Debugging tips and techniques for AgentCubes projects

There is no one specific step-by-step approach to debugging in general, nor in addressing problems with AgentCubes projects specifically. However, there are some general guidelines one can use in approaching problems that can well serve the debugging process.

- What are you trying to do? It is important for the user to be able to explain what the intended behavior is, rather than simply describe what is happening that is not desired.
- Read the appropriate agent behavior; try to understand what it says before running the simulation. Computers do what they're instructed, which may differ from what is desired. Computers read code, not minds!
- Reset the world before exploring. Start from "ground zero" when investigating a problem. Failing to do this is analogous to entering a theater half-way into the film: you're likely to have missed some critical elements.
- Close all behavior windows (look for minimized windows); this will indicate where there are behavior changes that have not been made permanent. One cause of unexplained behavior is that the user made changes to it, but failed to apply the changes by clicking on "apply" or "OK" before running the simulation.
- If "mysterious" agents appear, check the world by removing one layer at a time to determine if there are hidden agents. A common error in constructing worlds is to treat the world like a paint canvas and the agent placement tools like a paintbrush. Each layer of a world is, in itself, a layered structure and the placement tools will place agents on top of each other so that the agents underneath may not be visible. For most shapes, such placement may be obvious if, for example, the world has small dimensions and the objects have large 3D configurations, such as cubes. However, tile shapes are flat and can be placed undetectably beneath other shapes. Teach students to use the multi-agent (rectangle) tool for placing rows, columns, or blocks of agents, and the pencil tool for placing individual agents.
- AgentCubes permits orienting the world in ways that AgentSheets does not. For example, one can turn the world on its side. Beginning students sometimes flip the world upside down or turn it 180 degrees, and forget to reload it before populating it with agents. Debugging exercise 'Problem 10' shows examples of this. The consequence is agent movement is backwards from what is expected, particularly with respect to using keys (e.g., arrow keys) to direct agent movement: the 'up' key moves the agent down or the 'left' key moves it to the right, etc. Unfortunately, the only resolution is to clear the world and re-build it.
- AgentCubes differs from AgentSheets when managing changes to agent behavior. In AgentSheets, changes to agent behavior are not made permanent until the user accepts them by tapping the 'OK' or 'Apply' buttons in the agent behavior editing window. At that time, AgentSheets stops the simulation, applies the changes to the agent behavior both in the simulation in memory as well as on the hard drive. This also requires the user to select the 'play' button in the worksheet in order to resume simulation. In contrast, AgentCubes permanently applies each change to agent behavior, both in memory and on the hard drive, as the user enters it in the agent behavior editing window, *without* stopping the simulation. The upside of this approach is that the user can immediately see the effect of the change in the simulation, which can be useful in both testing and running simulations. The downside is that if the user is not careful in applying or removing conditions, the change can cause unintended consequences ranging from loops of sound or messages to changes in the world. In order to minimize these kinds of problems, it is a good idea to teach the technique of always *stopping the simulation and reloading the world before making any changes to agent behavior or shapes*.
- Most error messages generated by AgentCubes are not problems in the sense of AgentCubes malfunction. Rather, they indicate user programming errors, such as failing to create a method in an agent that receives a message. However, most of the AgentCubes error messages are also somewhat

lengthy and may be confusing to beginning users. It is important for users to read an error message completely to understand the problem being reported. If an error message cannot be understood and the user suspects a problem, it is helpful when reporting it to capture an image of it via a standard print screen or print window function on the computer.

