CS 330 Artificial Intelligence and Game Development
Spring 2022

Jay Sebastian, jms0147@uah.edu, adapted from the original assignment by Mikel D. Petty, Ph.D.

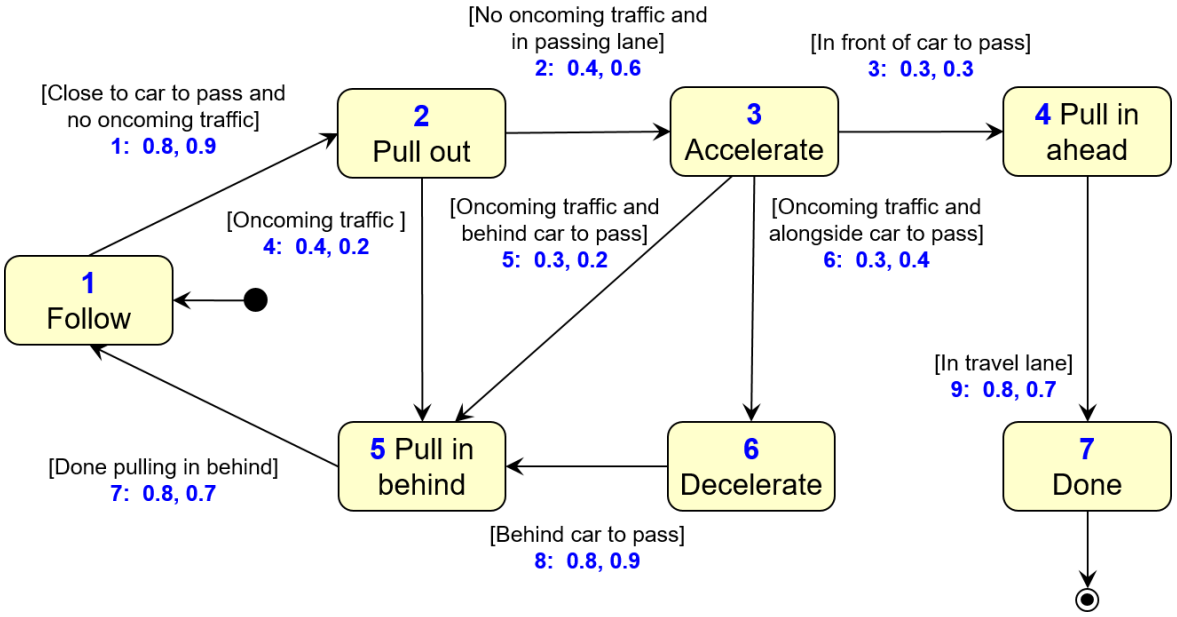
Programming Assignment 4: Hard-coded State Machines

Objective

Implement and test a hard-coded state machine.

Requirements

Implement the Passing maneuver state machine discussed in Lecture 18. The following figure is the state machine diagram from that lecture, with additional information needed for this assignment added to the diagram and shown in blue:



In the figure, the blue numbers in the states' symbols are the state numbers. Under each transition's condition there are three blue numbers. From left to right, they are: the transition number, the transition probability for scenario 1, and the transition probability for scenario 2. The probabilities are different for the two scenarios. The scenarios and transition probabilities are explained later in this document.

There are no characters or game variables in this assignment. You are to implement only the states and transitions shown in the figure above. You will implement only the control structure for the state machine. All of the states' actions and the transitions' conditions will be implemented as stubs. The intent of the program is to show the sequence of states that the state machine passes through from its activation until its deactivation.

You do not need to implement real actions, such as Follow and Pull out, associated with the states. Your implementation should have "stub" functions for the actions that instead of executing an action simply write a line to an output text file stating that that action is executing. As an example of what I mean by a line for each state, the following is an excerpt of the output from my own implementation of this program:

```
iteration= 5
state= 1 Follow
state= 2 Pull out
state= 5 Pull in behind
state= 1 Follow
state= 2 Pull out
state= 5 Pull in behind
state= 1 Follow
state= 2 Pull out
state= 2 Pull out
state= 3 Accelerate
state= 4 Pull in ahead
state= 4 Pull in ahead
state= 7 Done
```

Similarly, you do not need to implement the actual conditions, such as [Close to car to pass and no oncoming traffic], associated with the transitions. Instead, your implementation should have "stub" conditions that simply test a random number for certain values. Most programming languages offer a built-in or library function to get a random number uniformly distributed between 0 and 1. The random number should be compared to probabilities (explained in detail below) for the transitions to determine if a transition's condition is true or false.

Your program should execute two "scenarios" or sequences of iterations, where one iteration is one execution of the state machine from activation in the start state to deactivation in the end state. Scenario 1 should have 100 iterations, with the iteration number and state-by-state sequence shown for each iteration, as in the example above. After the 100th iteration (only), your program should write to the output text file the total number of times each state was entered during the 100 iterations and the relative frequency that each state was entered. The relative frequency for state *X* is (calculated as number of times state *X* was entered) / (total number of times all states were entered). This requirement implies that your program will have to keep track of the number of times each state was entered. Do not reset those counts between iterations, i.e., the statistics should be reported as the totals for all 100 iterations.

