Nifty Assignment: Model the Monte Hall Problem using Event-Driven Programming

Andrew A. Anda
Computer Science Department
Saint Cloud State University
Saint Cloud, MN, 56301
aanda@stcloudstate.edu

Abstract

We describe how to guide students through the modeling of the Monty Hall problem. The Monty Hall problem can be modeled in a CS0 or CS1 programming and problem solving course by a student using only the following tools: an event-driven programming language such as JavaScript, if-else syntax, an ability to use the rand() function to generate pseudo-random integer values in a range, and a basic ability to apply functional decomposition to program design. The use of the mod (%) operator can be taught and exploited at this time to simplify the logic at a critical point. Rather than start from scratch, the students reverse engineer an existing applet for which the Java source is not visible.

1 Pedagogical Motivations

Piaget stated that developmental learning is driven primarily by a process of *equilibration* following a *disequilibrating* experience, an experience that challenges assumptions. He suggested that designers of interactive experiences design tasks that both attract learners and provoke disequilibration. Experience in which learners construct a working physical arrangement significantly impact knowledge construction. (Roschelle, 1995)

Few experiences are more disequilibrating than wrestling with a paradox. A class of paradoxes which lend themselves to verification through computational modeling are *veridical* paradoxes - paradoxes where the result appears counterintuitive, but can nonetheless be demonstrated. (Cucić, 2009) Commonly known veridical paradoxes which admit verification through computational modeling include: (political representation system) apportionment paradoxes, the birthday paradox (probability that some pair within a larger set of people will share a birthday), and the "Let's Make a Deal" paradox (also termed the Monty Hall problem). Both the birthday paradox and the Monty Hall problem are members of a larger class of probabilistic paradoxes which can all be demonstrated through discrete stochastic simulation.

Peter Denning (2003) identifies *modeling and validation* as one of the five main categories of computing practice. Any programming assignment which includes modeling and validation strengthens and broadens a student's perceptions regarding the utility of programming as a problem solving methodology and utility.

After a student models a veridical paradox, and after the student proceeds to test the model to confirm the counterintuitive solution, the student gains an enhanced appreciation for the value of computational modeling. (as well as reinforcing recently learned programming language components and algorithmic heuristics)

2 The Assignment

2.1 The Problem to be Solved (Modeled)

Students model, through *reverse engineering* of an existing Java applet, the Monty Hall problem. ('Wikipedia', *The Monty Hall problem*). The Monty Hall problem refers to a puzzle based on the U.S. televised game show, *Let's Make a Deal*. The player is presented with three doors and informed that there is a desirable prize behind one of the doors and booby prizes (often represented as a goat) behind the other two doors. When the player selects a door, one of the doors not selected is opened revealing a booby prize. The player is then presented with the option of staying with their selection or switching their selection to the other remaining unopened door.

To facilitate the students' modeling process, students will be presented with a pre-existing computational event-driven emulation (West, 1996) of the game which they will be expected to analyze (via state tables and/or state diagrams) then reverse engineer and emulate, As the existing emulation is based on a Java applet, the students will not have access to its Java source code.

2.1.1 Problem Analysis and Strategy

The common naïve intuitive assumption is that there will be an equal probability of winning (1/3) regardless of whether one stays with one's choice or switches. A more careful analysis proves that one doubles one's probability of winning (2/3) upon switching. Because this solution is so counterintuitive, it can be classified as a veridical paradox.

2.2 Prerequisites

Level: CS0 – CS1
Students should be able to apply the following structured programming language components: an event-driven programming language such as JavaScript, `if-else` logic, an ability to use the `rand()` function to generate pseudo-random integer values in a specified range, and a basic ability to apply functional decomposition to program design. The use of the mod (%) operator can be taught and exploited at this time to simplify the logic at a critical point (to facilitate the random selection of a booby prize door to open). Note, that loops are not required if JavaScript or a comparable event-driven programming language is used.

Using the event-driven features of JavaScript, the students will be able to perform the modeling of the game, so students should have had prior experience with event-driven program design (e.g. clicking on the image of one face of a six-sided die to "roll" the die).

