



# Creating “Sokoban”

You are a warehouse keeper (Sokoban) who is in a maze. You must push boxes around the maze while trying to put them in the designated locations. Only one box may be pushed at a time, and boxes cannot be pulled. When boxes are covering all of the destinations, the level is complete.

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

This curriculum has been designed as part of the Scalable Games Design project.  
It was created using ideas from and portions of prior work completed by  
Fred Gluck

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 create a game of Sokoban
- To explain and use the Computational Thinking Patterns listed below

## Prerequisite Skills:

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

## Computational Thinking Patterns:

- Cursor Control
- Collision
- Polling
- Push
- Pull

## Length of Activity:

- Five 30-45 minute lessons, although some students may advance more quickly

## Activity Description:

- Part 1: Create basic world (worksheet) and the agents
- Part 2: Program the Sokoban to move and push crates
- Part 3: Creating letter and number agents
- Part 4: Polling and ending the game

## Table of Contents

<b>Teacher Instructions:</b>	<b>Part 1 – Create Worksheet and Agents</b>
<b>Student Handout 1A:</b>	<b>Part 1 – Create Worksheet and Agents</b>
<b>Student Handout 1B:</b>	<b>Part 1a - Create a Game</b> <b>Part 1b – Create Agents</b> <b>Part 1c – Create a Worksheet</b>
<b>Student Handout 1C:</b>	<b>Agent Creation Models</b>
<b>Teacher Instructions:</b>	<b>Part 2 – Understand Agents Behaviors</b>
<b>Student Handout 2:</b>	<b>Part 2 – Understand Agents Behaviors</b>
<b>Teacher Instructions:</b>	<b>Part 3 – Program the Sokoban to move</b>
<b>Student Handout 3:</b>	<b>Part 3 – Program the Sokoban to move</b>
<b>Teacher Instructions:</b>	<b>Part 4 – Program the Sokoban to push crates</b>
<b>Student Handout 4:</b>	<b>Part 4 – Program the Sokoban to push crates</b>
<b>Teacher Instructions:</b>	<b>Part 4 – The Destination</b>
<b>Student Handout 4:</b>	<b>Part 4 – The Destination</b>
<b>Teacher Instructions:</b>	<b>Part 6 – Creating letters and numbers</b>
<b>Student Handout 6:</b>	<b>Part 6 – Creating letters and numbers</b>

## Sokoban (Continued)

**Teacher Instructions:** Part 7 – Incrementing numbers

**Student Handout 7:** Part 7 – Incrementing numbers

**Teacher Instructions:** Part 8 – Winning the Game

**Student Handout 8a:** Part 8 – Winning the Game

**Student Handout 8b:** Part 8 – Winning the Game

**Student Handout 8c:** Part 8 – Winning the Game

**Student Handout 8d:** Part 8 – Winning the Game

## Vocabulary/Definitions

- Absorb.....**This is the opposite pattern of Generate. Instead of an agent generating other agents, an agent absorbs a flow of other agents in the absorption pattern (i.e. a tunnel absorbing cars), making them ‘disappear’
- Action .....**the requested behavior of an agent if the conditions are true
- Agent .....**a character in the game
- Array .....**a rectangular arrangement of agents
- Broadcast .....** controllers broadcast (or send out) a signal
- Collision .....**the situation when two agents physically collide.
- Condition .....**the situation that must be ‘true’ for an action to occur
- Depiction.....**a second image of the original agent. For example, the Sokoban can have two depictions: what it usually looks like, and what it looks like after it has been squished
- Generate.....**the ability to create a new agent. To satisfy this pattern, an agent is required to generate a flow of other agents; for example, cars appearing from a tunnel
- Global Variable...a variable accessible by all agents**
- Increment.....**to increase by one
- Method .....**a set of rules to follow in a specific situation
- Set .....**programming code which assigns a value to a simulation property

## General Teaching Strategies<sup>1</sup>

### Basic Philosophy

- The educational goal of these lessons is to learn and apply Computational Thinking Patterns in the context of a familiar game. Emphasis on these Computational Thinking Patterns is essential for student transfer of programming concepts between related games and simulations.
- Every effort has been made to create instructions with an eye toward guided discovery. Direct instruction has been used for those aspects where students are learning the code for the first time; however, materials have been provided to ensure that students are understanding the programming concepts, as opposed to simply copying code. Note that for each curriculum guide, special materials have been designed for students who are new to this program.
- Student materials are available for each portion of the game design. These materials are intended to be used in addition to teacher materials, which provide prompts and discussion points. Students may become frustrated with too little teacher support. Students may lose out on conceptual understanding with too much teacher support.

### Guided Discovery Process

- **Model the process** rather than just giving students the answer. Build the game on your own, before trying it with your students to enable you to see possible struggling points.
- Have students work through problems on their own. Ask guiding questions or give helpful suggestions, but **provide only minimal assistance** and only when needed to overcome obstacles.
- Don't fear **group work!** It is common for computer programmers to talk through problems with one another, and to use code snippets found from other programs, and

---

<sup>1</sup> 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.

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.

## Sokoban (Continued)

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 some that might be more effective than others

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

### Building Blocks

- Each project is designed to build on the prior one. Very little student support is provided where expertise has already been created. Conversely, material or programming techniques that are relatively new necessarily include more student support.
- Be sure to talk through the building blocks (especially for PacMan in the area of diffusion and Hill Climbing) as these Computational Thinking Patterns will appear often in future games and simulations.
- Remember that conceptual understanding takes time, and it may be necessary to explain some concepts multiple times, using different examples in different situations, so that all students can be successful.

### Support Learning

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

### Differentiated Instruction



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

## Sokoban (Continued)

- **Students who need a challenge:** Some students with more fluency in programming may finish this very quickly – be prepared for them to move on earlier than other students by having student materials ready in advance.
- **Students who need more assistance:** Other students (especially those with no prior programming experience with AgentSheets) may struggle a bit more. There are two options for differentiated instruction. Consider the needs of the student and the class as you decide which approach will work best.

**Students who need a challenge:** Some students with more fluency in programming may finish this very quickly – be prepared for them to move on to the remaining parts.

**Students who need more assistance:** Other students may struggle a bit more. There are two options for differentiated instruction. Consider the needs of the student and the class as you decide which will work best.

Option 1: Pair the struggling student with an experienced student

Option 2: Provide the struggling student with all student handouts, which provide more directed instruction steps

*Note that Part 8 is particularly scaffolded within the handouts.*

**Time management issues:** While students can be more engaged when they design their own agents, some students can spend too much time on this design or find it frustrating. Some handouts provides block images of each agent as portrayed in this lesson, which can be used as general guidance on how to create the agent.

- **Vocabulary for ELL Students:** Absorb, Action, Agent, Array, Broadcast, Collision, Condition, Depiction, Generate, Global Variable, Increment, Method, Set
- **Note:** Two student packets are available. The STANDARD packet is for students with some basic knowledge of AgentSheets. The ALTERNATIVE packet is for students with NO prior AgentSheets experience who may need more support. The ALTERNATIVE packet should NOT be used for most students as it significantly reduces the active thinking processes.



## Teacher Instructions:

### Part 1 – Create Worksheet and Agents

Introduce this project to the students by asking them if they know the game, Sokoban.

- Ask students to explain how the game works, and the rules of the game.
- You may choose to show a video or two of a typical Sokoban game
  - <https://www.youtube.com/watch?v=Ht4sC9PXCpw>
  - <https://www.youtube.com/watch?v=Q2JczQVPjC0>

If you show these videos, ask your students...

- What does the Sokoban have to do?
- How do you win the game?
- What special features are part of the game?
- What are the triangles for?
- Why does the Sokoban have to walk around the box?
- Is there a strategy to win this game?

Explain that these are all design features that must be considered when planning a game. Now, tell them that they will be designing their own Sokoban game.

As a class, briefly create a description of the Sokoban game similar to the one below.

- identify game objects, called agents, by locating nouns in the game description

*You are a warehouse keeper (Sokoban) who is in a maze viewed from above. You must push boxes around the maze while trying to put them in the designated locations. Only one box may be pushed at a time, and boxes cannot be pulled. When boxes are covering all of the destinations, the level is complete.*

- categorize agents into user controlled agents (hint the game is called Sokoban), agents that move or do other things by themselves (sometimes also called artificial intelligence agents) and completely passive agents acting as props such as the street.

*User controlled agents:*

- *Sokoban*

## Sokoban (Continued)

*Artificial Intelligence Agents:*

- *walls, boxes, destinations*

*Passive Agents:*

- *Floor Tiles*
- identify agent interaction by locating verbs in the game description

*You are a warehouse keeper (Sokoban) who is in a maze viewed from above. You must push boxes around the maze while trying to put them in the designated locations. Only one box may be pushed at a time, and boxes cannot be pulled. When boxes are covering all of the destinations, the level is complete.*

Give the students 10 minutes or so to work with a partner to discuss what steps will be needed to create this game. If students have already completed Frogger and Journey, they should have a pretty good idea of how to do this. Add new students to a pair of experienced students who will be willing to talk through their thinking. Stress the need to think through the programming process. At this point, there should not be hands on the keyboards (even though some will want to jump right in to programming!).

**This lesson is intended to be taught in a guided discovery manner. Be sure to give students time to work on their own and figure things out using the program. Encourage students to work together and talk through problems with one another. Emphasize that troubleshooting is a normal and important part of programming.**

**Solicit and discuss possible ideas, without providing any evaluative feedback (do not tell students if their ideas are good/bad, right/wrong). Once there has been some class discussion, provide students with the handouts.**

- **STUDENT HANDOUT 1A** simply provides the details needed for experienced students to get started on their own. (Found on page 3 of the STANDARD student packet)
- **STUDENT HANDOUT 1B** Provides step-by-step instructions for creating a worksheet and agents for students who are new to AgentSheets. (Found on page 3 of the ALTERNATIVE student packet)
- **STUDENT HANDOUT 1C** shows possible designs for each agent, but encourage students to create their own designs if time allows. (Found on page 4 of the STANDARD student packet and page 7 of the ALTERNATIVE student packet)

## Student Handout 1A:

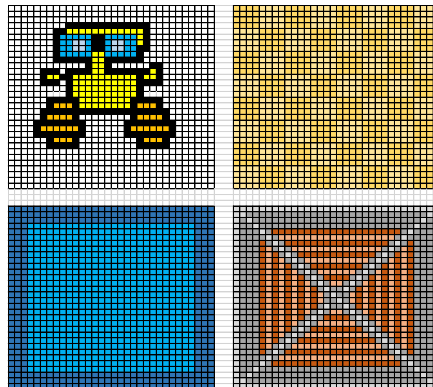
### Part 1 – Create Worksheet and Agents

In this project, you will create a worksheet with a Sokoban. This Sokoban will be tasked with pushing a crate to a specific destination. Since you have already created a prior game, these instructions will be less specific. If you are stuck, talk with the person next to you about ways to correct the problem.

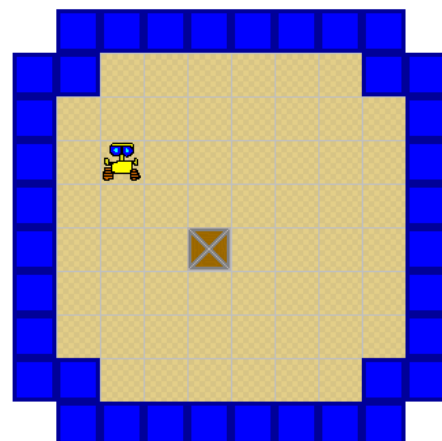
### Tasks:

1. Create a new game called Sokoban
2. Create agents for the game. You will need the following agents:

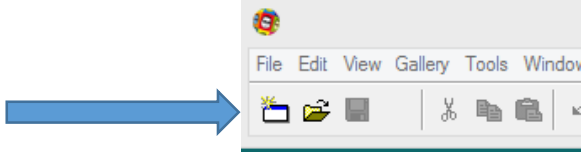
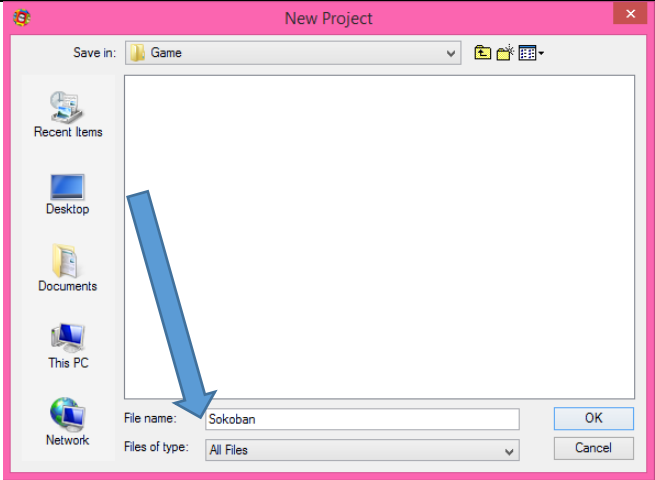
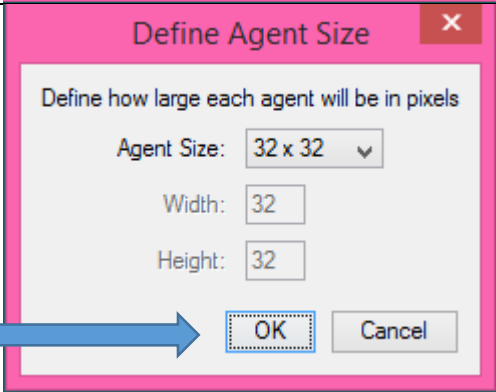
- Sokoban
- Floor Tile
- Wall
- Crate



3. Create the worksheet for the game. Here is the basic worksheet – you will have an opportunity to make it more complex later in the course.
4. Enable the Sokoban to be cursor controlled, such that it moves up/down/right/left with the arrow keys.
5. Prevent the Sokoban from walking through walls.

