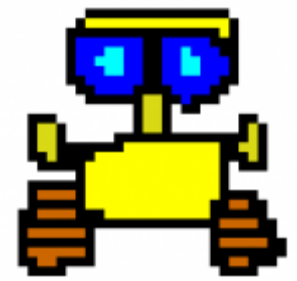


Creating "Sokoban"

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.

.

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**

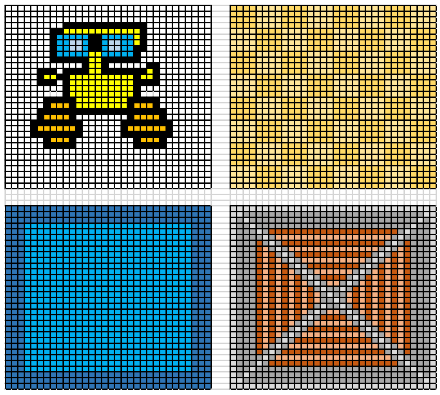
Student Handout 1A:

Part 1 – Create Worksheet and Agents

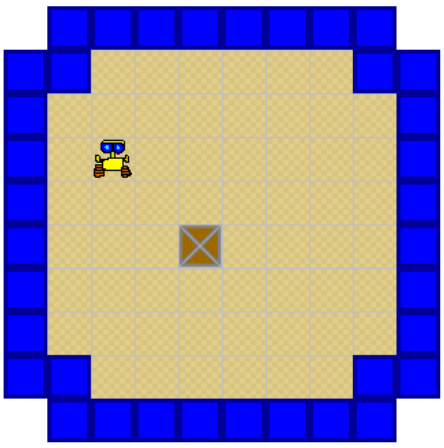
In this project, you will create a worksheet with a Sokoban. This Sokoban will be tasked with pushing a crate to a specific destination. Since you have already created a prior game, these instructions will be less specific. If you are stuck, talk with the person next to you about ways to correct the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

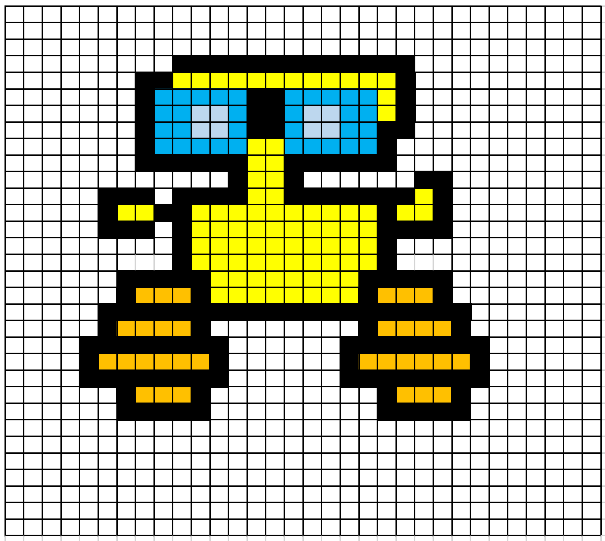


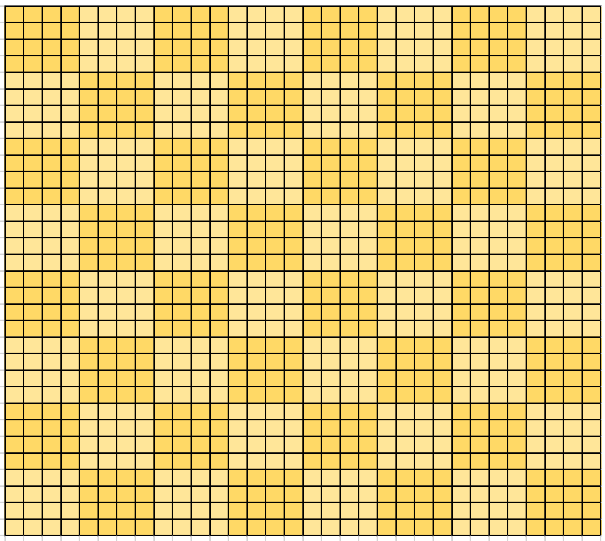
4. Enable the Sokoban to be cursor controlled, such that it moves up/down/right/left with the arrow keys.

