule for the left arrow key.



The left, right and down arrow key rules calculate the new value of the Direction variable.
What does the up arrow key rule do?

In the First Person Navigation method, the up arrow key makes the agent move in whatever direction it currently has stored in its Direction variable by calling the Move in Direction method.



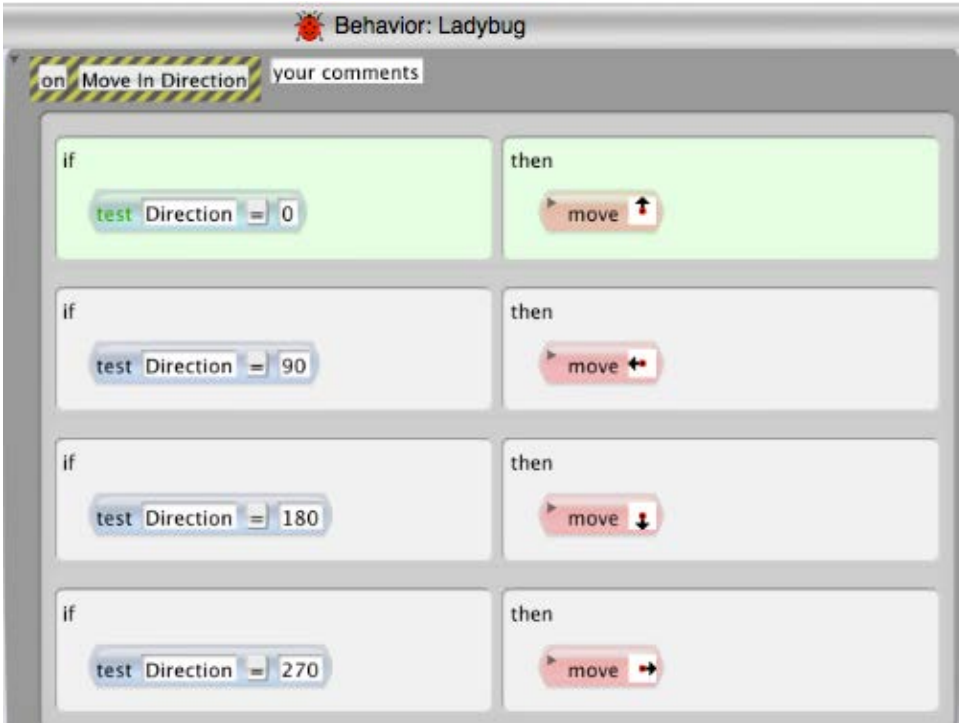
Create the Move in Direction Method

Ask your students what the rules in the Ladybug's Move in Direction Method should do?

These rules should say something like if Direction = up, move up and so on through all the directions.

Directions for rotate to and rotate by actions:
0 degrees is **North or up**
90 degrees is **West or left**
180 degrees is **South or down**
270 degrees is **East or right**

The rules must test for the numbers that stand for the different directions in AgentCubes.



Here are the rules for the Move in Direction Method.

Making Sure the Direction Agent Attribute Contains a Meaningful Value

Ask your students what will happen if the player selects the Ladybug, clicks on the First Person button, then clicks on the green play button and immediately types the up arrow.

Does the Direction attribute have a value? How does Move in Direction know which rule to call?

Actually AgentCubes sets an agent attribute to 0 if the programmer has not given it a value. In this case, if Direction = 0, then the Ladybug will move up.

What if the player had run the game for a while in bird's eye perspective using the regular move rules and then switched to first person and typed the up arrow? Some very odd things may happen in this case such as the Ladybug moving sideways or backwards. Let the students experiment.

Note: It is good practice for the programmer to set the Ladybug's Direction to a meaningful value and make sure that it always has a meaningful value.

So we should initialize the Ladybug's Direction attribute to 0 when the Ladybug is created and we should add actions to the bird's eye move rules to set the Ladybug's Direction every time the Ladybug moves.

Initializing the Ladybug's Direction Agent Attribute

We can set the Ladybug's Direction agent attribute when the Ladybug is first created and before it is moved.

Create a new method and click on the word "on" in the black and yellow striped tag in the upper left corner to change it to "when creating new agent".

This method just needs a single action to set Direction to 0.

Now erase the Ladybug off your world, draw it on the world again and **save the world!** Saving means that Direction will always be set to 0 when you switch to this world.