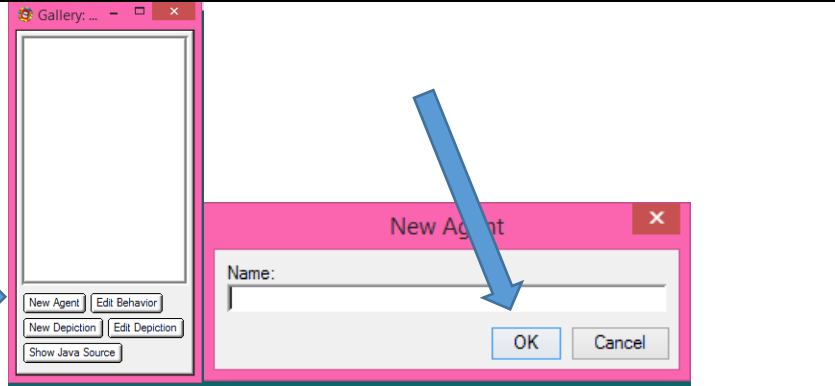
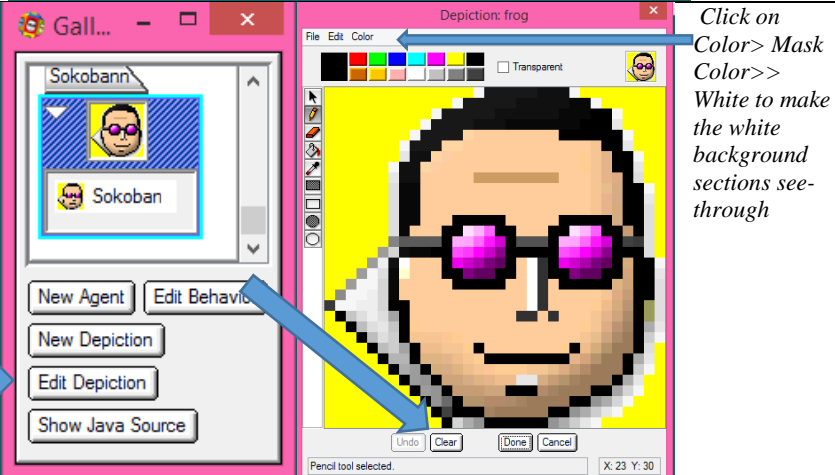
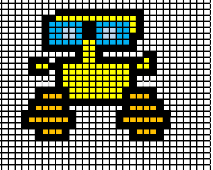
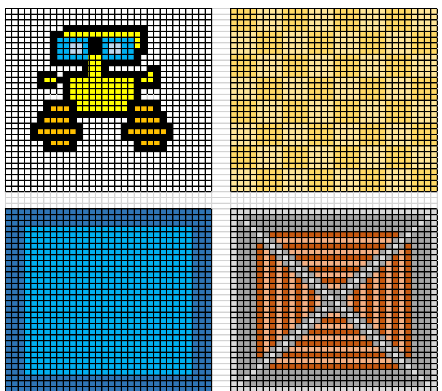


## Student Handout 1B: Part 1a – Create a game

<p><b>Step 1</b></p>	<p><b>Create Game</b></p> <p>Click on the new game icon (far left)</p>	
<p><b>Step 2</b></p>	<p><b>Name the Game</b></p> <p>Name it Sokoban and click OK</p>	
<p><b>Step 3</b></p>	<p><b>Define Agent Size</b></p> <p>Do not change - Click OK</p>	

## Sokoban (Continued)

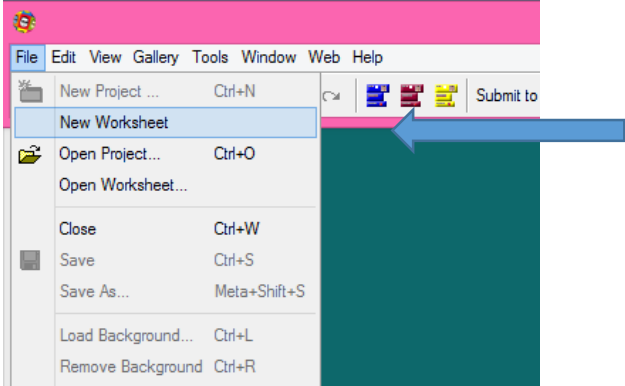
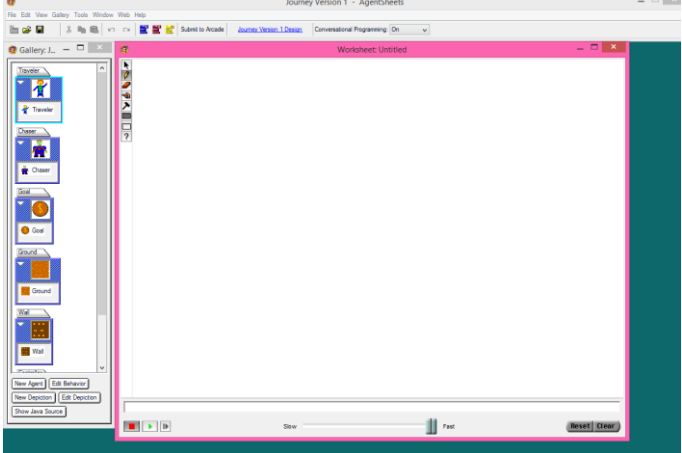
### Student Handout 1B: Part 1b – Create agents

<p><b>Step 4</b></p>	<p><b>Create Agent</b></p> <p>Click on New Agent</p> <p>Name it Sokoban</p> <p>Click ok</p>	
<p><b>Step 5</b></p>	<p><b>Edit Agent</b></p> <p>Click Edit Depiction</p> <p>Click Clear to erase the current image.</p>	 <p><i>Click on Color&gt; Mask Color&gt;&gt; White to make the white background sections see-through</i></p>
<p><b>Step 6</b></p>	<p><b>Draw Sokoban</b></p> <p>Click Done</p>	 <p>Here is an example of one way to draw the Sokoban. You can be creative. If you make a mistake, use the eraser or click CLEAR to clear the whole area.</p>
<p><b>Step 7</b></p>	<p><b>Draw remaining agents</b></p>	<p>Sokoban      Floor Tile</p>  <p>Wall      Crate</p>

## Sokoban (Continued)

### Student Handout 1B: Part 1c – Create Worksheet

The worksheet is the game space –  
it is where the agents will perform their actions.

<b>Step 8</b>	<b>Make the worksheet</b>  <b>Click File&gt;&gt;New Worksheet</b>	
<b>Step 9</b>	<b>Make the worksheet bigger</b>  <b>Notice it is big, but not so big that it fills up the whole space.</b>	

#### Select Tool



**Pencil Tool** – places a single agent on the worksheet

**Eraser** – erases agents from the worksheet

Will be defined later


Will be defined later

**Draw Rectangle** – places agents in an array (rectangle)

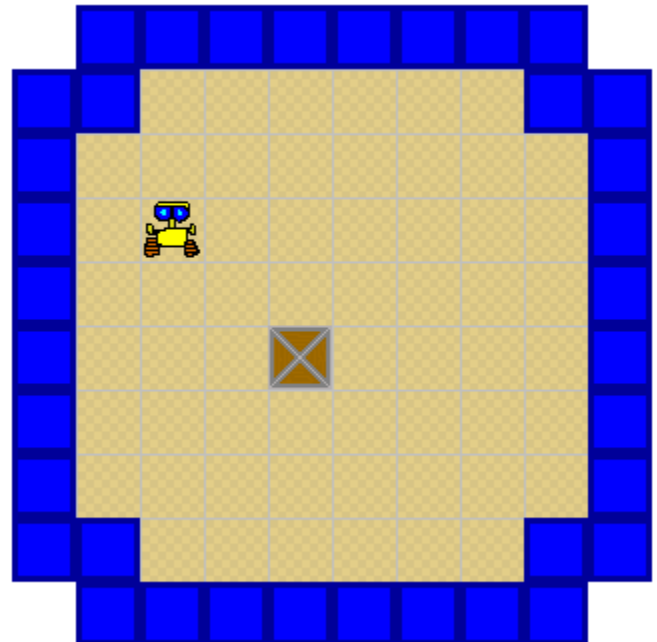
**Erase Rectangle** – erases agents in an array

Will be defined later

## Sokoban (Continued)

<b>Step 10</b>	Use the tools to place items on the worksheet.  <b>Pencil:</b> places agents one at a time  <b>Filled in Rectangle:</b> Places agents in an array.	 <p>It is important that you do not draw over the agents. Agents can stack on top of one another which is not visible from a 2-dimensional perspective. Take care to create a single layer of agents.</p>
----------------	--	--

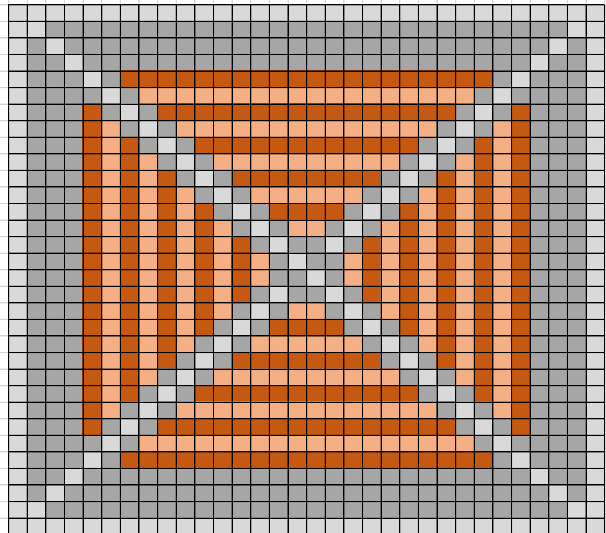
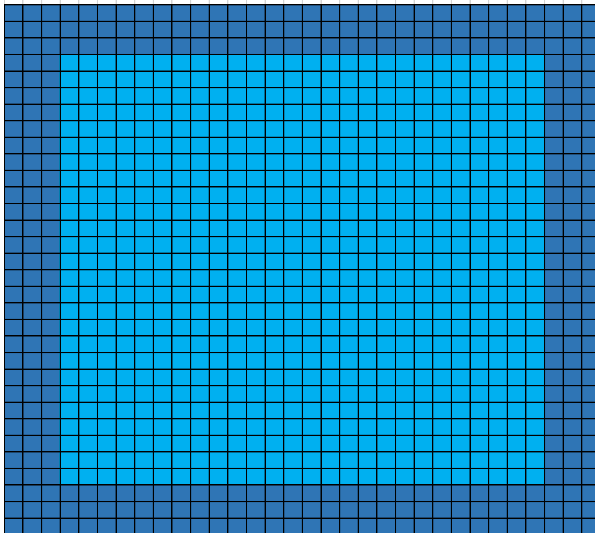
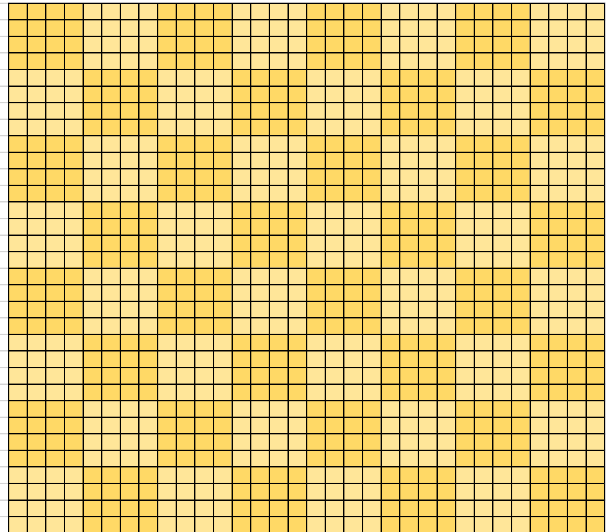
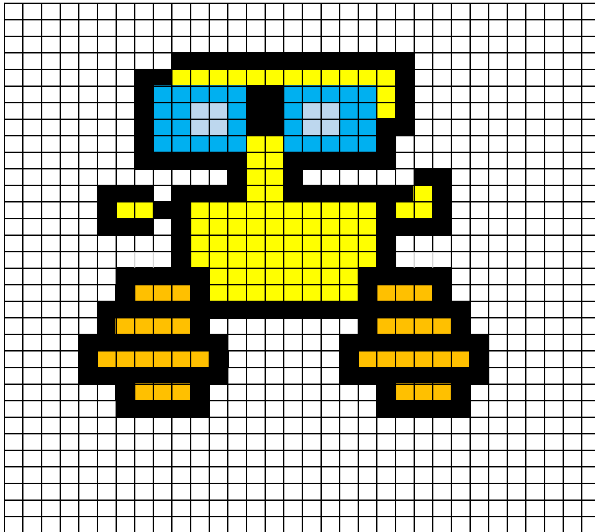
This is what your worksheet might look like at this point.



**This is a good time to save the worksheet!**

## Student Handout 1C: Agent Creation Models

Use these as quick starting points for your own agent. They don't have to look exactly like the model!





