Reverse Mousetrap Machines with Mental Imagery

Mark Karle

Computer Science, Vanderbilt University

# Abstract

Mousetrap is a game where players build a Rube Goldberg-type contraption that traps a mouse. In this case, a “reverse mousetrap” machine is one that saves the mouse. For a project in a Computational Mental Imagery course, color-coded reverse mousetrap machines were hand-drawn with the aim to create an AI that would reason about the machines using mental imagery rather than mathematical formulae. Rules about the simple physics were created and applied to the image to create several frames of the machine in action. Matlab was used to run the simulation.

# Introduction

The reverse mousetrap machines were drawn in a color-coded manner according to the following rules:

· Blue objects are movable.

· Black objects are not movable.

· Yellow circles inside blue objects act as fixed pivot points.

· Green objects are strings; and green circles in blue or black objects are attachment points for strings.

Additionally, rules about the physics of the world are as follows:

· All objects are assumed to have equal density (so there is no such thing as “heavy” versus “light” objects of the same shape and size).

· Gravity acts to produce a constant downward velocity on any unsupported objects (no acceleration). A pendulum will not swing back and forth, but will fall and come to rest pointing downwards.

· Objects cannot bounce. In the above figure, the blue ball (if left alone) will fall and stop on the floor just under the spot where it has fallen from.

· Objects cannot slide. A box sitting on a ramp will just sit there

With these assumptions about the world, some designs were shown to someone not in the class, Carter Portwood. He quickly explained what would happen to the simpler designs, but designs such as in Fig. 1 which had a more complicated route took longer for him to begin to describe. (Portwood 2016). I agreed with him on all accounts.

