The Daring Duck!

**Introduction:**

You are in a two-week competition.

You may compete individually or in teams of 2 or 3.

You must code your own Turing Machine in Java to begin. Unlike conventional labs, this part is very open-ended. I will provide some guidance on one approach you may use at the end, but you are free to ignore that guidance. As long as your Turing Machine will properly represent the final state (tape and FSM state) of any inputted machine, it is legitimate.

**The Goal:**

Your goal is to RUN a machine that halts before the competition is over. Just imagining the machine is not sufficient! It must run and terminate.

The team or individual with the highest number of 1s on the tape after their machine halts has the best shot of winning.

**Score breakdown:**

You may use any number of states, call this number |S|.

You may use any alphabet, Γ *(Remember that Γ includes Σ and your null symbol automatically!)*

The cardinality of your starting tape input = |In|.

Your program may have any output, but only 1s count towards your score. O = number of 1s.

Your team score = O / (|S|+|Γ|+|In|)

In order to get credit, you must present a diagram of your machine and a computer readout showing the final output.

**And one last rule: *your program must halt before the end of class in exactly two weeks from when this is assigned.***

**Java Guidance for a TM (you are free to 100% ignore this section):**

It’s been a bit since we coded in Java! Recall that Java works using classes. Thus, an OOP approach would be to break your TM down into several classes. Here is one way you could approach it:

* A Tape class, that receives a starting input in its constructor, and can go left, right, read, and write. The Tape class should automatically resize when it discovers that more tape is needed. It should also have some sort of toString() or oneCount() method so that we can all see the output when the machine is done running!
* A State class that holds a name, various Transitions, and knows whether or not it is a terminal state.
* A Transition class that keeps track of a tape symbol as input and the results: what state to move to next, what to write, and whether to move the tape left or right.
* A StateMachine class that holds all of the states, keeps track of what the current state is, and given an input, moves to the next state and tells the tape to move left and right.
* A TuringMachineRunner class that has main, manually assembles your chosen machine, runs the Turing Machine, checks whether each new state is a final state, and does something reasonable with the output when the final state is reached.

**Grading:**

* **A on assignment, plus THREE automatic , free TEST grades of ‘100%’**: Anyone who beats Mr. Isecke’s score. (No, I will not tell you what that score is until the contest is over, because I went in blindly when I had my two weeks, and so should you. Even knowing that the previous best score was “7.5”, or “200”, or “800,000”, or “45 septillion” is a serious advantage for a competitive team.)
* **A on assignment, plus a Rubber Duck Trophy**: the winning team or individual of the year.
* **A**: Fully functional TM, a machine that has been run, terminated, and obtained a Score greater than 100.0.
* **A-**: Fully functional TM, a machine that has been run that did not yet terminate, but would have obtained a score greater than 100.0 had it terminated. (This is the grade for people who underestimate how long it will take to run their machine, and thus do not terminate, or who run into technology problems and can’t get at least to 100.0 at the last minute. Note that it is possible to get a score of 100.0 with a machine that will run in far less than a second.)
* **B+**: Fully functional TM, a machine that has been run, terminated, and obtained a Score greater than 10.0.
* **B**: You must run a machine, terminate, and obtain a Score above 1.0 in order to get a grade of B. **You should do this before you attempt your final machine!**
* **C**: A functional TM, but score is below 1.0
* **D**: No functional TM, but enough code that I see that you mostly understood how to do it, you just got stuck.
* **F:** None of the above.