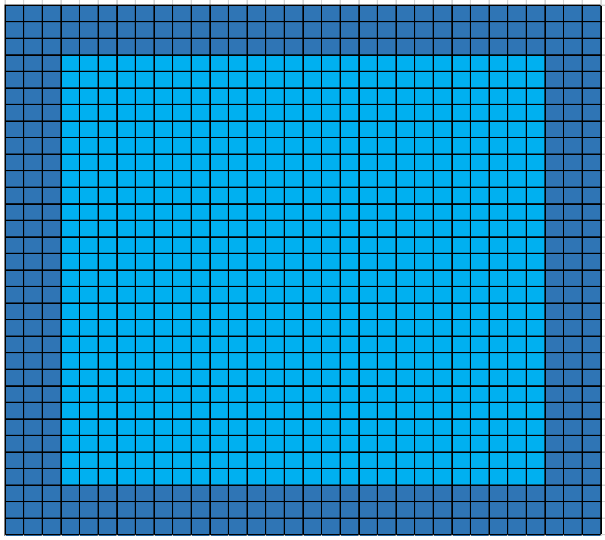
5. Prevent the Sokoban from walking through walls.

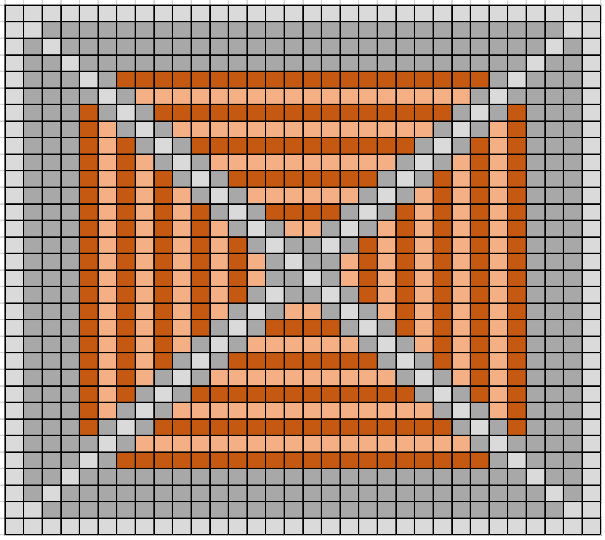
Student Handout 1C:
Agent Creation Models

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





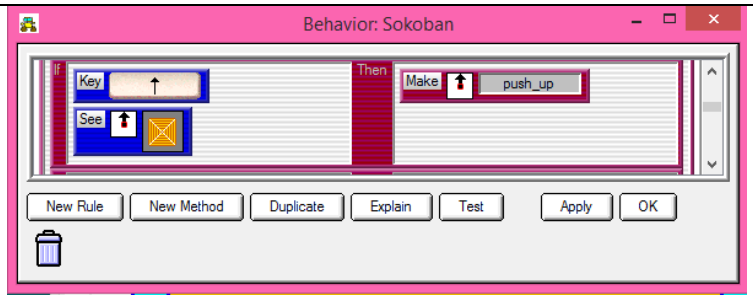
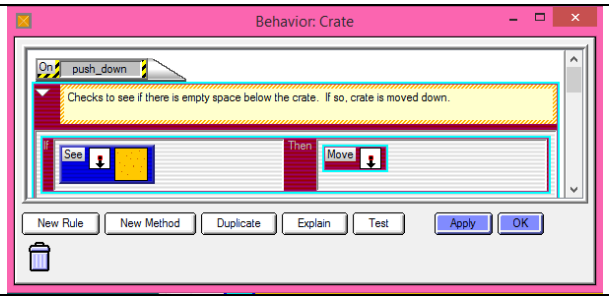
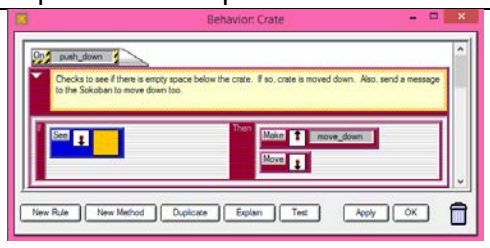




