

## Clinic 2: Marble Computing DSL

**Due:** Tuesday, May 7th, 11:59pm (Submit your code and writeup to Gradescope)

**Bugs:** We make mistakes! If it looks like there might be a mistake in this handout or the starter code, please ask a clarifying question on Ed.

Modern computers use electricity and tiny transistors to perform computations. But there's nothing preventing you from computing with more exotic materials! Figure 1 shows a few computing devices made out of materials like wood, LEGOs, and marbles.

This clinic will focus on *marble computers*, particularly the Turing Tumble model popularized by the Turing Tumble game [4]. You can play with a small-scale, graphical Turing Tumble simulator [6], and we definitely suggest you do so before starting the project! The ultimate goal of this project will be to write a DSL to make writing marble computers, especially very large or complicated ones, much easier than having to drag-and-drop every piece onto the graphical simulator linked earlier.

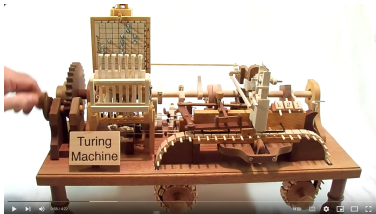
## 1 Classic Turing Tumble Explanation

We first briefly describe the pieces to ‘classic’ Turing Tumble. The computer is programmed by placing pieces on a grid of pegs.

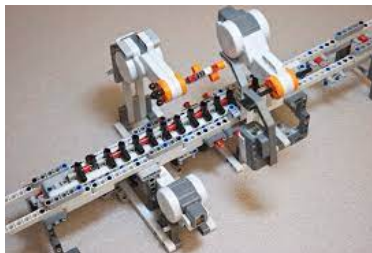
**Marble Droppers.** In classic Turing Tumble, there are two reservoirs of marbles at the top of the board (Figure 2a). One reservoir drops blue marbles, the other drops red marbles. There are also two ‘triggers’ at the bottom of the board; when a marble lands on one of them, a marble is dropped from the corresponding reservoir. This mechanism can be implemented using a mechanical system behind the board.

**Ramps** A *ramp* moves marbles down one row on the board. There are two types of ramps; one moves the marble to the bottom left corner, the other moves the marble to the bottom right corner. Figure 2b shows three ramps organized in a sequence to move a marble down three rows.

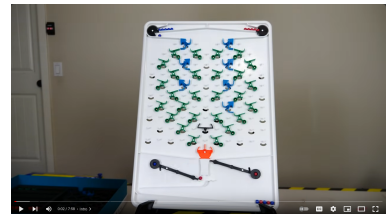
**Bits** A *bit* is like a ramp, except it switches direction every time a marble uses it. Figure 2c shows a bit that will alternate placing the incoming marble on the right or left ramps.



(a) Wooden Turing Machine [3]



(b) LEGO Turing Machine [1]



(c) Turing Tumble [2]

Figure 1: Computing devices made out of wood, LEGOs, and marbles!