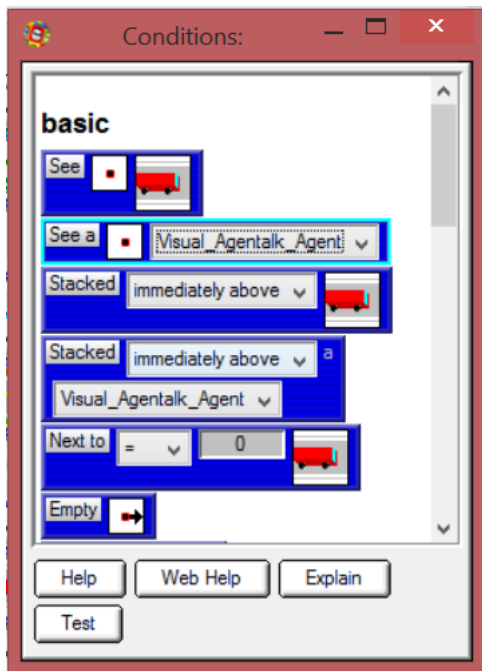
*This section is review for experienced students but would be helpful for students new to programming.*

## Teacher Instructions:

### Part 2 – Understanding Agent behaviors

At this point, students should have all six agents created and their worksheet created. Now they are ready to give their agents behaviors.

Behaviors are thought of in this way...



IF...I have enough money.... THEN .... I can go to the baseball game.

Sometimes there is more than one condition

IF...I have enough money and I have a ride.... THEN .... I can go to the baseball game.

Sometimes there is more than one action

IF...I have enough money and I have a ride.... THEN .... I can go to the baseball game and buy a soda.

The CONDITIONS box lists the conditions under which the action will occur. The ACTIONS box lists the actions which occur when the CONDITIONS are true. **Student Handout 2** is designed to give students practice in this (Found on page 8 of the ALTERNATIVE student packet – not included in the STANDARD student packet.). You can give it to them to solve in pairs, or use it as a class activity.

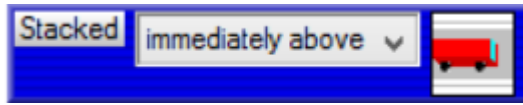
## Student Handout 2: Understanding Conditions and Actions

Explain each condition or action below

Conditions:



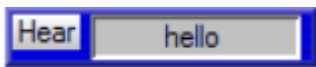
A \_\_\_\_\_



B \_\_\_\_\_



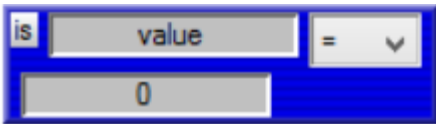
C \_\_\_\_\_



D \_\_\_\_\_



E \_\_\_\_\_

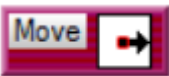


F \_\_\_\_\_



G \_\_\_\_\_

Actions:



A \_\_\_\_\_



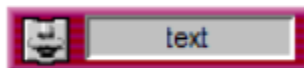
B \_\_\_\_\_



C \_\_\_\_\_



D \_\_\_\_\_



E \_\_\_\_\_

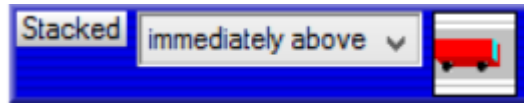
## Student Handout 2: ANSWER KEY Understanding Conditions and Actions

Explain each condition or action below

Conditions:



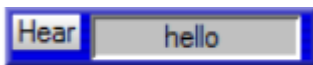
**A** IF the agent sees a truck to the right



**B** IF the agent is on top of the truck



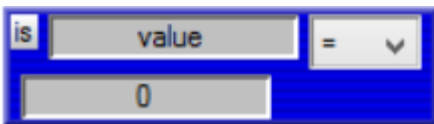
**C** IF there is nothing to the right



**D** IF the agent hears HELLO



**E** IF the user presses the A key

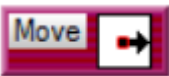


**F** IF the value is equal to 0



**G** IF the agent is next to five or fewer trucks

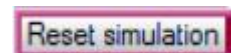
Actions:



**A** Move to the right



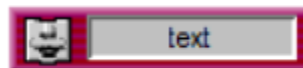
**B** Erase the agent



**C** Reset the simulation



**D** Change the agent to this image



**E** Speak this text

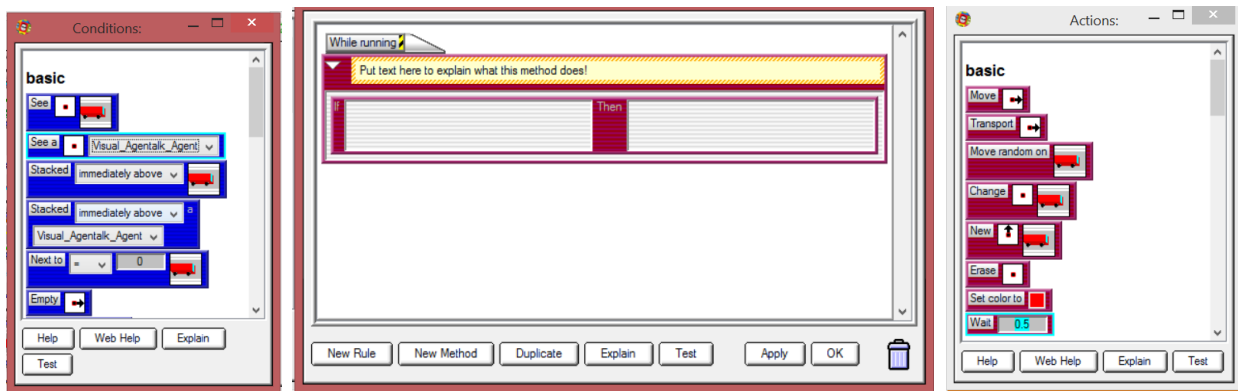
## Teacher Instructions:

### Part 3 – Program the Sokoban to move

- This section is review material for experienced students and will likely not be needed. It is provided for classes with a higher proportion of new students and is found on Page 9 of the ALTERNATIVE student packet. If your class has a high number of new students, consider using a breakout group to teach this information while experienced students move ahead.

Ask the students to edit behaviors of the Sokoban. To do this, they will click on EDIT BEHAVIOR on the SOKOBAN located in the GALLERY BOX.

Three boxes will appear as shown below: the CONDITIONS box, the ACTIONS box and the BEHAVIOR box. (It is helpful for the students to lay them out in this order)



Use drop and drag to bring in conditions from the left side of the BEHAVIOR box, and actions from the right side.

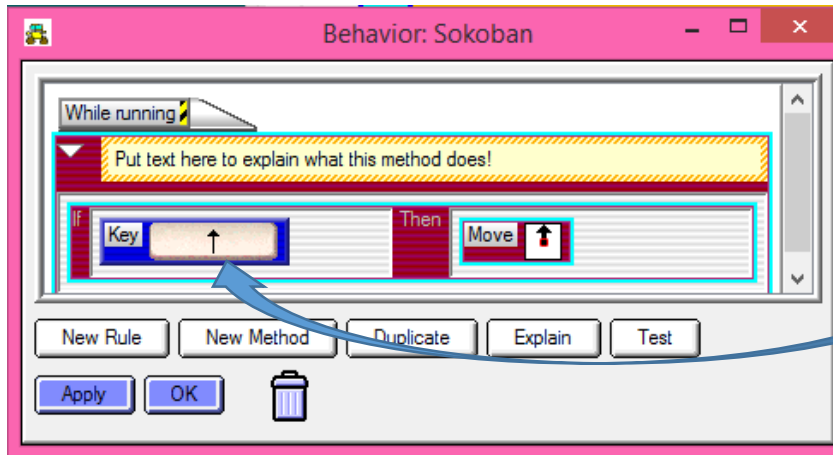
Ask students...

How will we get the Sokoban to move? Can we use voice control? (Just tell the Sokoban to move?) No, what else could we use?

A student will likely suggest using the keyboard. Introduce the term CURSOR CONTROL such that you use the keyboard to move an Agent. Have students work together to determine the proper condition and action to make the Sokoban move up.

## Sokoban (Continued)

After a couple of minutes of discussion, solicit ideas (correct code is shown below)



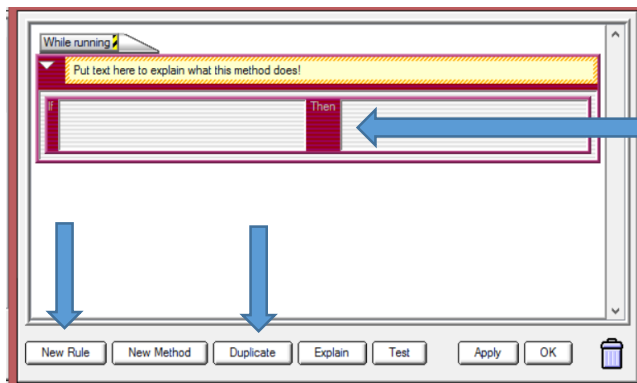
Press the UP ARROW key on the keyboard to get the up arrow indicator.

Have the students test their rule. Click on the green arrow (PLAY) to start the game. Have them press the up arrow to see if their Sokoban moves up. Press the Red Square (STOP) to stop the game, and RESET to reset the worksheet. When the student presses the up arrow, the Sokoban should move up.

Helpful  
Tips

NOTE: Some students will likely find out at this point that they didn't save their worksheet. If that is the case, they will need a few minutes to recreate their worksheet. You can avoid this by reminding everyone to save their worksheet before testing their game!

Show students how to add rules by clicking NEW RULE at the bottom of the box. Also show them how to DUPLICATE rules using the DUPLICATE button.



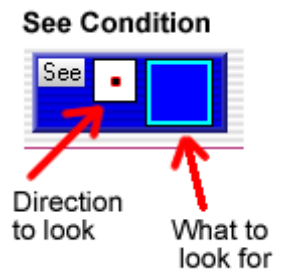
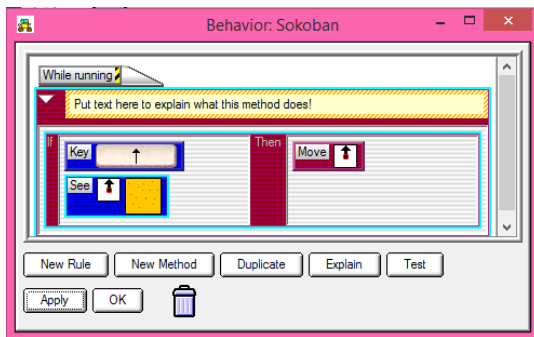
Select a rule by clicking in the center of it. Click on DUPLICATE to duplicate the rule. Click on DELETE to remove it.

Have the students create the rest of the rules to move their Sokoban up, down, right and left.

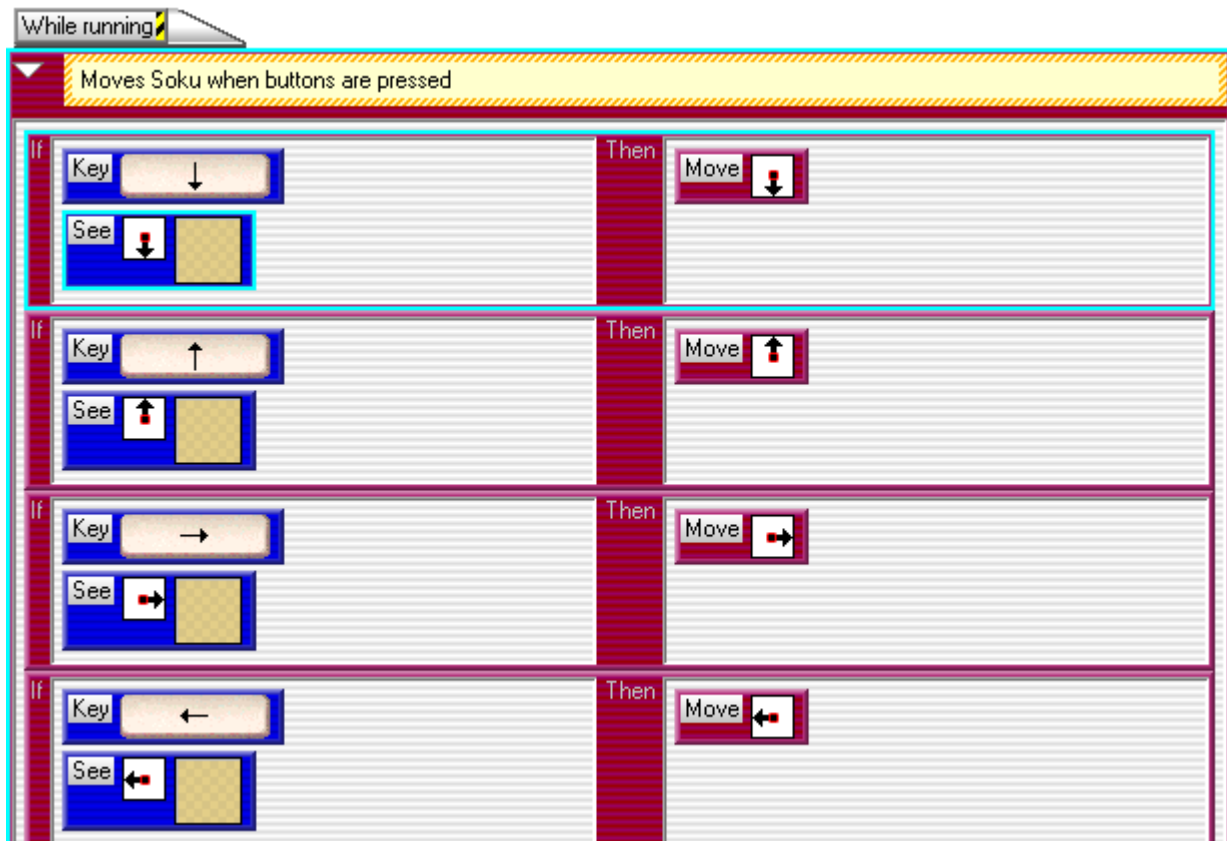
## Sokoban (Continued)

When the students test the movement of the Sokoban, ask the students what happens when the Sokoban gets to a wall? (Right now, the Sokoban can walk through walls.) Ask students how they can prevent this.

