CSCI 338
Project 1

Assigned 8/30/2022, due by start of class (3:05 pm) on 9/29/2022. Please submit this assignment to the appropriate dropbox on D2L. If D2L is down, email it to me. You must follow the collaboration policy detailed on the course website.

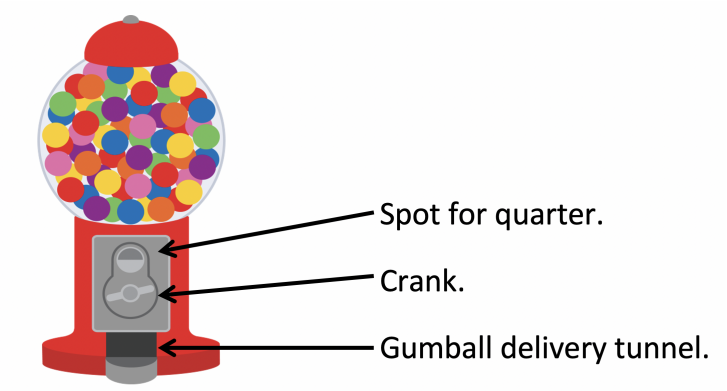
The first part of this project is to gain familiarity with a finite automaton simulation tool called JFLAP. Follow the following instructions to get JFLAP up and running and to gain some familiarity:

1. Go to `https://www.jflap.org/` and download JFLAP.
2. Download and extract the following file:
   `https://www.cs.montana.edu/yaw/teaching/338/assignments/Proj1_PenDFA.zip`
3. Start JFLAP and select File→Open→PenDFA.jff. This is the state diagram for a clicky pen (Click the back, ink cartridge deploys. Click again, it retracts. . .)
4. Test the DFA by selecting Input→Step by State. In the dialog box, try the input: click.click.click.click. and select OK. Make sure you don't forget the periods in the input! Notice which state is shaded. Click "Step" and note what happens to both the state and the input (lower left corner). You can select "Reset" to start the input running again.
5. Go back to the state diagram. The X in the upper right hand corner (below the window exit button) will exit the input testing and bring you back to the state diagram. Right now we have the state diagram for an operational pen. Modify it so that there is a new state called "broken" that is visited from either other state via the "stomp." action. The DFA is now an NFA. Why?
6. Make a new Step by State test and test the input: click.click.click.stomp. Was there any issue with that input? Now test: click.click.click.stomp.click. Was there any issue with that input? Why?

Include the modified state diagram from JFLAP for step 5 and answers to any questions above in your submission.

The second part of this project is aimed at demonstrating the power of state machines

with respect to object oriented programming. Consider a classic, simple, gumball machine. You insert a quarter, turn a crank, and get a gumball:



You are going to build a computer application that operates like a gumball machine. I.e., You will build and implement a DFA for a gumball machine. This will be accomplished in several steps:

1. Assume the machine has an infinite number of gumballs in it. Use JFLAP to make a state diagram consisting of two states: 1 - HasQuarterState, and 2 - NoQuarterState. Put in transitions for the following actions: 1 - insertQuarter, 2 - removeQuarter, and 3 - turnCrank. Feel free to ask me questions if the instructions are too vague.

2. Consider the following code: `https://www.cs.montana.edu/yaw/teaching/338/assignments/Proj1_code_assigned.zip`. Go through it and make sure you understand how it works. The State file is an interface (`https://docs.oracle.com/javase/tutorial/java/concepts/interface.html`). It enforces that any classes implementing the interface (e.g. HasQuarterState) have a method for each method header in the interface. This is advantageous for mimicking a DFA since each state needs to handle ever character of the alphabet (or every action in our example). Each state from your state diagram is its own class in the code and it implements the interface (i.e., each state handles every possible action). There is one conceptual error in the code related to how one state behaves with a specific action. A state is doing something it shouldn't do with one of the actions. Find the error and fix it.

3. Now suppose that the machine starts with a finite number of gumballs. This means that it can be emptied with repeated purchases. Modify your code to include an empty state as well as a refill action. I would recommend keeping track of how many gumballs are in the machine in the GumballMachine class. You will also need to add some code in an appropriate place to make sure you don't dispense a gumball unless there are gumballs to dispense. Modify the Driver to make sure your new code works as expected.

Include your JFLAP state diagram and source code in your submission.

The final part of this project has you building your own video game state simulation: You

are going to represent a player in a video game with a finite state machine (think Mario). Come up with some set of states that Mario can be in (e.g. Jumping, Running, Squatting, Dead,...). Come up with some actions that the player can enact on Mario (e.g., pressing r key puts Mario into the Running state, so long as Mario was not in the Squatting state or Running state. If in Squatting state, nothing happens. If already in Running state, Mario goes to Standing state). Come up with some actions that the game can enact on Mario (e.g., Fireballs being shot at Marion kills him unless he is currently in the Squatting or Jumping state). You have the freedom to come up with your own states and actions. Just make them somewhat reasonable. Make at least 4 states, 3 player actions (e.g., pressing r key), and 1 game actions (e.g. fireball).

In JFLAP, build the state diagram to represent Mario. Make sure that each possible action is handled for each state. I.e., What will happen when Mario is in the Running state and the player tells him to Squat? You are not going to have an accept state, it is just a state diagram detailing how transitions between states are made.

Use the GumballMachine code as a starting point and implement your solution. You minimally need a Driver or some sort (like for the GumballMachine) where I can call player and game actions on Mario and have the results displayed to me. One bonus point for a ground up re-design of Super Mario Bros. Results of actions need to match what your state diagram says they will be. Include your JFLAP state diagram and source code in your submission.