2.3 Lesson Plan

To facilitate the students' modeling process, students will be presented with a pre-existing computational event-driven emulation (West, 1996) of the game which they will be expected to analyze (via state tables and/or state diagrams), reverse engineer and emulate) As the existing emulation is based on a Java applet, the students will not have access to its Java source code.

First, acculturate the students to the game applet by demonstrating the playing of a few rounds. If the students all have terminals, they can explore the applet themselves. Then introduce the concept of a finite state automaton, where a system moves from state to state based on events/actions. Ask the students how many states the applet cycles through before returning to a start state. Note that some actions can result in a preservation of state (e.g. clicking on the donkey image immediately after it has been revealed performs no visible action) Present a state table as a framework to facilitate the exploration and mapping of actions to states. Have the students populate their state tables as they explore the applet. When the students have completed their state tables (they can compare their state tables with their neighbors' tables at this time), state diagrams may be introduced. Have the students translate their state tables to an equivalent state diagram.

Now, you can present the students with your formal assignment document. With state tables and diagrams in hand, the students can proceed to apply functional decomposition in conjunction with event-driven programming design to implement their model of this game. In my formal assignment document (Anda, 2015), [which assumes use of the Reed (2011) textbook] I provide links to the images used for each of the doors (I allow students the option to express their creativity by allowing them to link to image files of their choice) and an JavaScript/HTML framework for the doors and the function that is called when a door is selected. I stipulate the set of counters to display the proportion of games won when switching or staying put respectively. I also present a JavaScript statement which facilitates the random selection of a donkey image door when the user happens to have selected the winning door:

```
door_to_open = (winning_door + RandomInt(1, 2)) % 3;
```

This example of the use of the mod operator can be used to initiate a more thorough discussion of its applications and properties.

When the student has (in the student's opinion) completed the programming, the student is to run their program through a sufficiently numerous count of iterations. They are to then use the computed tallies (that they are displaying) to assess whether their results verify the counter-intuitive theoretical probability analysis. If their results continue to significantly deviate from the expected outcome, then they should recognize that a logic error is the cause. If a logic error is indicated, the student will be expected to try to locate and eliminate it (perhaps with assistance from the instructor).

In a standard CS1 course using `cin` and `cout` for I/O, this problem could be implemented using a menu-based textual user interface within a loop.

2.4 Assessment

I have been assigning this problem in our CS0 JavaScript-based class for several years. All students are challenged by this assignment, but because they are implementing an interactive game, I find them motivated to rise to the challenge. Some students benefit from instructor assistance in a lab setting if they are having problems with their design or implementation. Despite the challenge, most students complete this assignment. After successfully completing this assignment, students should have a stronger command of functional decomposition, modeling, and a stronger appreciation for the utility of computational modeling.

References

Anda, Andrew A. (2015) "*Let's Make a Deal* Assignment"
URL: http://web.stcloudstate.edu/aanda/cs200/assignments/MontyHall.html

Cucić, Dragoljub (2009) *Types of Paradox in Physics.*
URL:http://arxiv.org/ftp/arxiv/papers/0912/0912.1864.pdf

Denning, Peter (2003) "Great principles of computing", *Commun. ACM* 46, 11 (November 2003), 15-20. DOI=10.1145/948383.948400
http://doi.acm.org/10.1145/948383.948400

Reed, David (2011) *A Balanced Introduction to Computer Science*, 3rd Ed., Prentice Hall, Boston.

Roschelle, Jeremy (1995) *Learning in Interactive Environments: Prior Knowledge and New Experience* . American Association of Museums.
URL:http://www.exploratorium.edu/ifi/resources/museumeducation/priorknowledge.html
Accessed: 2015-03-22.
(Archived by WebCite at http://www.webcitation.org/6XEXm10td)

West, R. Webster and Street, Scott (1996) *The Let's Make a Deal Applet*.
URL:http://www.stat.sc.edu/~west/javahtml/LetsMakeaDeal.html Accessed 2015-03-20.
(Archived by WebCite at http://www.webcitation.org/6XBWTqAVO)

'Wikipedia', *Monty Hall problem*, (wiki article), March 11, 2015, Available from <http://en.wikipedia.org/wiki/Monty_Hall_problem>.[20 March 2015]