Suggest the SEE condition if no one else suggests it. The correct code for moving up is as follows:

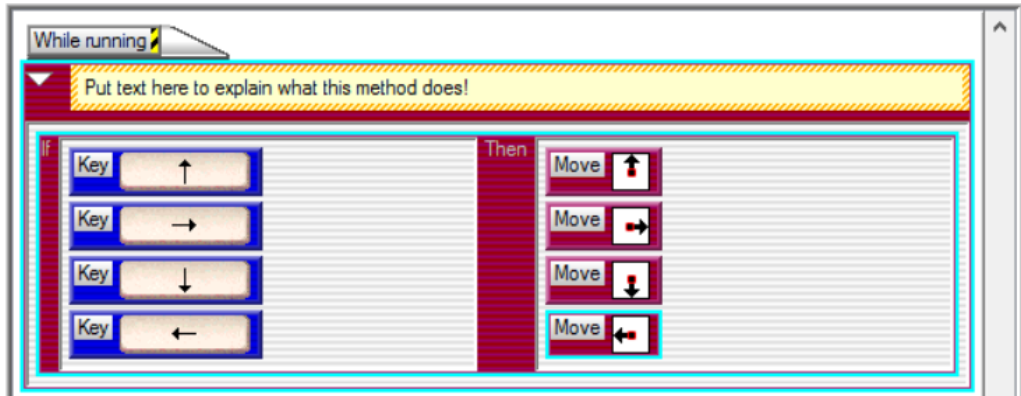


The entire code for the Sokoban looks like this:



## Sokoban (Continued)

### COMMON MISTAKE:

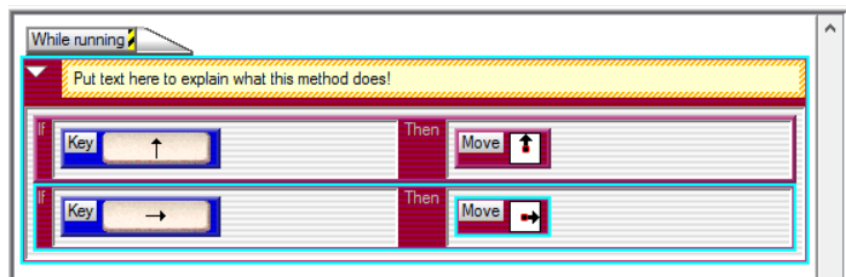


In this case, the agent is being 'told' ...if I click the up, down, right and left arrow, then you move up, down right and left.



*If students have this mistake, use it as a class discussion for them to determine why it's wrong. Consider having them talk through (or act out) what it occurring.*

The rule should be,  
if I click the up arrow,  
you move up.  
If I click the right arrow,  
you move right.



They must each be their own, separate rule!

### Thoughts on Troubleshooting:

Think about ways to remove yourself as a teacher from being the only problem solver...

Consider these prompts when students ask for help...

- What have you already tried?
- Have you consulted with a friend?
- Have you considered whether your rule order is correct?

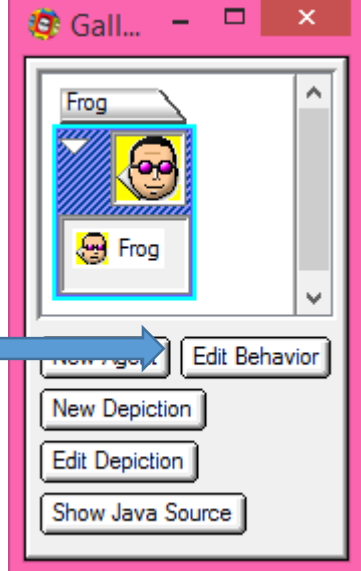
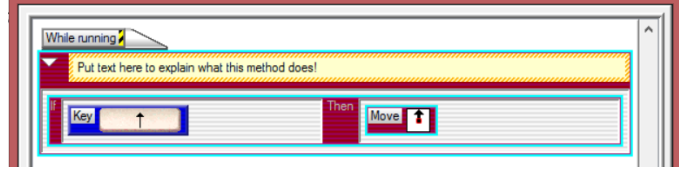

Resist the urge to find their mistake or take over their computer and fix their program!

Use the proper terms for the Computational Thinking Patterns (such as Cursor Control) so that students are comfortable with these terms.

## Student Handout 3:

### Part 3 – Program the Sokoban to move

*Click on the agent to add behaviors to that agent*

<p><b>Step 1</b></p>	<p><b>Create behavior for the Sokoban</b></p>	
<p><b>Step 2:</b></p>	<p><b>Cursor Control for Sokoban</b></p> <p>This makes the Sokoban move UP when you push the UP arrow. Create the rest of the rules for the Sokoban</p>	
<p><b>Step 3:</b></p>	<p><b>Prevent the Sokoban from walking through walls.</b></p> <p>Create the rest of the rules for the Sokoban</p>	 <p>Notice that the rule says IF I push the DOWN arrow, AND I see a floor tile in the down direction, THEN I move down.</p>



## Sokoban (Continued)

### Teacher Instructions:

### Part 4 – Programming the Sokoban to Push Crates

Gather the class to talk about the crates. Consider these prompts:

- What do the crates do?
- Where do they come from?
- Where do they go?
- How do they move
- How often do they travel on the street?
- What happens with the crates when they reach their destination? Do the crates disappear?

**New Computational Thinking Pattern: PUSH**  
When the Sokoban tries to "push" a Crate, the Crate will look ahead in the same direction as the "push". If the space ahead of the box is clear (no Walls) the Crate will signal the Sokoban to move one space in the "push" direction and the Crate will also move one space in the same direction.

By now your students should have a pretty good idea of what types of coding will be necessary. Solicit ideas and allow students to consider both correct and incorrect suggestions. Move students toward the idea that the Sokoban will PUSH the crates, but then point out that there is no PUSH command. Remind experienced students (inform new students) of how we can use the MAKE action to put a set of actions in motion.

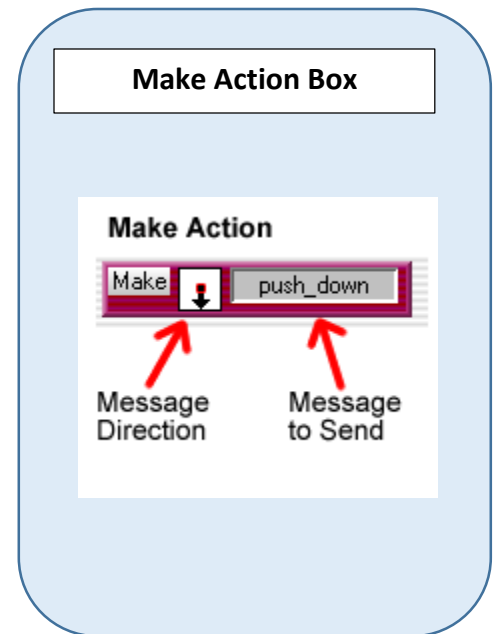
IF I see a CRATE to the RIGHT, I should PUSH the CRATE to the RIGHT

To have the Sokoban push crates, we have to learn about a new behavior action: “**Make**”. The Make action allows sending messages between agents, even back to yourself!



## Sokoban (Continued)

Use these behaviors IN ADDITION to the existing behaviors for the Sokoban.



Have the students test out their code. Ask them to push a crate. What happens?

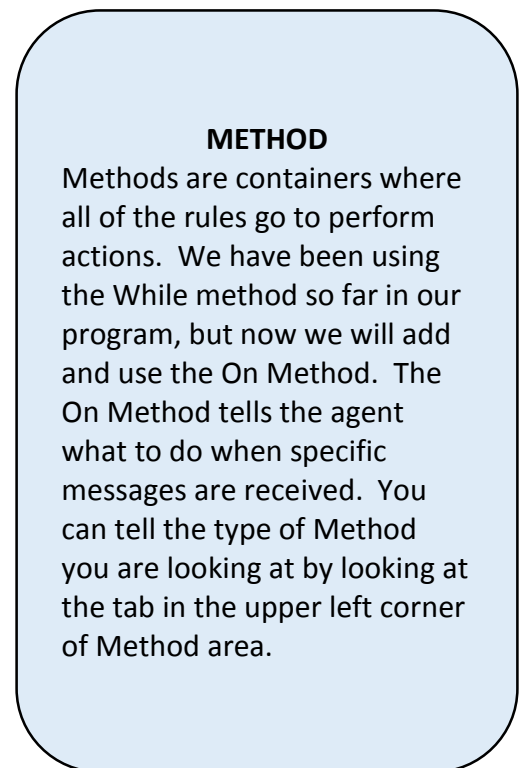
They should get an error message that says, **“Sorry, I am just a ‘crate’. I do not know how to react to the message: ‘push\_down’”**

Again, prompt the students...

- **Why are we getting that error message?**
- **What does the error message mean?**
- **How will we resolve it?**

Again, allow students time to think through this problem – do not give them the next step right away....

Once they’ve had a chance to talk through this new problem, remind students of using a METHOD. (METHODs were used in the polling for goals in the Journey game.) Methods send messages to agents, telling them what actions to take.



# Sokoban (Continued)

## Thinking through pushing the crate...

Remind students that the agents will only do what they are told...they cannot choose to do anything on their own.

So, ask the students what commands are needed to push a crate, and model the commands they give. You can use a tissue box or a chair to model the behavior. Be sure to have some 'Walls' to work around.

For example...

When they say...	Do this...
Push the crate	Push the (tissue box) crate forward, but don't move.
Move up	Move yourself, but not the crate
Push the crate (but there is a wall there)	Bang into the wall, but don't go through it

You want to push them to a solid understanding by modeling each spoken behavior. Make them give you the conditions (for example, move forward if you see carpet in front of you).

This will be challenging and frustrating at times for some students. Take it step by step and let all students participate in giving possible solutions.

Eventually you should get the students to agree on the following set<sup>2</sup> of commands.

*If you see carpet in front of you, move the crate forward. If the crate moves forward, then you should take a step forward as well.*

More specifically, these rules...

- If there is a crate in front of the Sokoban, and the "right arrow" button is pressed, perform the action, "push\_right"
- If the crate is told to perform the action "push\_right", the crate will look to see if there is floor tile in front of the crate. If there is, the crate will move forward, and then then tell the Sokoban to perform the action "move forward".
- If the Sokoban is told to perform the action "move forward" the Sokoban moves forward one step.

**Provide students with Handout 4** (This handout is found on page 5 of the STANDARD student packet and page 9 of the ALTERNATIVE student packet.)

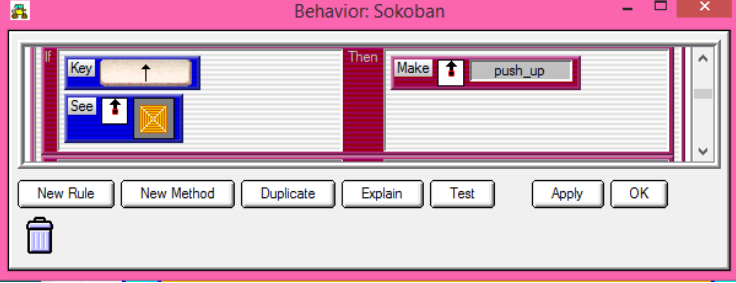
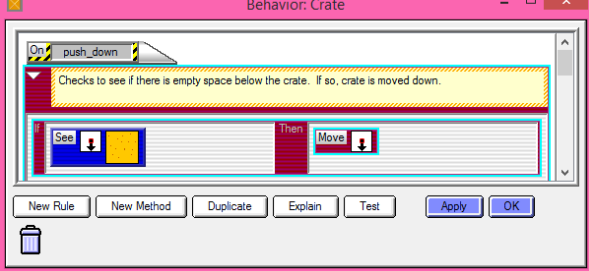
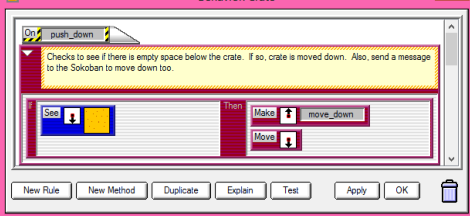
---

<sup>2</sup> Note that there are other possible solutions – encourage students to try out their solutions as well.