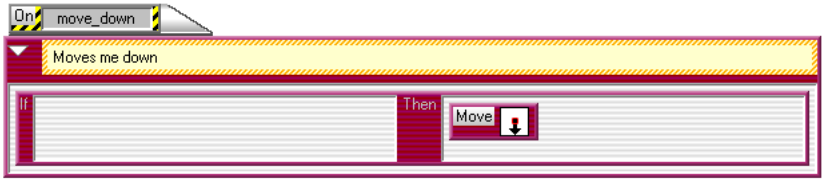
Student Handout 4:

Part 4 – Programming the Sokoban to Push Crates

Click on the agent to add behaviors to that agent

Step 1	**Enable the Sokoban to push the crates.** **This is the behaviors in the UP direction for the Sokoban. Code the rest of the directions.**	
Step 2:	**Create the Method push_down for the CRATE** **Reminder: click New Method**	
Step 3:	**Create remaining methods for push_up, push_left and push_right**	No hints here!
Step 4:	**Test your game**	You should not get any error messages. Your crate should move in the proper direction. Does your Sokoban move? Why not? 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.
Step 5:	**Change the Sokoban rules so that your Sokoban moves down when the crate is pushed down.**	 Crate checks to see if a Floor tile agent is below. If there is a Floor tile agent below the crate, it sends a message back to the Sokoban telling it to move down and then the crate moves down.

Sokoban (Continued)

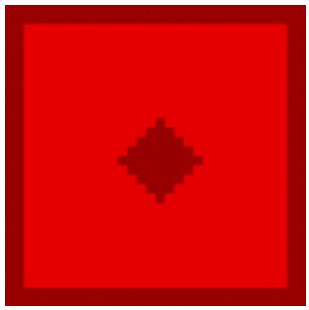
Step 6:	**Test your game**	The Sokoban does not know how to react to the message (move_down) that it is receiving back from the crate; therefore, you will see an error message from AgentSheets when we run our game now!
Step 7:	**Create the Method move_down for the SOKOBAN**	
Step 8:	**Change the Sokoban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

Student Handout 5: The Destination

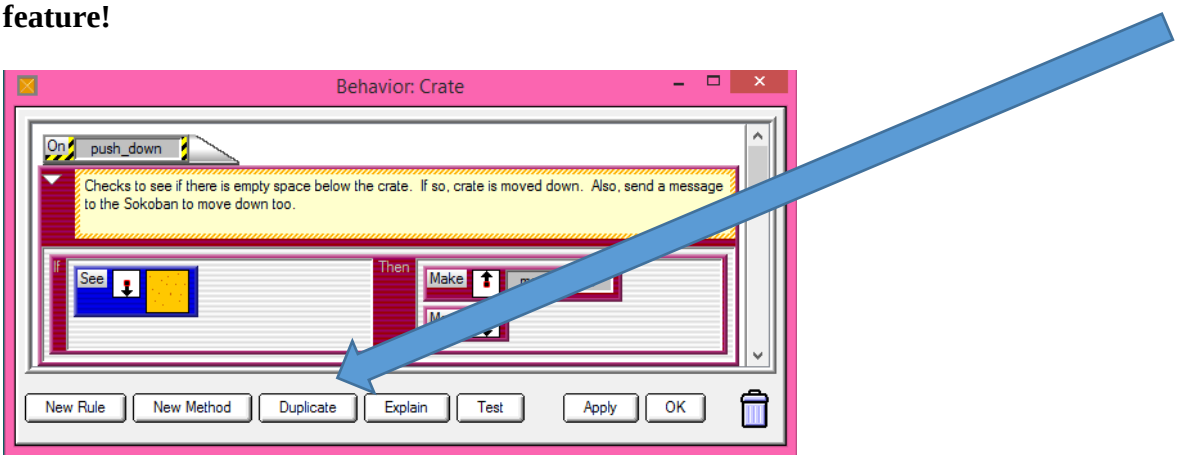
You are tasked with creating the destination tile for Sokoban. Here are the rules:



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!