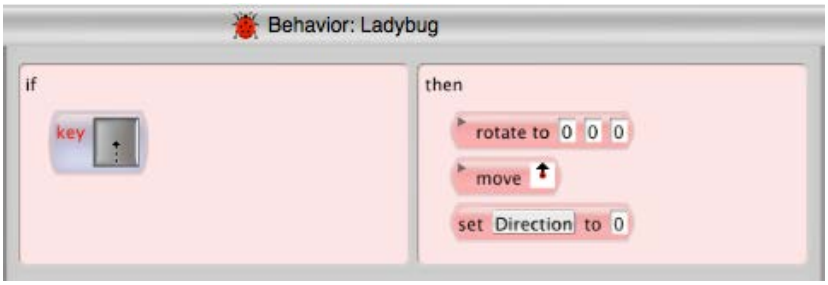
To check the value of Direction, double click on the Ladybug and the agent attribute window will open. It should tell you the value of Direction.

Setting the Direction in the Bird's Eye Move Rules

Ask the students how the regular move rules should be changed to set the Direction attribute every time the Ladybug moves.

These rules in the Ladybug's while running method just need set actions added to them.

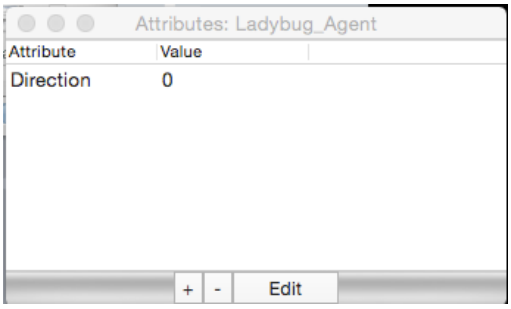
Here is the new move up rule from the Ladybug's while running method:



Here are the other three move rules.

Test Your First Person Move Rules

Use the big arrow tool to double click on the Ladybug agent.
The Agent Attribute window should appear:



Remember what the direction numbers mean:

Turn the game on and move the Ladybug around in bird's eye view and first person **one move at a time**.

Check that the value of Direction matches the way the Ladybug is facing.

Directions for rotate to and rotate by actions:
0 degrees is **North or up**
90 degrees is **West or left**
180 degrees is **South or down**
270 degrees is **East or right**

If you find a mistake, go back and check the rules for misplaced arrows or incorrect direction values.

Compare your rules to the pictures above.

Adding First Person Rules to the Traveler or PacMan

If you used an alternate shape for PacMan or Frogger, just switch back to using the regular shape for your agent.

If you made a separate ladybug agent, copy and paste each of the methods you made for the Ladybug's first person rules into the Traveler or PacMan.

- Copy and paste the rule from the Ladybug's **while running** method that tests for first person and calls the First Person Navigation method into the Traveler's or PacMan's **while running** method above the regular move rules.
- Edit the Traveler's or PacMan's basic move rules to include the rotate to and set actions so that the agent turns to face the direction it is moving and sets the Direction to match the way it is moving.

Hooray! Now you can play Journey or PacMan in First Person!

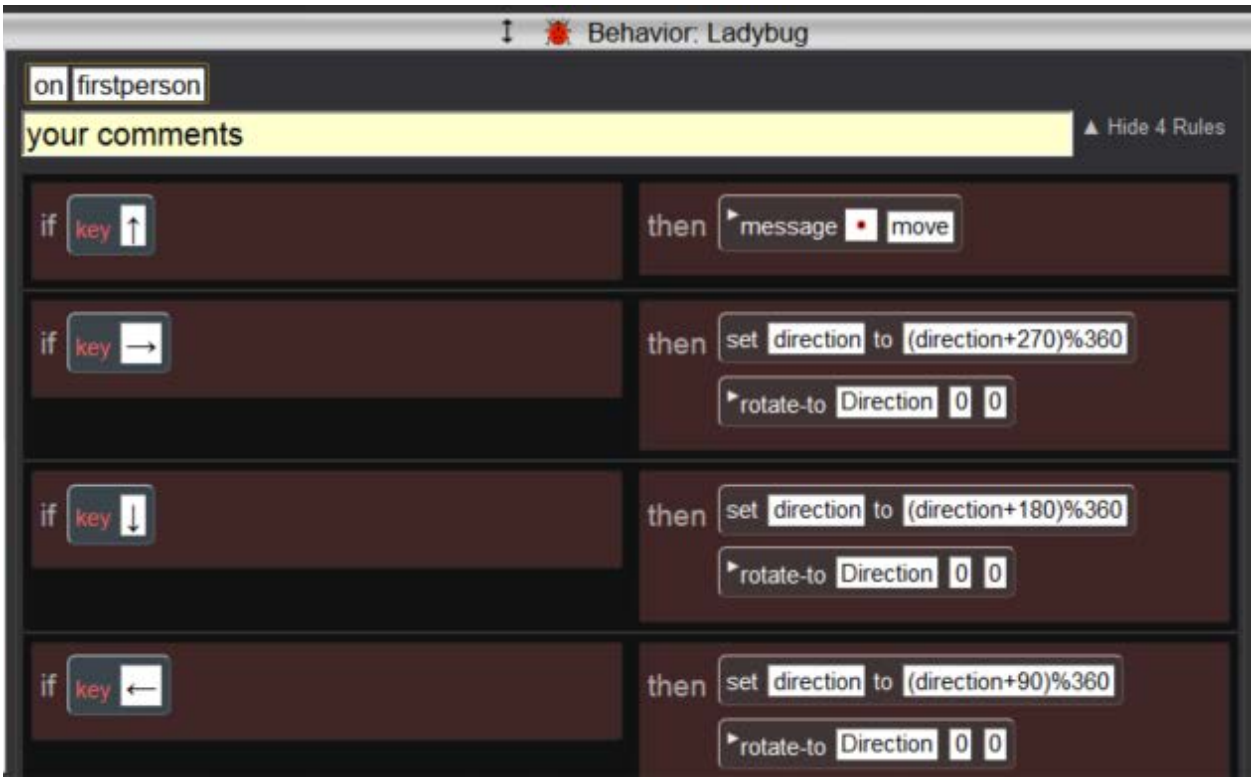
Student Handout C: Create the Rules for the First Person Navigation Method Using Pre-Existing Code