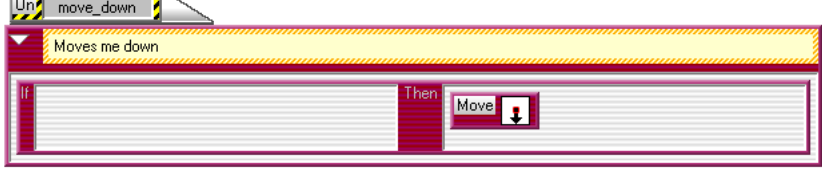
## Student Handout 4:

### Part 4 – Programming the Sokoban to Push Crates

*Click on the agent to add behaviors to that agent*

<p><b>Step 1</b></p>	<p><b>Enable the Sokoban to push the crates.</b></p> <p>This is the behaviors in the UP direction for the Sokoban. Code the rest of the directions.</p>	
<p><b>Step 2:</b></p>	<p><b>Create the Method push_down for the CRATE</b></p> <p><b>Reminder: click New Method</b></p>	
<p><b>Step 3:</b></p>	<p><b>Create remaining methods for push_up, push_left and push_right</b></p>	<p>No hints here!</p>
<p><b>Step 4:</b></p>	<p><b>Test your game</b></p>	<p>You should not get any error messages. Your crate should move in the proper direction. Does your Sokoban move? Why not?</p> <p>If you get any error messages, go back and check your programming. Do not continue on until the program works as expected at this point.</p>
<p><b>Step 5:</b></p>	<p><b>Change the Sokoban rules so that your Sokoban moves down when the crate is pushed down.</b></p>	 <p>Crate checks to see if a Floor tile agent is below. If there is a Floor tile agent below the crate, it <u>sends a message</u> back to the Sokoban telling it to move down and then the crate moves down.</p>

## Sokoban (Continued)

<b>Step 6:</b>	<b>Test your game</b>	The Sokoban does not know how to react to the message (move_down) that it is receiving back from the crate; therefore, you will see an error message from AgentSheets when we run our game now!
<b>Step 7:</b>	<b>Create the Method move_down for the SOKOBAN</b>	
<b>Step 8:</b>	<b>Change the Sokoban rules so that your Sokoban moves up when the crate is pushed up.</b>  <b>What other rules must be changed?</b>	Look back to Step 5 for help on this.
<b>Step 9:</b>	<b>Create the remaining “Move” methods</b>  move_up move_right move_left	See Step 7 for help on this.

**You are ready to move on once the following items work correctly...**

- Does the Sokoban move in all directions over the Floor?
- Can the Sokoban push crates in all directions?
- Does the Sokoban also move in the same direction as the crate was pushed?
- Do the Walls block the Crates and Sokoban correctly?

## Teacher Instructions:

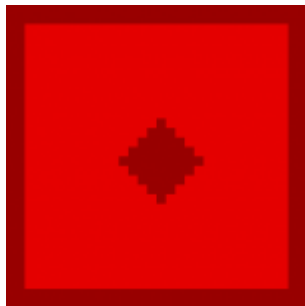
### Part 5 – The Destination

Ask the students what is missing from their Sokoban game... They should identify that they need a way to win the game. In this portion of the lesson, students will create a destination agent and change their rules to enable the Sokoban and the Crate to be on top of the destination tile.

### Provide students with Student Handout 5 – the remaining explanation is for clarity for the teacher

Student Handout 5 is found on page 7 of the STANDARD student packet, and page 12 of the ALTERNATIVE student packet.

We will now create a Destination agent that can have other agents pushed onto it.



Destination agent

**Create and draw the Destination Agent:** Create a new agent called Destination, the same way you created the Sokoban and the Walls. As you did when you created the Sokoban, draw the picture you'd like to use to represent the destination.

**Program Destination:** Now we would normally add some behavior to our new agent, but in this case there is nothing we need to program! This means that the destination will not actually perform any actions in our program.

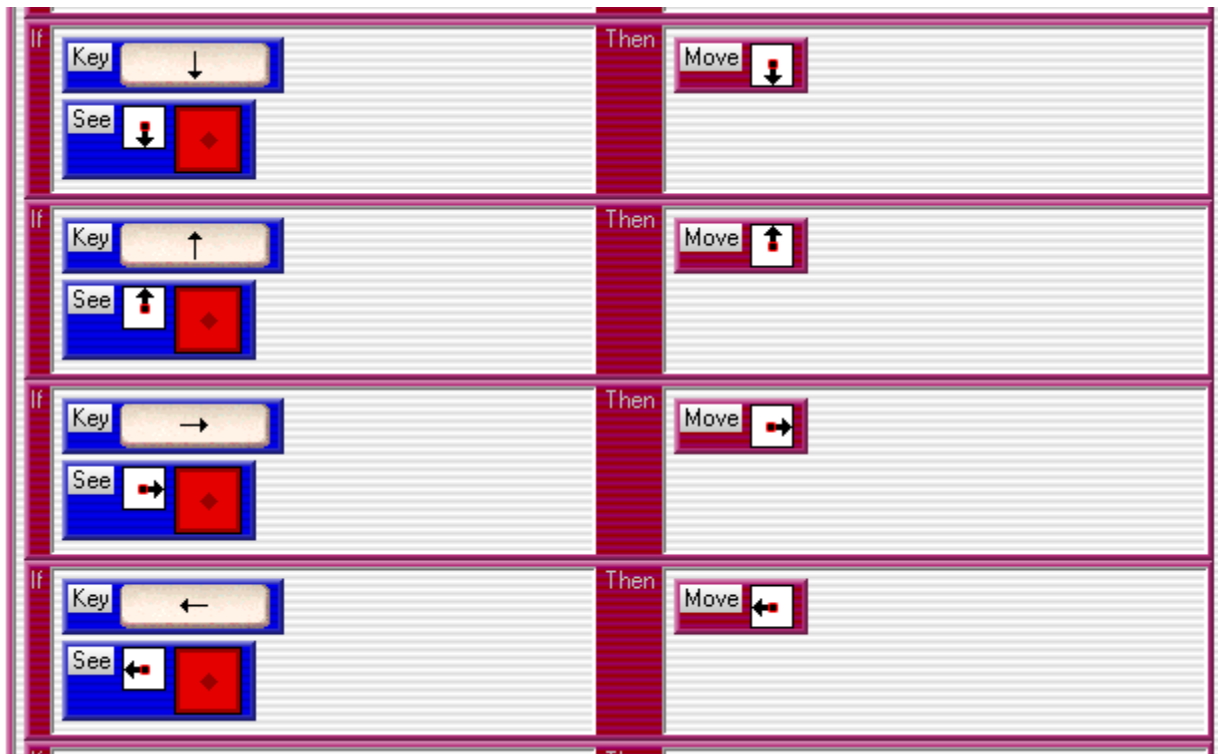
Currently our Sokoban can walk over floors without a problem, but cannot travel over destinations. We now need to allow the Sokoban to move over destinations as well.

ACADEMIC LANGUAGE and  
COMPUTATIONAL  
THINKING PATTERNS

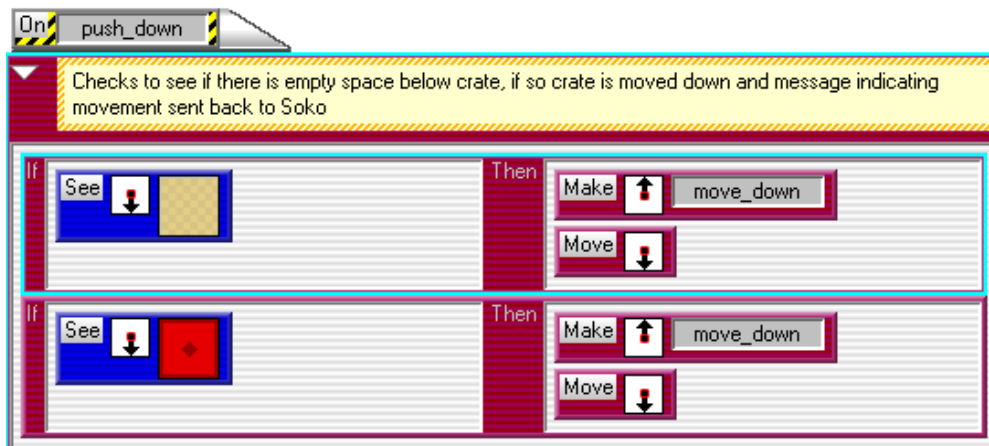
It is important to use the proper terminology during this discussion. As students use words like create, make, erase, disappear...restate their ideas using the terms GENERATE and ABSORB.

## Sokoban (Continued)

When you're done the Sokoban's behavior editor should have the original four movement rules for moving over Floor Tile agents and now four movement rules for moving over destinations too!



**Moving the crate over the Destination:** We can push the crate over the Floor tiles, however it will not currently move onto Destination agents and we will never be able to finish our game! It is actually very easy to fix this behavior. Open the Crate behavior editor and find the On method for "push\_down", click on the If or Then labels of our rule (this should highlight the entire rule). Now click the "duplicate" button, which will make an exact copy of the highlighted rule. Now in our new duplicate rule, we just need to change the See action depiction to the Destination agent. Here is a picture of the modified "push\_down" method:



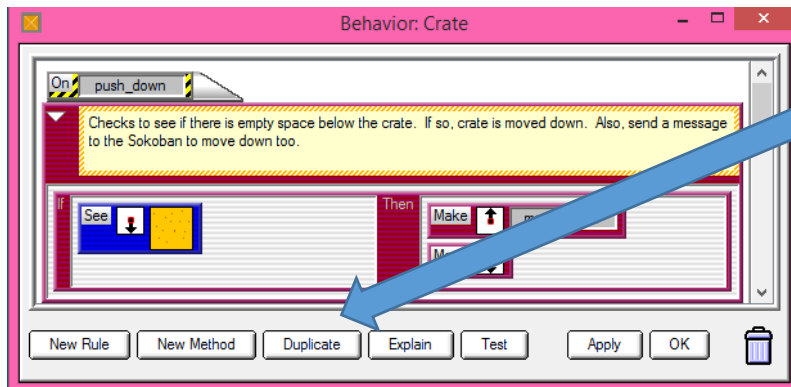
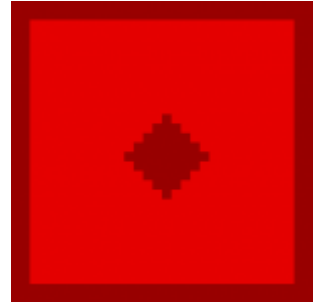
## Student Handout 5: The Destination

You are tasked with creating the destination tile for Sokoban. Here are the rules:

**Step 1:** Create missing agent (destination tile) and add it to the worksheet.

**Step 2:** Program the Sokoban to be able to move on the destination tile.  
Hint: you will be adding rules, not deleting or changing existing rules

**Step 3:** Program the Crate to be able to move on the destination tile.  
Hint: you will be adding rules to the method `push_down`, for example, not deleting or changing existing rules. This is a great opportunity to test out the **DUPLICATE** feature!



**Step 4:** Test the program. You are ready to move on when you can answer YES to these questions:

- Try to move the Sokoban onto and off of the Destination in every direction
- Try to push a crate on and off the Destination in every direction
- Do the Sokoban and Crate move on and off the Destination correctly? If not, check the Crate and Sokoban rules and retest.
- If the movement over the Destination is fine for the Sokoban and Crate, good work!



## Teacher Instructions:

### Part 6 – Creating letters and numbers

Notice that this game version is able to count the number of steps taken by the Sokoban to move the crate to the destination. Show this image to the students and ask them which pieces they need to program. They should come up with the following needs:

#### Letters

- Have no behaviors
- Indicate where the step count is located
- Create letters S-T-E-P

#### Destinations

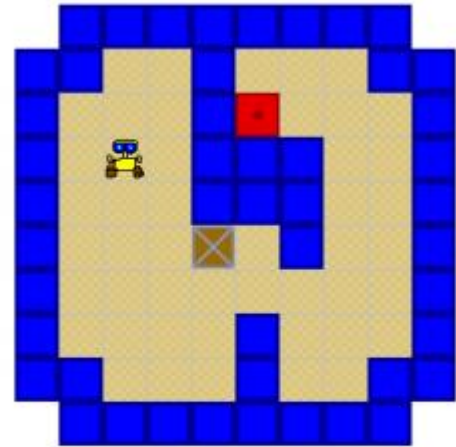
- Have no behaviors
- The Crate and Sokoban can move over Destinations
- If Crates cover all level Destinations the level is complete

#### Numbers

- Create numbers 0-1-2-3-4-5-6-7-8-9
- Increments by one every time the Sokoban takes a step
- Creates new number if needed
- Indicates to next Number when it needs to increment

#### Game Master

- Manages the incrementing of the step count
- Asks Crates to update their status
- If all Destinations are covered, indicates that the level has been finished



S T E P S

0

Remind students of how the goals were incremented by one when polled in Journey.

Remind students of the Controller Agent in Journey. This is the same role that the GAME MASTER plays in Sokoban.

## Sokoban (Continued)

In the STUDENT HANDOUT (found on page 8 of the STANDARD student packet and page 13 of the ALTERNATIVE student packet), students will be directed to make the letters S-T-E-P as one agent called Letter, with four depictions, Letter\_S, Letter\_T, etc.

Students will then be directed to make the numbers 0-1-2-3-4-5-6-7-8-9 as one agent called Number, again with different depictions.

Students will then update their worksheet.

### Tips and Tricks

We actually do not need to modify the original Letter or Number depiction since we will not use it in our game, however it can be helpful to modify it.

Making depictions that have meaningful names is a good programming practice! When we use the "Letter\_S" depiction of the Letter agent we know what it should look like, however if we just use "Letter" we would not know which letter it is referring to.

Change the MASK color to WHITE to make it transparent!

## Student Handout 6

### Part 6: Counting the steps

To count the steps in Sokoban, you first need to create agents that are letters and numbers.

Step 1: Create a letter agent

Create 4 depictions:

- Letter\_S
- Letter\_T
- Letter\_E
- Letter\_P

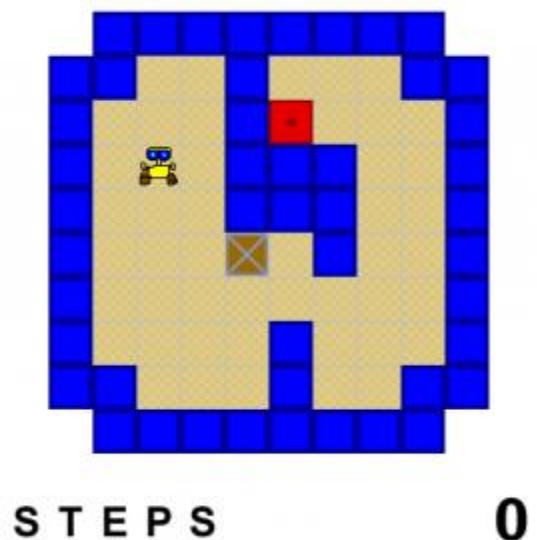
Step 2: Create a number agent

- Create 10 depictions for the numbers 0-9

**0 1 2 3 4 5 6 7 8 9**

Step 3: Modify worksheet

Add the word STEPS and add the digit 0 as shown.