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**

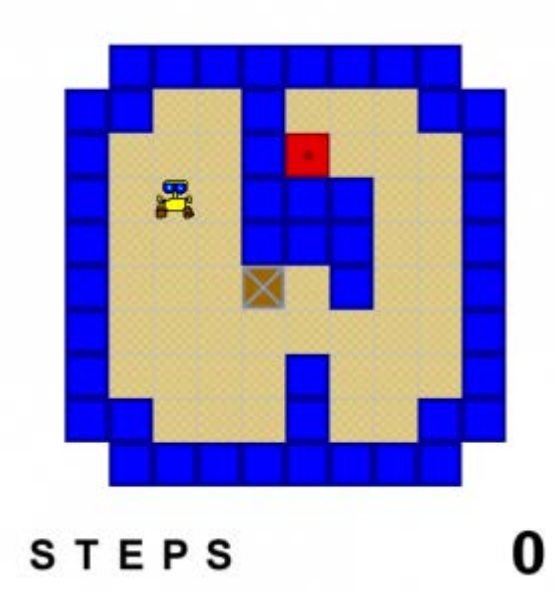
Step 2: Create a number agent

- **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.

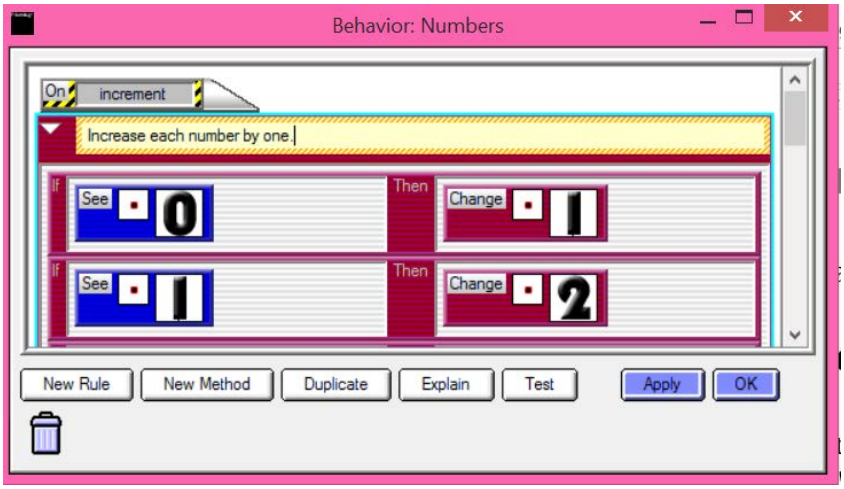


Sokoban (Continued)

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

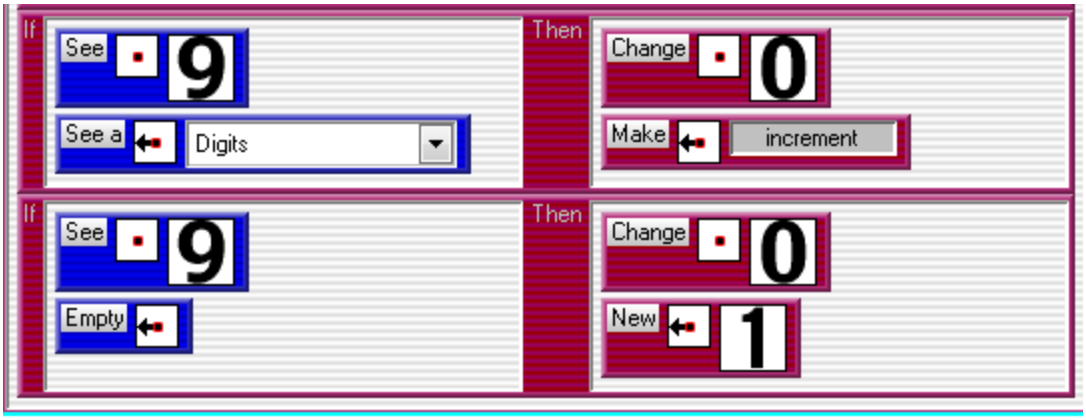
Step 5: Add Rules to Numbers Agent "increment" Method:

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!