Sometimes we want to know how to do something but don't know where to start. Even with a basic understanding of what we want to accomplish (ideally, the left, right and down arrows would turn the agent, while the up arrow would move the agent forward in whatever direction it faces) and knowing something about how one might go about doing this (using modulo) may not be enough to get us started. When that happens, we might use someone else's code. Yet it's never a good idea to just insert someone else's code into your project without trying to understand it. In this portion of the lesson, we will give you the code, and then ask you to make sense of the code.

The first thing I noticed when I looked at another game that worked the way I wanted mine to work was this code. It was the first line of the 'While Running' method of the Ladybug.

Add this code to your project. What do you think this line of code will do? Why do you think it's the first line of code?

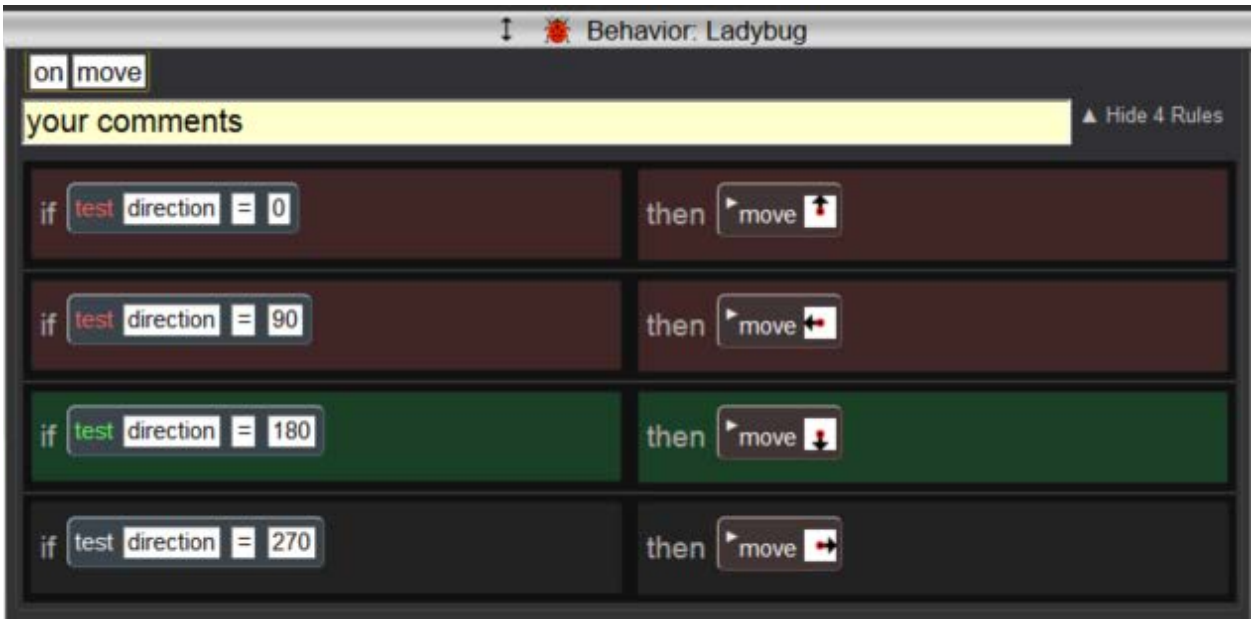
I then looked for the method that was called by that line of code. Here's what I saw.



