Solving the Rubik's Cube 2.0

New Mexico Supercomputing Challenge
Final Report

April 6th 2016
Team 7
Aspen Elementary School
Los Alamos Middle School

Team Members:

Andy Corliss
Max Corliss
Phillip Ionkov
Ming Lo

Teacher/Sponsor:

Suzy Koehn

Project Mentors:

Latchesar Ionkov
Li-Ta Lo

Executive Summary

Our objective was to create a 3D Rubik's Cube simulation in NetLogo that can scramble and solve itself autonomously or manually by human input. We started this project last year, where we created a Rubik's Cube that can scramble itself and can be solved/scrambled manually. This year, we completed our goal by making the cube able to solve itself autonomously. We thought the project was interesting because it was about Rubik's Cubes, a puzzle which we all enjoy. We also all enjoy programming and think it is a good skill for the future.

Statement of Problem

A Rubik's Cube is a three dimensional cube that has 6 sides that are manipulable and can be scrambled so that it can be solved again. There are approximately 43 quintillion possible combinations on a Rubik's Cube, and if you made a single turn every second on a 3x3x3 Rubik's Cube, it would take around 14 trillion years to go through all the combinations. The most expensive Rubik's Cube is the Masterpiece Cube which is valued at approximately 1.5 million U.S. dollars. You can solve the 3x3x3 Rubik's Cube in at most 20 moves from any position (called God's Number.) The people who proved that are Tomas Rokicki, Herbert Kociemba, Morley Davidson, and John Dethridge [1]. The world record for the 3x3x3 Rubik's Cube is 4.9 seconds. There are many Rubik's Cube competitions where people try to set new records for solving the Cube fastest many ways: blindfolded, one-handed, or with their feet. There are also different kinds of Rubik's cubes such as the 4x4x4, 5x5x5 etc. and special "cubes" like the pyraminx, a pyramid with colors on it that users can manipulate. There are many methods for solving the 3x3x3 (and other) Rubik's Cubes. Computers can solve it by finding the colors and seeing where they have to go to solve the Cube. The main reason people think that the Rubik's Cube is hard or impossible to solve is because to solve the Cube, solvers must use special algorithms to avoid scrambling already completed parts, which many people do not know how to do.

Our goal was to make a program in the NetLogo 3-D interface that could generate a 3x3x3 Rubik's Cube that could be scrambled and solved manually and autonomously. We accomplished all of these tasks.

Description of Method

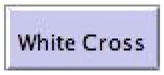
NetLogo 3-D is the programming language that we used. It allows users to create and manipulate 3-D objects. We inputted the algorithms that we use in order to solve the cube [2]. We chose this language because some of us had experience in Netlogo, and Netlogo is a basic coding language which is fairly easy to learn and is suitable for first-time coders. Our Rubik's Cube interface is a collection of 54 "turtles" (the agents in Netlogo and Netlogo 3-D) facing different directions to represent a real-life Cube. We manipulate this cube by changing the colors of the turtles. We achieved this by having each turtle ask its corresponding turtle (relative to which turn is being executed) which color it is, and then have the turtle change to that color. This process repeats for all the affected turtles, and represents a turn on the cube, which can be replicated on a real-life Rubik's Cube.

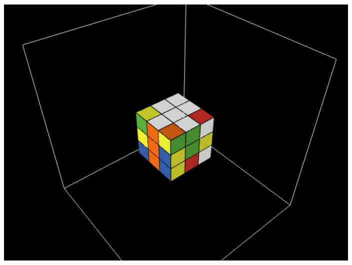
Validating the Model

We made sure that the things we did were right by testing the Cube. At first, when we scrambled the Cube, there would be impossible situations. For example, sometimes the corners would have white on two or even all three sides. Our mentors helped us fix this problem. The other turn buttons worked fine. Additionally, some of our algorithms were flawed in the beginning because they had incorrect turns. We fixed this issue by turning our digital and real-life cubes to simulate the algorithms that worked and corrected the mistakes. Sometimes when we were finding the position of a side piece (like in White Cross and Middle Layer), there would be two places where a piece could be. This is because our sensing only used two “axes”. We fixed this problem by making turns to get it into a place with only one position, and then doing the function again. Also, during the year we noticed that our color scheme did not correspond with the classic Rubik’s Cube scheme: our sides, which should have been in Blue-Orange-Green-Red order to conform with a standard Rubik’s Cube, were switched around.