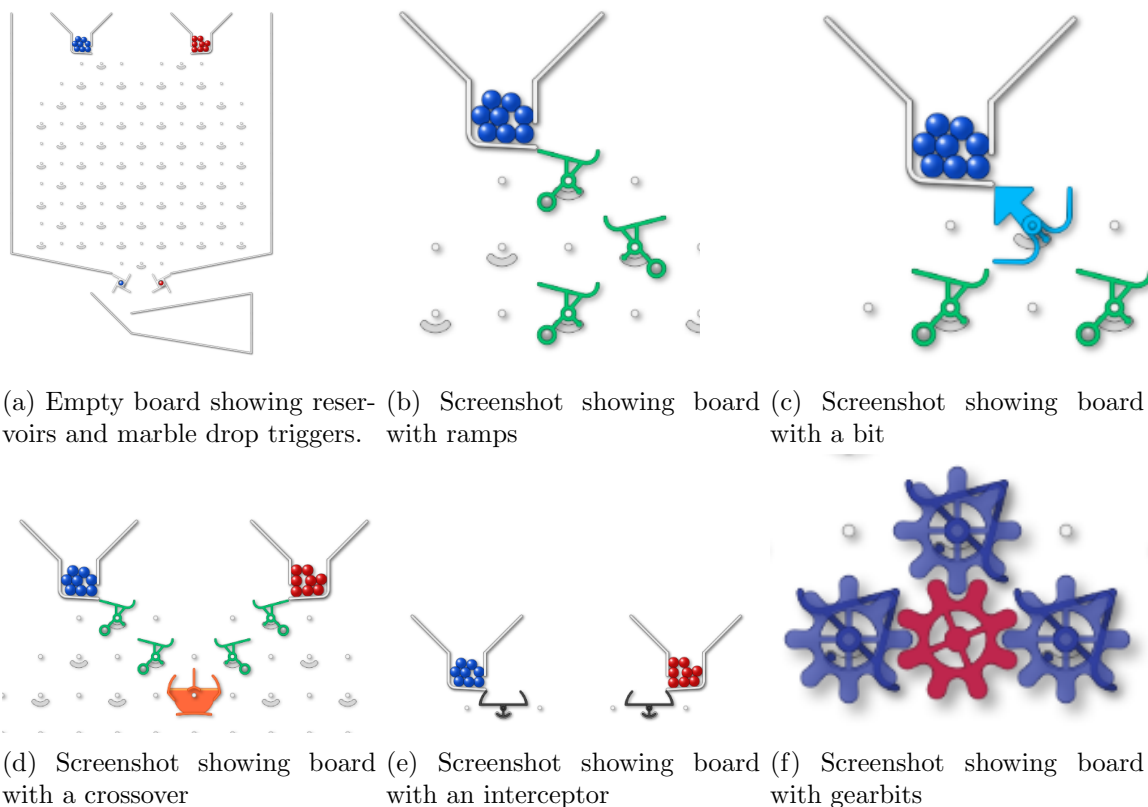


Figure 2: Turing Tumble Pieces

**Crossovers** A *crossover* switches the direction of a marble: marbles coming in from the left are dropped out to the right, and vice-versa. Figure 2d shows a crossover that will release red marbles on its left and blue marbles on its right.

**Interceptors** An *interceptor* stops marbles. Figure 2e shows an interceptor being used to make “no-op” machine.

**Gears and GearBits** A *gear* connects multiple *gearbits* that touch it. Connected gearbits move together: when one switches directions, they all do. Figure 2f shows multiple gearbits connected via a gear. If a marble falls on the leftmost or rightmost gearbit, all three gearbits will change direction. If a marble falls on the top gearbit, it will change directions twice (once for the top gearbit then once for whichever gearbit the marble falls on after that), so the overall action has no effect on the state of the gearbits.

## 2 Python Turing Tumble Simulator

The graphical Turing Tumble simulator is great, but it forces you to drag-and-drop pieces, making it difficult to design very large computers. We’ve provided you with `libmarbles`, a small Turing Tumble simulator written in Python to use instead (though you’re more than welcome to use your own, or hack your DSL into the Javascript simulator linked above!). Our Python simulator makes one major change to the traditional Turing Tumble model: you are allowed to place your

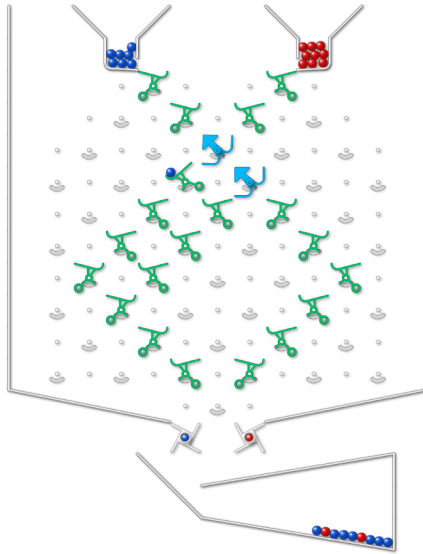


Figure 3: Variant of the blue-blue-blue-red machine shown in the graphical simulator

own “marble drop triggers” anywhere on the board, and they are allowed to drop marbles from anywhere else on the board. This relaxes the restriction that you only have two drop triggers at the bottom of the board and two reservoirs at the top.