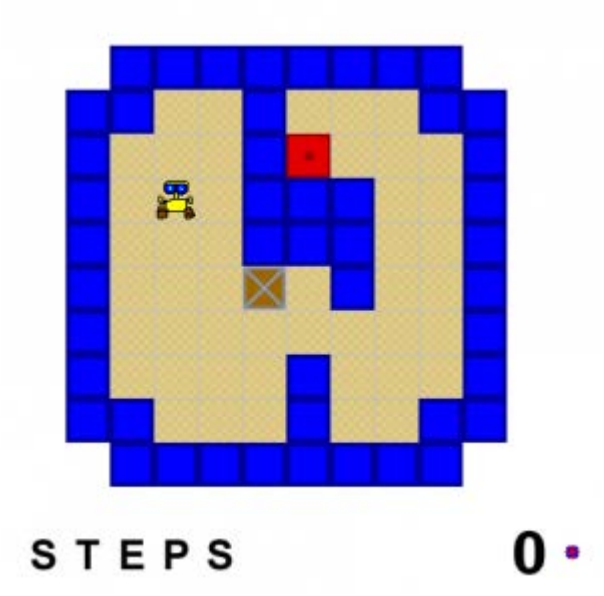
***Warning:** If your counter is not on empty space (i.e. on the floor, wall, etc.) you want to make sure that the last "if" is not "empty to the left" but rather "sees floor to the left" or whatever you have the counter on....*

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!

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.

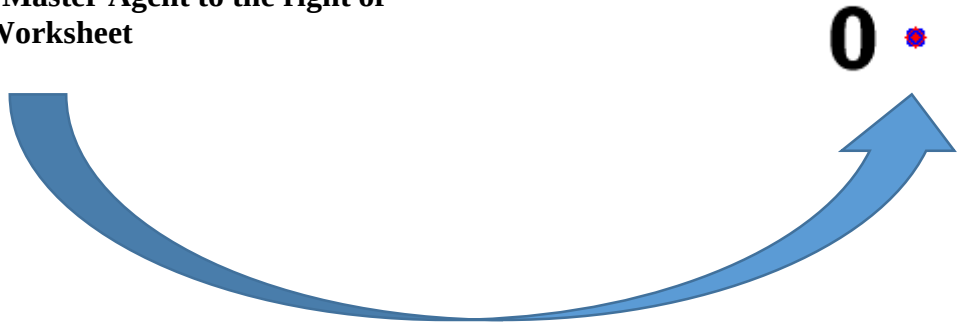
Step 1: Create a Game Master who will

- *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.

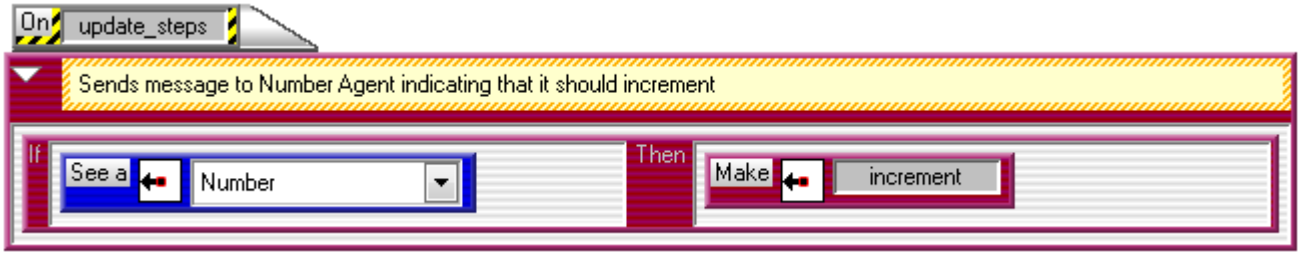
Place the Game Master Agent to the right of the zero in the Worksheet