The statistics output from my implementation for scenario 1 follows. The transition probabilities, the state counts and frequencies, and the transition counts and frequencies are listed in numerical order by state, e.g., the state count for state 8 is the 8th value in the state counts line. Your output should follow this example closely in format. Include all lines shown below in your output.

```
scenario                = 1
trace                   = TRUE
iterations              = 100
transition probabilities= 0.8 0.4 0.3 0.4 0.3 0.3 0.8 0.8 0.8
state counts            = 810 811 324 136 674 122 100
state frequencies       = 0.272 0.272 0.109 0.046 0.226 0.041 0.034
transition counts       = 649 297 100 352 98 99 549 99 100
transition frequencies  = 0.277 0.127 0.043 0.150 0.042 0.042 0.234 0.042 0.043
```

Reset the counts between scenario 1 and scenario 2. For scenario 2, disable the state-by-state trace and output *only* the final statistics to the output text file. In my own implementation, the state-by-state output is controlled by a Boolean variable in the program. For scenario 2, execute 1,000,000 (one million) iterations of the state machine. (This may take some time; it takes ~1 minute for my R program to execute 1,000,000 iterations.) Write the output of scenario 2 to a different text file from scenario 1.

Because the transitions' conditions are tested using random numbers, it is highly unlikely that your output for the 100-iteration scenario 1 will match my scenario 1 output (shown above) exactly. However, for the 1,000,000-iteration scenario 2, the effect of the random numbers will "even out", and your state counts and frequencies should be quite close to my output.
(My output for scenario 2 is not shown above because it will be used as a part of the grading rubric.)

Using the transition probabilities

The transition probabilities for scenario 1 are shown in both the figure and in the sample output earlier in the document. The transition probabilities for scenario 2 are shown in the figure.

The least obvious part of this assignment is how to use the transition probabilities to determine if the transition's conditions are true or false and which state is transitioned into. Two examples will illustrate the process.

Example 1: State 1 Follow, Scenario 1.
The state has one outgoing transition:
Transition 1, which has a probability of 0.8.
Generate a random number R uniformly distributed between 0 and 1.
If $0.0 \leq R < 0.8$, transition 1 is true, and the state machine transitions to state 2 Pull out.
If $0.8 \leq R < 1.0$, no transition is true, and the state machine does not transition.

Example 2: State 3 Accelerate, Scenario 2.
The state has three outgoing transitions:
Transition 3, which has a probability of 0.3.
Transition 5, which has a probability of 0.2.
Transition 6, which has a probability of 0.4.
Generate a random number R uniformly distributed between 0 and 1.
If $0.0 \leq R < 0.3$, transition 3 is true, and the state machine transitions to state 4 Pull in ahead.
If $0.3 \leq R < 0.5$, transition 5 is true, and the state machine transitions to state 5 Pull in behind.
If $0.5 \leq R < 0.9$, transition 6 is true, and the state machine transitions to state 6 Decelerate.
If $0.9 \leq R < 1.0$, no transition is true, and the state machine does not transition.

Important notes regarding the transition probabilities:

1. The bounds of the range of values for *R* for each outgoing transition from a state are cumulative for that state. In other words, for a state, each outgoing transition's range starts where the last one ends, as shown in Example 2. In example 2, if the range of values for *R* for transition 5 was instead set to $0.0 \leq R < 0.2$, the state machine will execute, but the results will be incorrect. The ranges start over at 0.0 for each state.
2. If no transition is true, the state machine does not freeze in its current state. Rather, there is no transition for that periodic check of the conditions, but there may be one at the next periodic check. Your program should simply repeat checking conditions and transitioning states until the state machine's end state is reached, regardless of how may condition tests are required.
3. When checking the conditions of the outgoing transitions from a state, generate one random number *R* and use it for all of the transitions for that state's check. If your program generates a new value of *R* for each transition, the state machine will execute, but the results will be incorrect. Generate a new value of *R* for each new check of a state's outgoing transitions, including if a state is being repeated because none of its transitions were true on the previous check.

Deliverables

1. Source code file(s) for the program
2. Two output text files, one for each scenario

Grading

This assignment is worth a maximum of ten (10) points. The grading rubric is as follows:

Points	Criterion
0	Assignment not submitted or deliverables missing
1	Assignment deliverables all submitted
0-1	Software engineering quality of the program reasonably good
0-2	Hard-coded state machine control structure present and correct
0-1	State actions stub functions present and correct
0-2	Transition conditions probability tests present and correct
0-1	Text files present and contain all required output
0-2	State counts and frequencies for scenarios 1 and 2 reasonably close to my output
0-10	Total

The grading items are independent of each other (except, of course, that you cannot get any of the other points if you don't turn the assignment deliverables in).

Resources

The following have been posted to the course Canvas page in Files > Programming Assignments > Program 4: State Machines:

1. R code for Program 4.
2. A sample output file for scenario 1. The sample output file will have my actual results for scenario 1. Your output file should resemble its format closely.

Due date

See Assignments page in the course Canvas website.

Assignment-specific recommendations
(See the general instructions for general recommendations.)

1. Remember, the two scenarios have different transition probabilities. Be sure to use the correct probabilities for each scenario. If you run scenario 2 with scenario 1's probabilities or vice versa you will get incorrect results and lose points.
2. This assignment is intended to be the easiest of the four programming assignments. Please don't overengineer it or think I'm expecting a super-flexible all-purpose state machine implementation. Simply implement the one required state machine in a straightforward manner as hard-coded nested if-else if-if statements, as shown in the slides and in my R program.

End of Programming Assignment 4