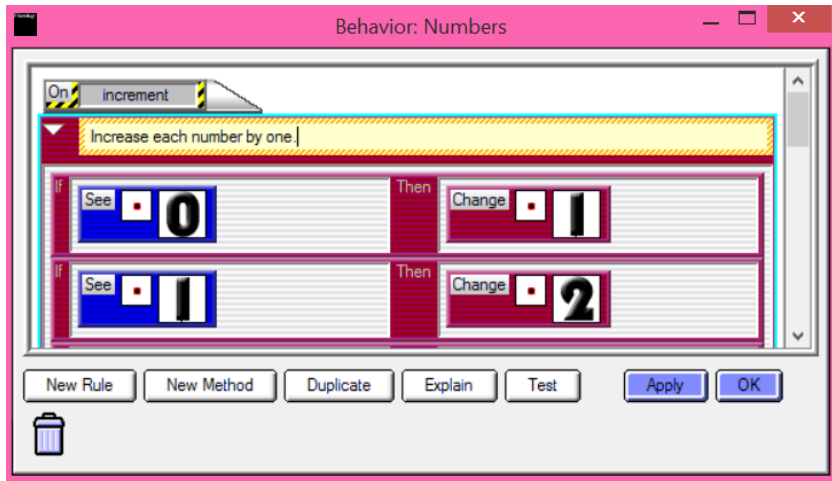


## Sokoban (Continued)

**Step 4: Add "increment" Method to the Numbers behavior**

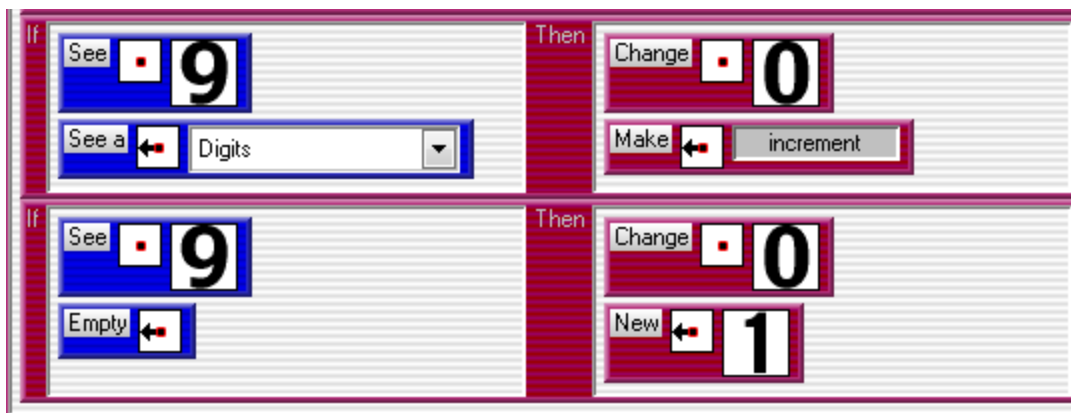
**Step 5: Add Rules to Numbers Agent "increment" Method:**

We now need to add rules to the Method we just created. We will actually have eleven rules, even though there are only ten numbers! Let's start with the first rule; if we see a "Zero" depiction, we need to change it to a "One" depiction. Make your rule look like the first rule in the following picture.



Use the other two rules from the picture as guides for how to make rules for the numbers 1 - 8.

What should happen if the current number is a "Nine" and we need to increment? We will need two special rules for this case. Use the picture below for the two "Nine" rules.



## Sokoban (Continued)

We used the "See A" Action for one of the rules. The difference between the "See" and "See A" Actions is that "See A" looks for any Agent regardless of the Depiction, while the "See" Action looks for a specific Depiction of an Agent.

We could program the same behavior using the "See" action as we did using the "See A" action, however it would require a separate rule for each Numbers depiction!

**\*\*Warning:** *If your counter is not on empty space (i.e. on the floor, wall, etc.) you want to make sure that the last "if" is not "empty to the left" but rather "sees floor to the left" or whatever you have the counter on....*

### **Fun Fact:**

We are not actually "incrementing" any numbers with our "increment" Method. We are updating Depictions to represent incrementing a number. We are simulating incrementing real numbers!

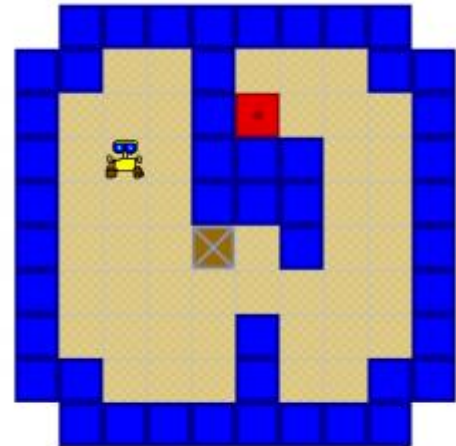
## Sokoban (Continued)

### Teacher Instructions:

### Part 7 – Incrementing Numbers

#### Flashback to Journey

In this game, the Traveler had to collect multiple goals before winning the game. To determine if all the goals were gone, we created a Controller to poll the goals, which increased the count by one, for each goal remaining on the board.



STEPS

0

Remind students of the Sokoban game, and how they used a Controller to count up the number of remaining goals. When the remaining goals were zero, the game was over. They will use the same setup here. This time, we will refer to the controller as the Game Master. The Game Master Agent will play two roles:

- Increment the step count
- Determine when the current game level has been finished

**Ask the students to create and draw the Game Master Agent:** Create a new agent called "Game\_Master", the same way you created the previous agents. Draw a picture you'd like to use to represent the Game Master.

#### Game Master Location

An interesting feature of the Game Master agent is that it does not need to be visible to the player, but it does need to be placed within our Worksheet at a specific location. To make it easier for us to see where we have placed this agent, it is a good idea to use some temporary or small depiction.

## Sokoban (Continued)

Place the Game Master Agent to the right of the zero in the Worksheet

0 \*

### Question for students:

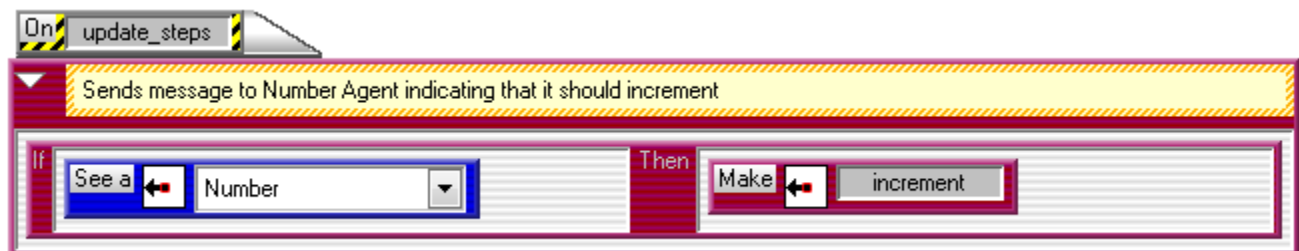
Why is it **very important** that a single Game Master agent is placed to the **Right** of the Number agent that we previously placed on the worksheet?

*Answer: Because there will be a digit going to the left of the zero when the count gets past nine. Also, because the code said look for the floor tile to the left...if we place the game master there, the code will no longer work.*

### Students will now need to program the following actions:

*When the Game Controller wants to update steps, he will look to see if there is a number to the left, and if there is, the increment method should activate.*

### Game Master Behavior: New Method, called `update_steps`



### Getting the Step Count to Increment

We now have the Game Master reacting to the "update\_steps" message and we have the Number reacting to "increment", but the numbers will still not increment when the Sokoban moves. To add this behavior we have to modify our Sokoban behaviors.

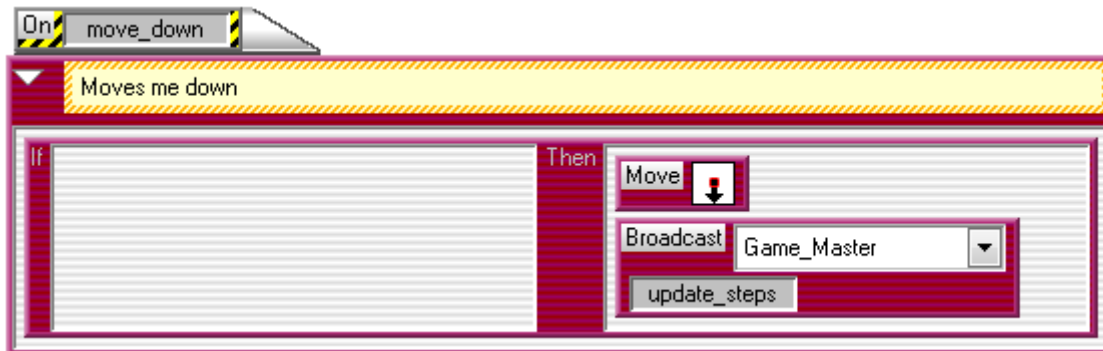
Every time the Sokoban moves it should send an "update\_steps" message to the Game Master letting it know that a movement has occurred.

Give the students a couple of minutes to talk through this problem...

## Sokoban (Continued)

How will they get the Sokoban agent to contact the Game Master agent.

The Sokoban agent will not be adjacent to the Game Master agent, so we can not use the "Make" action to send a message. Instead we will use the "Broadcast" action that sends a message to all Agents of a specified type and it does not matter where the receiving Agent is in relation to the sender.



Sokoban "DOWN" movement with Broadcast added

### Updating Other Sokoban Movements to Broadcast:

Now, add the same Broadcast action to the On Methods for "move\_up", "move\_right", and "move\_left".

The Sokoban needs to tell the Game Master that it has taken a step in two other cases: walking over Floor and Destinations. Add the same Broadcast action used previously to the eight rules for the other Sokoban movement. The picture below shows the updated rules for moving "DOWN" over the Floor and Destinations.



Sokoban "DOWN" movement with Broadcast added

**Provide students with Handout 7, found on page 11 of the STANDARD student packet, and page 16 of the ALTERNATIVE student packet.**



## Sokoban (Continued)

### Student Handout:

### Part 7 – Incrementing Numbers

#### Flashback to Journey

In this game, the Traveler had to collect multiple goals before winning the game. To determine if all the goals were gone, we created a Controller to poll the goals, which increased the count by one, for each goal remaining on the board.



STEPS

0 •

#### Step 1: Create a Game Master who will

- *Increment the step count*
- *Determine when the current game level has been finished*

#### Game Master Location

An interesting feature of the Game Master agent is that it does not need to be visible to the player, but it does need to be placed within our Worksheet at a specific location. To make it easier for us to see where we have placed this agent, it is a good idea to use some temporary or small depiction.

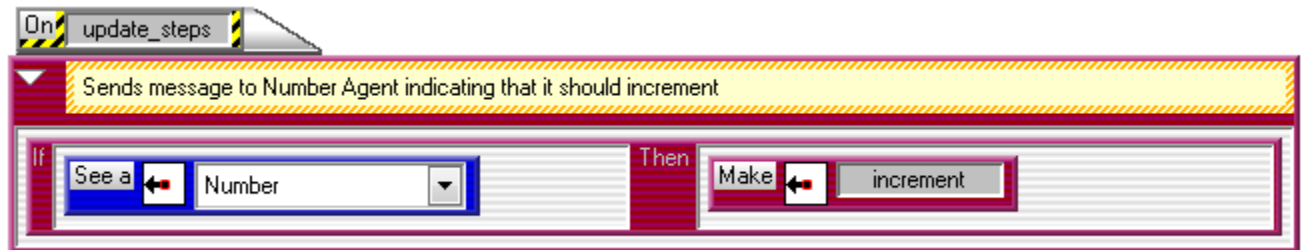
Place the Game Master Agent to the right of the zero in the Worksheet

0 •

*Step 2: When the Game Controller wants to update steps, he will look to see if there is a number to the left, and if there is, the increment method should activate.*

## Sokoban (Continued)

Game Master Behavior: New Method, called `update_steps`



*Step 3: Every time the Sokoban moves it should send an "update\_steps" message to the Game Master letting it know that a movement has occurred.*

Broadcasting Sokoban Steps:

Let's add...