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…

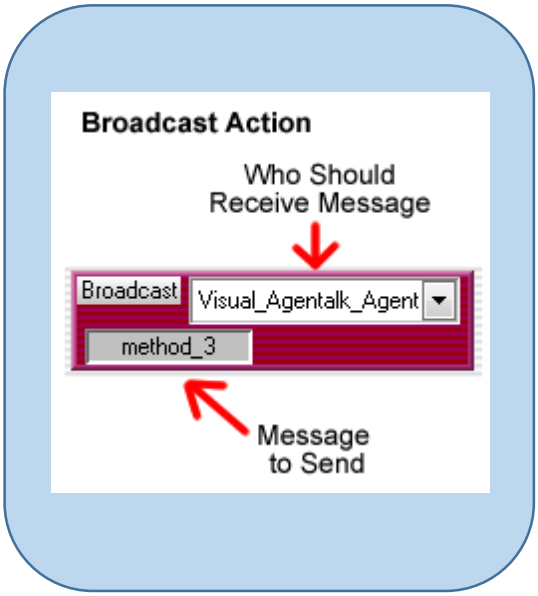
the "Broadcast" action

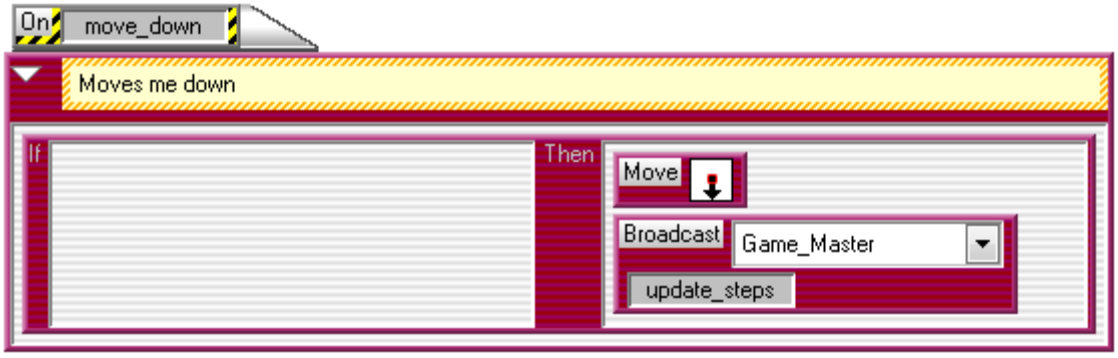
to the…

Sokoban

On…

"move_down" Method.



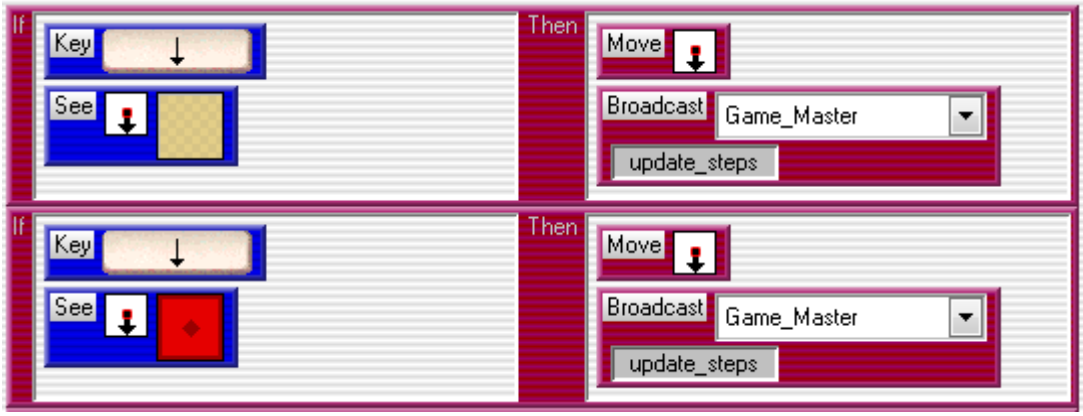


Sokoban "DOWN" movement with Broadcast added

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!

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.).

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.