Examples for using the library are given with the code. Let’s look at a relatively small one now:

```

1 from libmarbles import *
2
3 ##### alternate blue, blue, blue, red, blue, blue, blue, red, ...
4 board = Board()
5 board.place_cell((0, 0), Bit(RIGHT))
6 board.place_cell((1, -1), DropTrigger(Marble((0, 0), LEFT, BLUE)))
7 board.place_cell((1, +1), Bit(LEFT))
8 board.place_cell((2, 0), DropTrigger(Marble((0, 0), LEFT, BLUE)))
9 board.place_cell((2, +2), DropTrigger(Marble((0, 0), LEFT, RED)))
10
11 board.marble = Marble((0, 0), LEFT, BLUE)
12
13 board.interactive()

```

We first create an instance of the `Board` class, then we place cells (like bits and droptriggers) at different locations on the board. Each location is a pair of numbers, the first being the vertical location on the board (larger numbers mean further down) and the second being the horizontal location on the board (larger numbers mean more to the right).

Bits are given their starting direction, and droptriggers are told where to drop a marble from when the marble reaches the trigger. Finally, the board is given an initial marble location and ‘interactive’ mode is launched, which allows the user to step through execution of the computer.

This particular computer will alternate dropping marbles in the color pattern blue, blue, blue, red, blue, blue, blue, red, .... In fact, you can think of it as a two-bit counter! A similar machine in the Turing Tumble simulator is shown in Figure 3; note it is more complicated because there are only two reservoirs and two drop triggers allowed in the original Turing Tumble.

### 3 Your DSL

Your task is to design a DSL for the marble computing simulator. Your DSL should allow the user to concisely express a marble computer that would be difficult to design manually. You are encouraged to use the `libmarbles` library provided as a “backend” for your DSL, e.g., your DSL could produce an instance of the `Board` class.

We strongly suggest playing with the graphical simulator first before starting on your DSL. We also suggest using the “circuit view” on the graphical simulator, as that is the most similar to our Python interface.

Some ideas to consider:

1. If you’re primarily interested in computations on things like numbers, e.g., making an adding circuit, Turing Tumble encourages the use of Bit cells as inputs. So the user could encode two addends to be added together by encoding them in binary as the initial starting direction for sequences of Bits. You may want to provide explicit support for this, i.e., allow the user to specify ‘inputs’ and automatically convert between Python ints and Bit directions.
2. If you want to think about patterns instead of computations, Turing Tumble can also be used to make super cool patterns! The 3-blue 1-red pattern above is a fun place to start.
3. You might consider a more compiler-oriented design, e.g., allow your user to specify the behavior of the system in terms of a DFA and then compile it down.
4. In our Python simulator, the system will throw an error if a marble ever lands on an empty space. You could consider either adding a check to ensure your machine can never cause such an error, or maybe design your DSL so that it’s not possible to generate such machines.
5. You could look to hardware design languages like Verilog for inspiration.
6. One of the most frustrating things is the need to place individual pegs at specific X/Y locations on the board. It would be nice to be able to use relative locations, like “place this piece below-to-the-right of that piece” and have the system then find absolute locations for them.

We’re also new to Turing Tumble, so we’re super curious to see what everyone comes up with! If you find any cool patterns or marble computations, feel free to post them on Ed. There is also a forum of Turing Tumble enthusiasts [5] where they post designs; you could look there for inspiration or other example boards. In addition, here’s a practice guide from their website with problems and solutions [7].

### 4 Deliverables

Before writing code, please answer the first three questions in `written.md`. These aim to make you develop a plan for your implementation. Answer the **Feedback** questions in `written.md`.

### Submission

After implementing your DSL, please finish answering the feedback questions in `written.md`. Then just zip up your code and written responses and submit them to Gradescope!

## References

- [1] Building a lego turing machine. <https://www.cs.cmu.edu/~soonhok/building-a-lego-turing-machine.html>.
- [2] A computer that runs on marbles. <https://www.youtube.com/watch?v=8B0vLL8ok8I>.
- [3] Mechanical turing machine in wood. <https://www.youtube.com/watch?v=vo8izCKHiF0>.
- [4] Turing tumble — build marble computers. <https://upperstory.com/turingtumble/>.
- [5] Turing tumble forums. <https://community.turingtumble.com/>.
- [6] Turing tumble graphical simulator. <https://jessecrossen.github.io/ttsim>.
- [7] Turing tumble practice guide. <https://upperstory.com/turingtumble/assets/practice-guide-2021.pdf>.