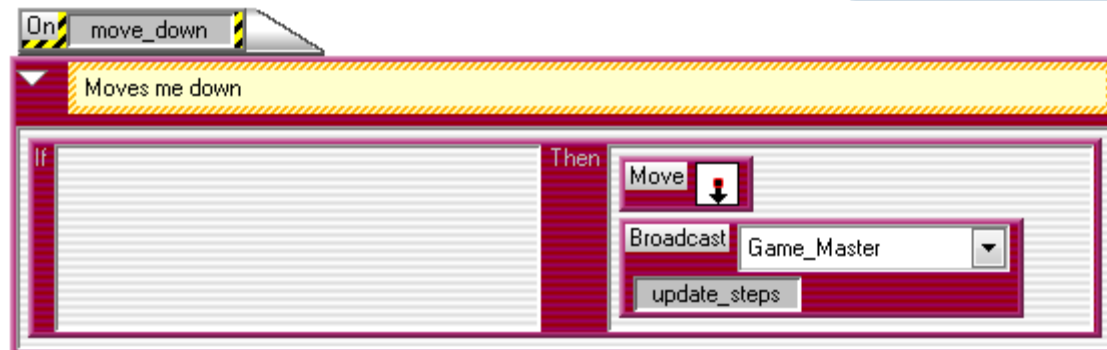
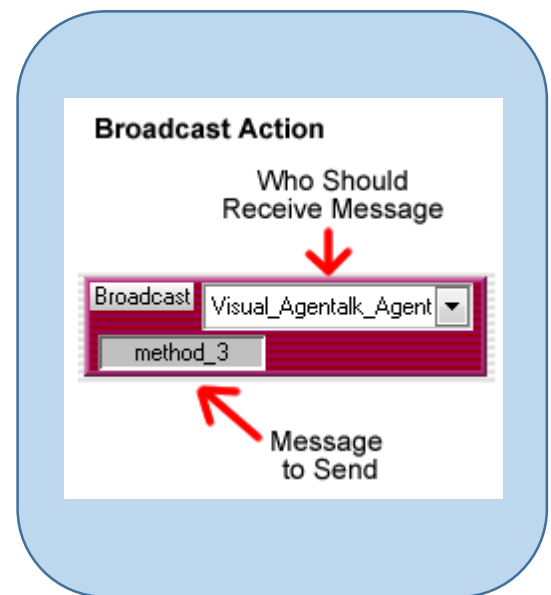
**the "Broadcast" action**

to the...

**Sokoban**

On...

**"move\_down" Method.**



*Sokoban "DOWN" movement with Broadcast added*

*Step 4: Now, add the same Broadcast action to the On Methods for "move\_up", "move\_right", and "move\_left".*

## Sokoban (Continued)

*Step 5: The Sokoban needs to tell the Game Master that it has taken a step in two other cases: walking over Floor and Destinations. The picture below shows the updated rules for moving "DOWN" over the Floor and Destinations.*



*Sokoban "DOWN" movement with Broadcast added*

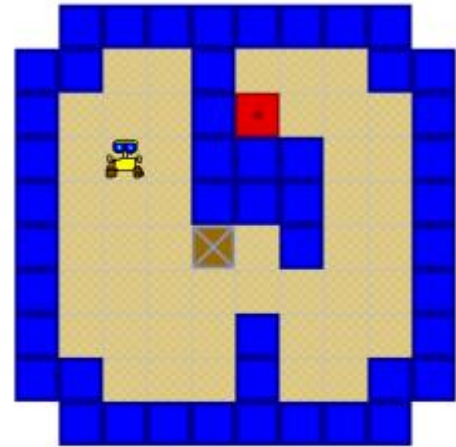
*Step 6: Add the same Broadcast action used previously to the eight rules for the other Sokoban movement.*

*Step 7: Test your game.*

- When the Sokoban moves or pushes a Crate does the step count increase?
- If you go over nine steps does the step count increase correctly?
- If the answer is "No" to either of the above questions, check the behaviors for problems and retest.
- If the step count is incrementing correctly (even if you had to change a few things and retest), you did a super job!

## Teacher Instructions:

### Part 8 – Winning the game



STEPS

0

Talk with the students at this point.

- How do we win the game? (when there are no more crates on floor tiles, you win)
- How do we know if we won the game (have to count all the crates left on the floor)
- When we check each time to see if we won the game, do we use the old count of crates, or do we start new? (we start new, with zero crates, and then we count up for each crate still on the floor).
- Do we need to count all time (continuously)? No, we can count a certain number of times per second

Now, go back through these same questions with the students and talk about the programming:

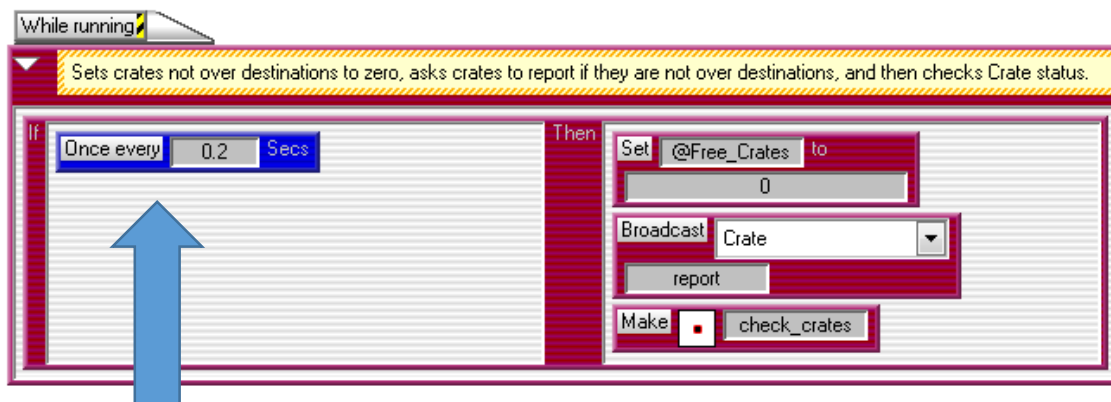
- How do we win the game? (when there are no more crates on floor tiles, you win)
  - We need a method to say if there are no more crates (`@Free_Crates = 0`) then you win.
  - Let's call that method "check\_crates"
- How do we know if we won the game (have to count all the crates left on the floor)
  - Set the number of crates to zero
  - Ask the crates if they are on a floor tile
  - If they are on a floor tile, increase the number of crates by 1
  - Let's call that method "report"

## Sokoban (Continued)

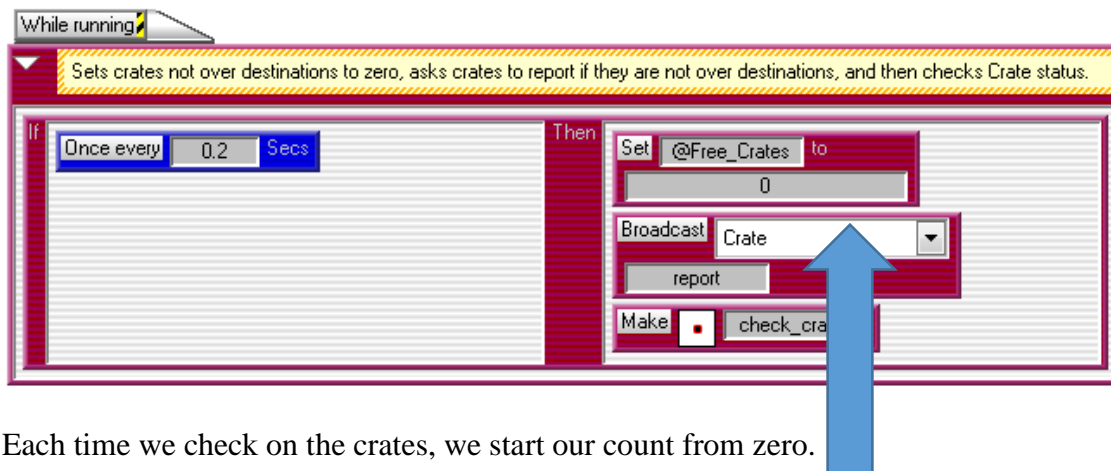
- When we check each time to see if we won the game, do we use the old count of crates, or do we start new? (we start new, with zero crates, and then we count up for each crate still on the floor).
  - Set the number of crates to zero
- Do we need to count all time (continuously)? No, we can count a certain number of times per second
  - Let's put in a condition that we will check every 0.2 seconds.

*During your discussion, students will likely come up with the code, at least in concept.*

### Game Manager Code with Explanation

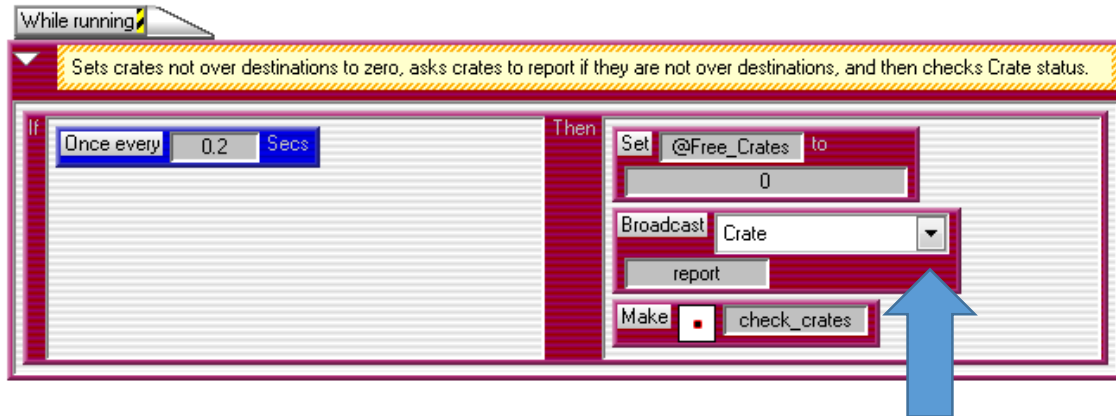


We don't need to check for crates continuously, so we will be it every so often. In this example, we did it once every 0.2 seconds, or five times a second.

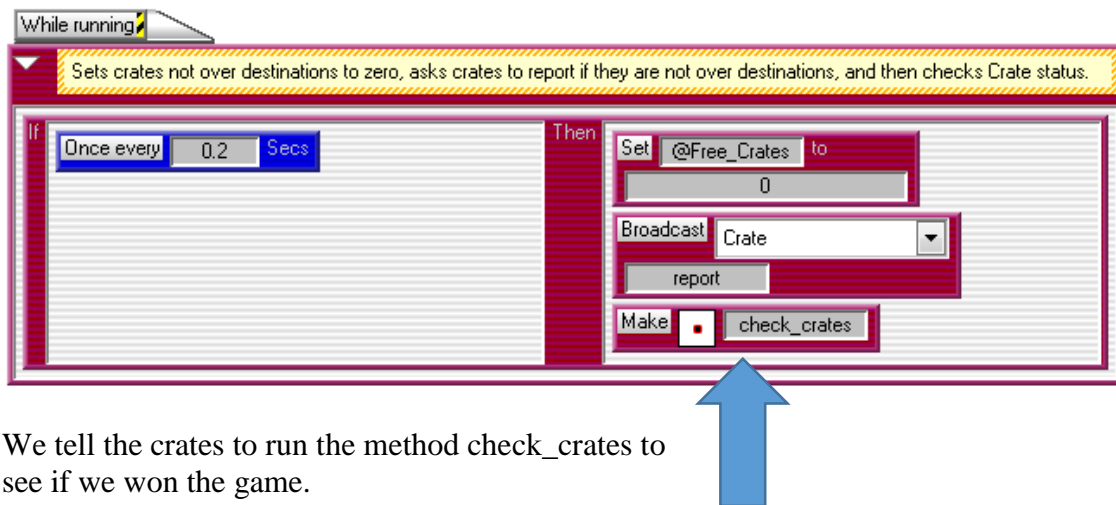


Each time we check on the crates, we start our count from zero.

## Sokoban (Continued)

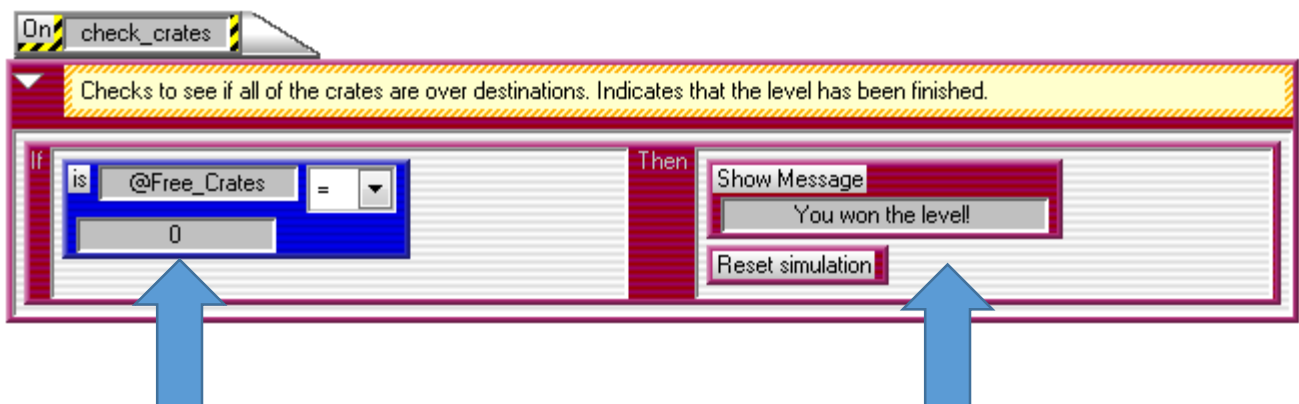


We send out a broadcast message to the crates, asking them to report if they are on the floor tiles.



We tell the crates to run the method `check_crates` to see if we won the game.

### Game Master's check-crates Method Code with Explanation



IF the crate count IS EQUAL TO zero

You win – reset simulation

## Sokoban (Continued)

### Crate's report Method Code with Explanation



IF the Crate is on the Floor Tile

Count it by adding a crate to the Free\_Crates count

By now, students have had significant practice with methods. Therefore, some will be more ready for a challenge than others. As a result, there are FOUR Student Handouts for this section.