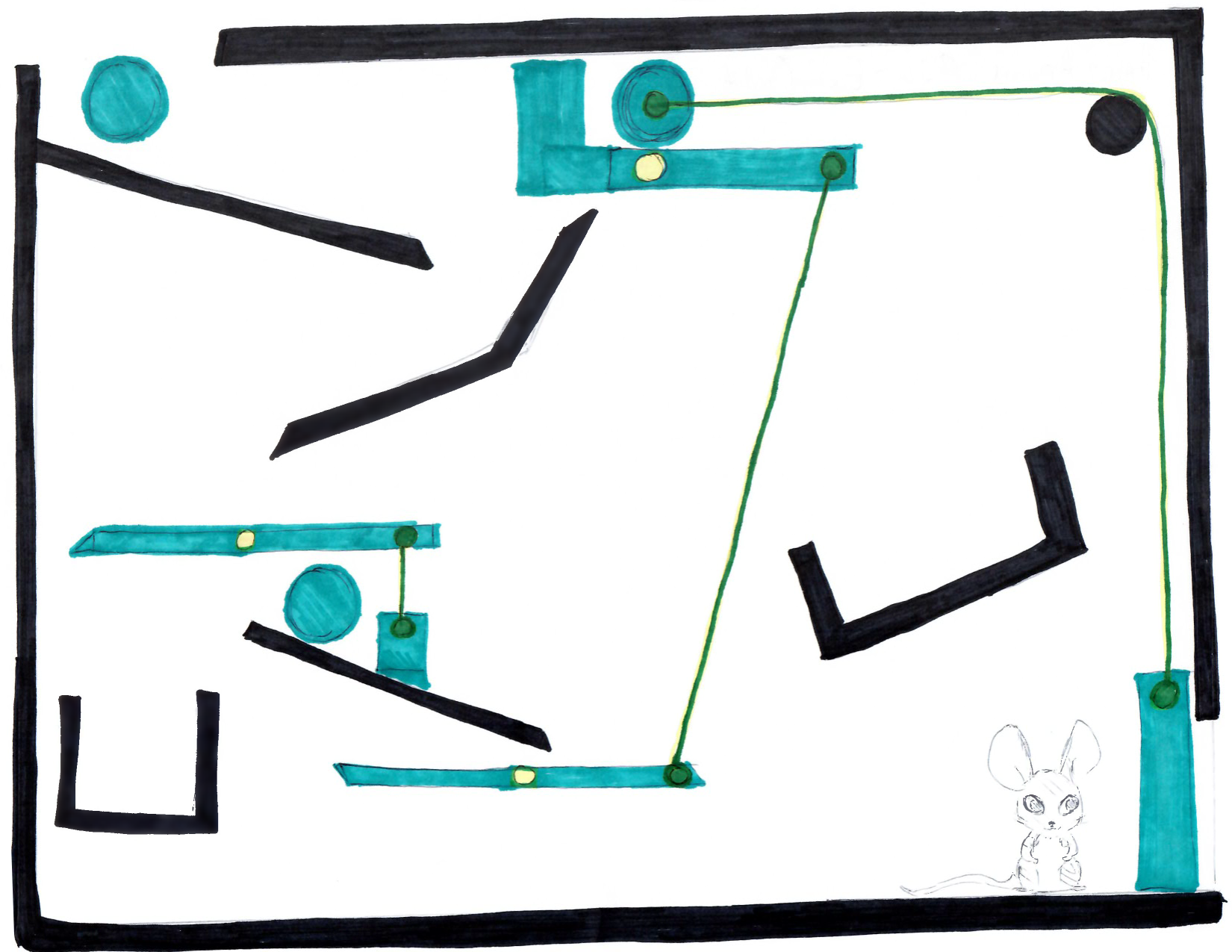


Figure 1. A more complicated design which took Portwood longer to describe

# Methods

The model works by assigning rules based on the type of object. For blue objects, three different types of forces were allowed: rotation due to gravity (for an object with a fixed yellow point), pull from a string (for an object tethered to a string only), and fall due to gravity (for an untethered object). Rotation was done by measuring the areas on either side of the fixed pivot point and rotating down toward the more massive side. Pull from a string happened whenever an attached object moved away from its previous position. The distance moved was calculated and the object being pulled moved at that distance up. All the simulations tested only required the string to pull an object up, so this assumption would break on other designs. Falling happened whenever an object is not touching another object. None of the designs tested required a complicated rule for tipping over on an incline, so this assumption would also break on other designs. The designs were declared successful if the gate keeping the mouse in was lifted and nothing fell on the mouse.

The model first must have the image split into respective colors. Color thresholding was done manually using the Color Thresholding tool in Matlab. This needed to be done for each unique design since the colors varied so much from drawer to drawer. Blue objects were then split into different binary images so that transformations could be done on them independently, and the same was done for green attachment points. Since black and yellow objects are immovable, they were kept in a single respective binary image. Blue object centers of mass were calculated based on the number of pixels associated with the objects. Blue object centers of rotation were the center of mass of a yellow object if it existed inside the blue one, the center of mass of a green attachment point if it existed inside the blue one, or the center of mass of the blue object if it was attached to nothing.

The simulation ran for at most 150 frames. Movement was done using image translation and rotation. Since there were a small number of objects, some ad hoc methods of choosing how to move the images proved to be more useful and time-efficient, but less robust to new designs. First, rotating objects were rotated if there was sufficient mass on one side to rotate the object. If it had an attachment point, the attachment point was also rotated about the same point as the rotated object. Then, the object to which it was attached was marked as “pulled” in an ad hoc manner. If an object was marked as pulled, the distance that the pulling attachment point moved is used to translate the pulled object. The distance was calculated either from the moved attachment point to an intersection point on the string, or directly to the other attachment point. This idea came from (Wang 2016). Then, if an object neither rotates nor is being pulled, it just translates down until it collides with another object. The frames loop until the gate stopping the mouse moves completely out of the way, the mouse collides with an object, or 150 occur.

# Results

The model succeeds in freeing the mouse in three of four tested cases. The design which did not work (mine) was colored so poorly as to provide poor thresholding, so some black objects were falsely detected in a blue area, so the object could not move. Not only that, but a blue lever was too long and would have gotten stuck on a black wall regardless.

The model had to be slightly modified for each new design so that proper object recognition could occur. The required changes were the size threshold for allowing an object to be recognized, the color threshold values, whether strings could handle intersection points, and ad hoc indexes for specific objects. Additionally, the model would fail under circumstances where a ball needed to roll down an incline. However, the model performed well on the designs it was subjected to.

# References

Wang, Xiaotian (2016). Personal Communication.

Portwood, Carter (2016). Personal Communication.