Add this code as a new method for your ladybug. Why is the ↑ key rule different than the others?

The symbol "%" is the symbol for Modulo. What math is being done here?

This is the method 'move' that was called when the player uses the up arrow.

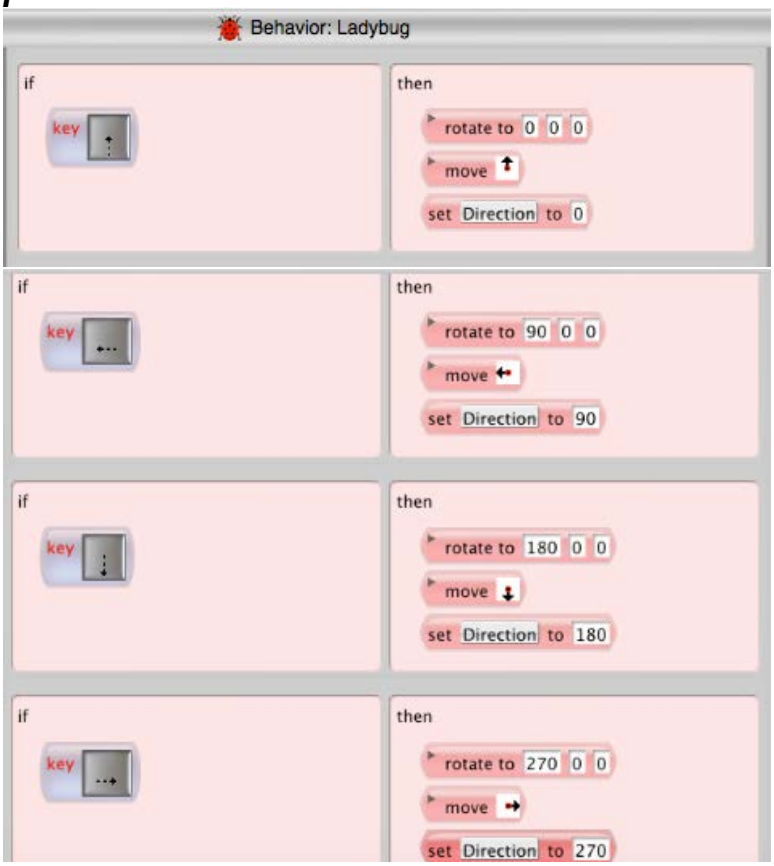


Add this code to your ladybug as a new method. Describe what's happening here.

There were some other changes to their code. I saw this added to the Ladybug.

Add this code to your ladybug as a new method. Why is this code needed when I create a new ladybug agent?

Finally, I saw that the original movement code was also changed. Here is the code from the Ladybug's 'while running' method. This code was AFTER the first rule

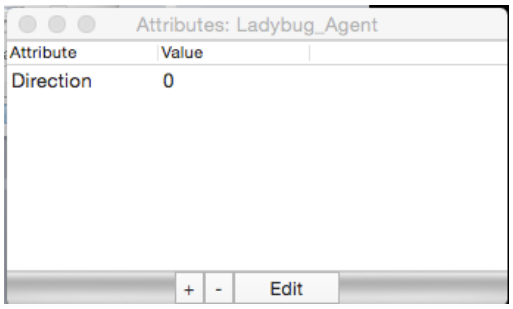


Change your arrow keys to match this code. Remember, this is in the 'While Running' method. Why is this needed? Hint: It would only be active code if the FIRST PERSON NAVIGATION rule is FALSE.

Test out your game. Does it work as expected? How can you use this to make PacMan or Journey work the same way?

Student Handout: TROUBLESHOOTING GUIDE

Test Your First Person Move Rules
Use the big arrow tool to double click on the Ladybug agent.
The Agent Attribute window should appear:



Remember what the direction numbers mean:

Turn the game on and move the Ladybug around in bird's eye view and first person **one move at a time**.

Check that the value of Direction matches the way the Ladybug is facing.

Directions for rotate to and rotate by actions:
0 degrees is **North or up**
90 degrees is **West or left**
180 degrees is **South or down**
270 degrees is **East or right**

If you find a mistake, go back and check the rules for misplaced arrows or incorrect direction values. Compare your rules to the pictures above.