- Rule order can be critical: Look for rules with no conditions. These rules will be performed unconditionally, which means any rules below such a rule will never be evaluated for execution. In general, rules that deal with "unusual" situations, such as game-ending rules, should be placed highest in the behavior list. As in life, first things first.
- Condition order can be critical. Normally, since all conditions of a rule must be true, one might conclude that their order is not important, just as the order of selection of a sports team is not important; rather only the final team composition is important. But there are situations in which this is not the case in AgentCubes. The most illustrative example is one in which there is a timing condition and a probability condition. For example, if the conditions are
    - Once every 1 second, with a 50% chance: This rule will be performed on the average about every 2 seconds.
    - However, reversing the conditions (with a 50% chance, once every 1 second), the rule will be performed every second.
    - The difference is due to how AgentCubes evaluates rules in the "While Running" method: every agent is scanned sequentially and then the cycle is repeated as fast as the computer can perform – hundreds of times per second or more. Since the 50% chance condition will occur about every other cycle, this means that the 50% chance will be true about every other pass through the agents. This is much more frequent than every second, so when the 1-second timer occurs, there is certain to be a 50% chance success within a fraction of a second of that event.
- Another twist regarding the 'once every' condition is the use of multiple rules with this condition in the While Running method. This requires a somewhat subtle understanding of how this condition is evaluated. Unlike some conditions which are relatively static, such as the shape of a nearby agent (for example, in the 'see' condition), the 'once every' condition is an event. When the event is evaluated in a rule, it is no longer true for evaluation in subsequent rules in the same simulation cycle. As a consequence, if there are two rules with this condition, and if the condition is currently true, and if the first rule has a second condition which is not true, then the 'once every' event is no longer true when the second rule with the 'once every' condition is evaluated. An example of this is illustrated in the debugging exercises, Problem 07, the 'Clock Problem'. This is a common problem for students with projects requiring, for example, alternating actions periodically. There are several ways to solve the problem.
- Use conversational programming! This is a unique tool among software development environments and a very powerful tool, as well. Use conversational programming to determine which conditions are satisfied and whether associated rules will be performed as expected.
- Examine agent attributes and simulation properties, as appropriate.
- Use the single step button. Watch behavior unfold incrementally, rather than trying to keep track of what happens in real time.
- Avoid "playing the game" to examine a specific event. Rather, move agents into appropriate positions
- Remove agents not involved in the problem to permit examining other agents without interference. For example, if there are six agents pursuing a single agent, and the problem concerns whether hill-climbing is working properly, temporarily remove all but one of the tracking agents so that you can focus on one agent at a time.
- Make a new world with only the agents involved in the problem. Sometimes, simply removing "extraneous" agents may not permit understanding the problem, especially if there are buried agents, or possibly a corrupted world. Starting "from scratch" with a simplified world can help address whether the problem is with the world construction or agent behavior.

- For diffusion, check the diffusion formula carefully. Usually, a mis-entered formula will cause an error message and the input rejected, but not always. Sometimes the formula will pass syntax checking but not be correct semantically.
- For hill-climbing, If not using the single action for hill-climbing, be sure to check the rules in the method for choosing the direction in which to move carefully. As with the diffusion formula, the code may be syntactically correct but not semantically.
- Check spelling of agent attributes and simulation properties. Note that these are created dynamically if not created deliberately. For example, if the user defines a simulation property named "total" (properly referenced as "@count"), but then mis-types the name as "@t0tal" or omits the "@" symbol, the simulation will run and dynamically create the additional name, but the behavior may not work as desired. Once an attribute or simulation property is used – either explicitly via a 'set' action or implicitly when referenced in either a 'test' or 'set' operand, it is defined for that project. Note that if two projects are opened concurrently, the combined set of attribute and simulation property names will exist. This can result in confusion when selecting a name in a 'set' action.
- AgentCubes Release 2 names for agents, attributes, methods, etc. must begin with a letter and contain only letters, numbers, the hyphen and the underscore characters. AgentCubes validates these names as they are defined when creating a new agent, a new agent shape, or a variable (agent attribute or simulation property), or method name. It is possible, however, to import an AgentSheets project directly into AgentCubes, and AgentSheets names had fewer naming restrictions. This can cause problems in the conversion process and unpredictable results in the converted projects. Therefore, before importing an AgentSheets project into AgentCubes, it is helpful to examine the names used in the AgentSheets project.
- As with naming conventions, AgentSheets has several features, such as conditions and actions, that are not supported in AgentCubes. These include communication with websites, the direction and set direction functions, and several others. Attempting to import projects with these features can cause unpredictable results. AgentCubes attempts to change the actions or conditions during the conversion process, and to indicate to the user with warning messages. However, it is up to the user to address these issues in the converted process. It is sometimes easier to make and change a separate copy the AgentSheets project to remove these usages before attempting the conversion.
- Switching worlds: Navigate to the desired world to switch after clicking on the "?" or existing world file name. Do not change the folder: all worlds for a project must be in the 'Worlds' folder of that project folder.