- Student Handout 8A: Gives basic explanation based on the questions you discussed
  - Found on page 14 in the STANDARD student packet, and page 19 of the ALTERNATIVE student packet.
- Student Handout 8B: Explains the steps of each code, but no code is shown
  - Not included in the STANDARD student packet, and found on page 20 of the ALTERNATIVE student packet.
- Student Handout 8C: Gives the beginnings of each code, and an explanation
  - Not included in either student packet – have available for students who need it
- Student Handout 8D: Gives the full code
  - Not included in either student packet – have available for students who need it

Start with Handout 8A for all students. Have copies of the other three handouts available for those who need them. Students should not be made to feel badly if they need more help – that is all part of the learning process.

## Student Handout

### Part 8A – Winning the game

#### *Challenge yourself to do it on your own.*

To finish programming your game, answer each question and produce the code:

- How do we win the game?
  - Create a METHOD for the GAME MASTER that shows when you win.
- How do we know if we won the game
  - Create a METHOD for the CRATES that tells you if they are still on the floor.
- When we check each time to see if we won the game, do we use the old count of crates, or do we start new?
  - Be sure your METHOD for the crates starts with the number of crates equal to zero
- Do we need to count all time (continuously)?
  - WHILE RUNNING, your Game Master should check to see if you won every 0.2 seconds.

Press Run and see if everything works correctly. Check

- When the Sokoban moves does the step count increment correctly?
- When the Sokoban pushes the Crate does the step count increment correctly?
- If you push all Crates over all Destinations does the level end?

If your answer to one of these is no, ask for Student Handout 8B for more hints. Note: There should always be one Destination per Crate in the game levels.

Otherwise, if everything works correctly GREAT JOB! You have finished your own Sokoban game! You can now go back and make any changes you want to make (like redrawing an agent, redesigning your game level, adding other behaviors, etc.).



## Student Handout

### Part 8B – Winning the game

#### Need more help? That's okay – here are some more hints:

We need a method to say if there are no more crates (set a simulation property @Free\_Crates = 0) then you win, and reset the game.

- Let's call that method "check\_crates"

We need a method to know if we won the game (have to count all the crates left on the floor)

- Set the number of crates to zero
- Ask the crates if they are on a floor tile
- If they are on a floor tile, increase the number of crates by 1
- Let's call that method "report"

Let's tell the Game Master that While Running, every 0.2 seconds, he should SET the number of crates (@Free Crates) to zero, he should then BROADCAST to the Crates, asking for a "report", and finally, he should "check\_crates" to see if you won.

Press Run and see if everything works correctly. Check

- When the Sokoban moves does the step count increment correctly?
- When the Sokoban pushes the Crate does the step count increment correctly?
- If you push all Crates over all Destinations does the level end?

If your answer to one of these is no, ask for Student Handout 8C for more hints. Note: There should always be one Destination per Crate in the game levels.

Otherwise, if everything works correctly GREAT JOB! You have finished your own Sokoban game! You can now go back and make any changes you want to make (like redrawing an agent, redesigning your game level, adding other behaviors, etc.).

## Student Handout

### Part 8C – Winning the game

**Need even more help? That's still okay – here is the piece of each code with explanations.**

#### Game Manager Code with Explanation

**While running**

Sets crates not over destinations to zero, asks crates to report if they are not over destinations, and then checks Crate status.

Every 0.2 seconds	set the crates to zero Ask the crates to REPORT back Run CHECK_CRATES to see if the game is over.
-------------------	--

#### Game Master's check-crates Method Code with Explanation

**Un check\_crates**

Checks to see if all of the crates are over destinations. Indicates that the level has been finished.

IF the crate count IS EQUAL TO zero	You win – reset simulation
-------------------------------------	----------------------------

#### Crate's report Method Code with Explanation

**On report**

Update amount of crates not on a destination

IF the Crate is on the Floor Tile count	Count it by adding a crate to the @Free_Crates
--	--

Press Run and see if everything works correctly. Check

- When the Sokoban moves does the step count increment correctly?
- When the Sokoban pushes the Crate does the step count increment correctly?
- If you push all Crates over all Destinations does the level end?

If your answer to one of these is no, ask for Student Handout 8D for more hints. Note: There should always be one Destination per Crate in the game levels.

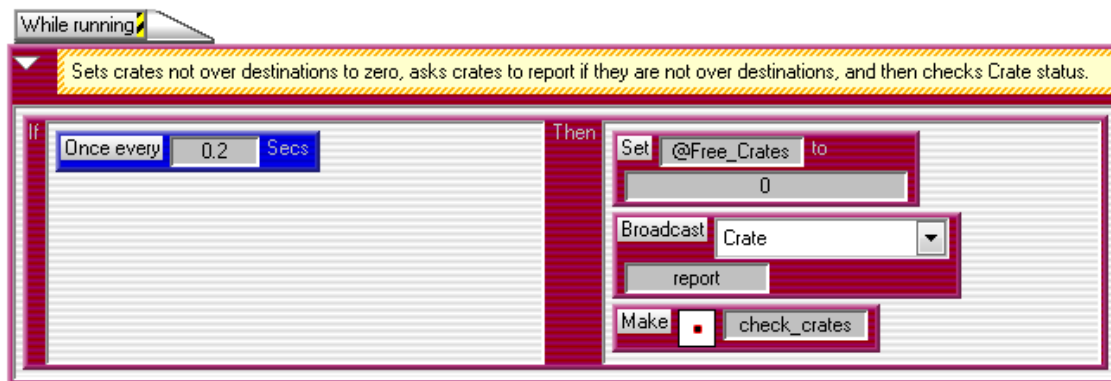
Otherwise, if everything works correctly GREAT JOB! You have finished your own Sokoban game! You can now go back and make any changes you want to make (like redrawing an agent, redesigning your game level, adding other behaviors, etc.).

## Student Handout

### Part 8D – Winning the game

Need even more help? That’s still okay – here is the code with explanations. Be sure to talk with a friend if you still have questions.

#### Game Manager Code



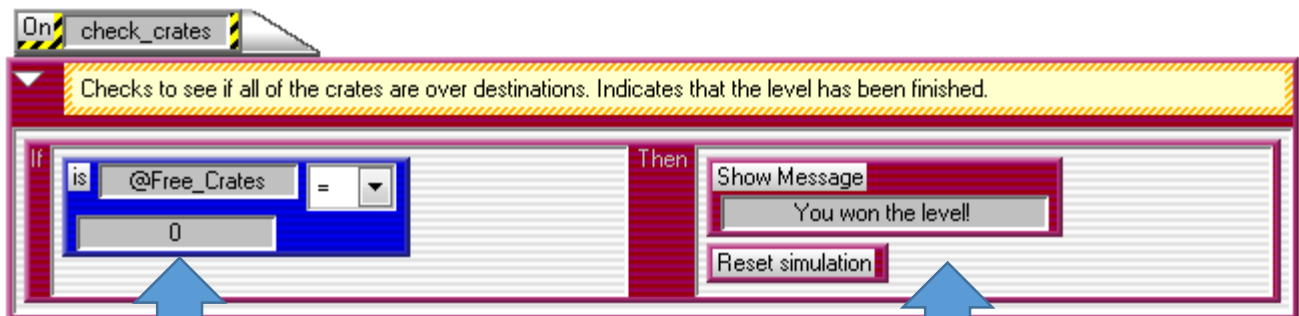
Every 0.2 seconds

set the crates to zero

Ask the crates to REPORT back

Run CHECK\_CRATES to see if the game is over.

#### Game Master’s check-crates Method Code with Explanation



IF the crate count IS EQUAL TO zero

You win – reset simulation

#### Crate’s report Method Code with Explanation

## Sokoban (Continued)



IF the Crate is on the Floor Tile

Count it by adding a crate to the @Free\_Crates count

Press Run and see if everything works correctly. Check

- When the Sokoban moves does the step count increment correctly?
- When the Sokoban pushes the Crate does the step count increment correctly?
- If you push all Crates over all Destinations does the level end?

If your answer to one of these is no, go back to the related section and see what you might have done wrong. Note: There should always be one Destination per Crate in the game levels.

Otherwise, if everything works correctly GREAT JOB! You have finished your own Sokoban game! You can now go back and make any changes you want to make (like redrawing an agent, redesigning your game level, adding other behaviors, etc.).

## End of Unit Review Sheet - Sokoban

- A) The main computational thinking patterns we reviewed were:
- 1) **Cursor Control**: intentionally moving an agent.
    - a. Using keyboard keys to move an agent.
    - b. Example is moving the Sokoban.
  - 2) **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. Example: Winning the game by placing the crate on the destination.
  - 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 crates to see if they are on the destination.
- B) The main NEW computational thinking patterns we learned were:
- 1) **Push**: moving an object and then telling the agent doing the pushing to move as well.
    - a. Example: The Sokoban pushed the crate.
- C) Other concepts we covered in AgentSheets are:
- 1) Incrementing Numbers
    - a. Thinking about how numbers change
    - b. Learning what’s special about the digit 9
  - 2) Using Incrementing Numbers to count steps
  - 3) Calling methods to do special tasks
  - 4) Troubleshooting the simulation, and considering rule order.
  - 5) Using sounds and messages in the game.
  - 6) Timing our actions using the “Once every” condition.

## ISTE Standards<sup>3</sup> specific to the implementation of Sokoban (Denoted with (★))

### Creativity and Innovation

*Students demonstrate creative thinking, construct knowledge, and develop innovative products and processes using technology. Students:*

#### Apply existing knowledge to generate new ideas, products, or processes:

- ★ Design and develop games
- Design and develop computational science models

#### Create original works as a means of personal or group expression.

- ★ Design original games
- Model your local environment, e.g., ecology, economy

#### Use models and simulations to explore complete systems and issues.

- Model scientific phenomena, e.g., predator / prey models
- Create visualizations

#### Identify trends and forecast possibilities.

- Build predictive computational science models, e.g., how the pine beetle destroys the Colorado pine forest
- Build live feeds to scientific web pages (e.g. weather information), process and visualize changing information

### Communication and Collaboration

*Students use digital media and environments to communicate and work collaboratively, including at a distance, to support individual learning and contribute to the learning of others. Students:*

#### Interact, collaborate, and publish with peers, experts, or others employing a variety of digital environments and media:

- ★ Students work in teams to build and publish their simulations as web pages containing java applets.

#### Communicate information and ideas effectively to multiple audiences using a variety of media and formats.

- Effectively combine interactive simulations, text, images in web pages

#### Develop cultural understanding and global awareness by engaging with learners of other cultures.

- ★ Students and teachers from the four culturally diverse regions interact with each other

#### Contribute to project teams to produce original works or solve problems.

- ★ Define project roles and work collaboratively to produce games and simulations

### Research and Information Fluency

---

<sup>3</sup> ISTE Standards for Students (ISTE Standards•S) are the “standards for evaluating the skills and knowledge students need to learn effectively and live productively in an increasingly global and digital world.” <http://www.iste.org/standards/standards-for-students>

# Sokoban (Continued)

*Students apply digital tools to gather, evaluate, and use information. Students:*

**Plan strategies to guide inquiry.**

Explore web sites and identify interesting connections

**Locate, organize, analyze, evaluate, synthesize, and ethically use information from a variety of sources and media.**

Find relevant related web-based information, compute derivative information

**Evaluate and select information sources and digital tools based on the appropriateness to specific tasks.**

Understand validity of information, e.g. Scientific journal information vs. Personal blogs

**Process data and report results.**

Write programs to access numerical information, define functions to process data and create output based on voice or plotting to represent data.

**Critical Thinking, Problem Solving, and Decision Making**

*Students use critical thinking skills to plan and conduct research, manage projects, solve problems, and make informed decisions using appropriate digital tools and resources. Students:*

**Identify and define authentic problems and significant questions for investigation.**

Define research questions and explore approach of exploration

**Plan and manage activities to develop a solution or complete a project.**

- \* Outline sequence of exploratory steps
- \* Experience complete bottom-up and top-down design processes
- \* Employ algorithmic thinking for creating programs to solve problems

**Collect and analyze data to identify solutions and/or make informed decisions.**

Collect data as time series, e.g., collect group size of predator and prey, export time series to excel, explore various types of graph representations, e.g.,  $x(t)$ ,  $y(t)$  or scatter  $y=f(x)$

**Use multiple processes and diverse perspectives to explore alternative solutions.**

- \* Experience and understand design trade-offs, e.g. Bottom-up vs. Top-down

**Digital  
Citizenship**

*Students understand human, cultural, and societal issues related to technology and practice legal and ethical behavior. Students:*

**Advocate and practice safe, legal, and responsible use of information and technology.**

- \* Learn how to use tools to locate resources, e.g., images with google image search, but understand copyright issues

**Exhibit a positive attitude toward using technology that supports collaboration, learning, and productivity.**

- \* Stay in the flow, where design challenges match design skills
- \* Experience success through scaffolded game design activities
- \* Mentor other students

# Sokoban (Continued)

## **Demonstrate personal responsibility for lifelong learning.**

- \* Explore options of going beyond expected learning goals

## **Exhibit leadership for digital citizenship.**

- \* In a collaborative setting become a responsible producer of content for diverse audiences

## **Technology Operations and Concepts**

*Students demonstrate a sound understanding of technology concepts, systems, and operations. Students:*

### **Understand and use technology systems.**

- \* Know how to organize files and folders, launch and use applications on various platforms

### **Select and use applications effectively and productively.**

- \* Know how to orchestrate a set of applications to achieve goals, e.g., make game and simulations using Photoshop (art), AgentSheets (programming), and Excel (data analysis).

### **Troubleshoot systems and applications.**

- \* Debug games and simulations that are not working

### **Transfer current knowledge to learning of new technologies.**

- \* Reflect on fundamental skills at conceptual level. Explore different tools to achieve similar objectives.