Results

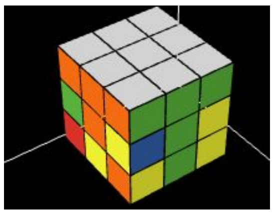
We had exceptional results. We improved last year’s project, where you could turn and scramble the Cube both manually and autonomously. We used the NetLogo 3-D interface to control how the Rubik’s cube turned and scrambled. This year, we created several buttons for different steps for solving the Rubik’s Cube. We also created sensing algorithms and used them so we could tell where pieces are located and tell the Cube what to do. We programmed the Rubik’s Cube so it could solve the White Cross, White Corners, Middle Layer, Yellow Cross, Yellow Corners, and orient the top layer by itself, which, when put together, solve the cube (see below). These are the algorithms and order suggested by [2]. We wrote 2165 lines of code. For our first two layers, our sensing algorithms work by finding a piece’s distance from 2 or 3 centers. For our last layer, we sensed whether a certain piece was in a certain position. To turn the Rubik’s Cube, we find the neighboring turtles’ color and shift the color to the correct position. To turn the Front side, we asked the bordering sides (Up, Down, Left, Right) turtles’ colors, and moved the colors to their corresponding turtles (which varied based on which turn we were making and which direction).

White Cross 

[2] 

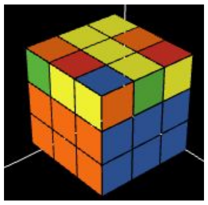
The first step we used to solve the Rubik’s Cube was the White Cross. The White Cross is a position where you get all of the white side pieces (or any chosen color pieces) and you align them next to their colors’ center, making a cross. This is the most challenging step because there are no algorithms that correspond to it. For humans, this step is easier because we only have to find the piece and put it in place. However, it has many possibilities (24) of where the necessary piece can be, which makes the step even harder for computers to solve. We created the White Cross step by first creating sensing algorithms. We decided to find a piece using the centers of its colors and the piece’s distance from the centers. Using this method, we created a sensing algorithm which looks for a piece’s distance from the centers and executes an algorithm to move it to its proper place. This was very difficult because sometimes this algorithm could not distinguish between two spots where a piece could be. Another difficulty was that we had to use the “brute force” method, in which our program listed possible places a piece could be and we created a set of turns for each place that would solve it. This step was very time-consuming and we spent much of the year working on it.

White Corners

[2] 

The next step we had to do was the White Corners. The White Corners is similar to the White Cross, except instead of side pieces, the solver positions the corner pieces. When you finish this step, you have completed the white side. This step is also difficult because it only has basic algorithms and lots of moving the pieces into positions necessary to use the algorithms. There are also 24 possible positions for a piece to be, making this step as hard as the White Cross (in number sense). We started making this the same way we made the White Cross, but we faced another problem. The side pieces have only two colors, so they only needed two centers to be sensed, but the corners have three colors, so we needed to use three centers to find the correct piece. This was a challenge because we didn't know exactly how to change the sensing code so drastically. We did manage to get it working and the rest of the step was considerably easier. Like the White Cross, we used the "brute force" method on the White Corners as well. This was still difficult because the correct piece could still be in many places, and it was harder to find a piece with three sides. Also, there were still places our sensing was not able distinguish. However, this step was less time-consuming than the White Cross because we had more experience with the code.

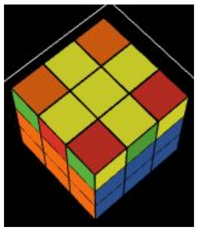
Middle Layer

[2]


The next step was the Middle Layer. To solve the Middle Layer, you move all the edge pieces that have no white or yellow into their corresponding places. This step is much easier than the White Cross and White Corners because there are more advanced algorithms and not as much sensing. Also, the possibilities of where the necessary piece can be, 16, is less than the White Cross and White Corners steps. We started to solve the Middle Layer by changing the top and front centers so that you could find the correct piece. (Instead of white, blue, orange, green, or red, it was two of the latter four). There were fewer places for the correct piece to be than both the White Cross and the White Corners. However, this was still very challenging. Since we had to change the centers, we had to think much differently than what we were used to. For example, in some cases we would not be certain which side was the front, leading to small mistakes in our algorithms.

Yellow Cross

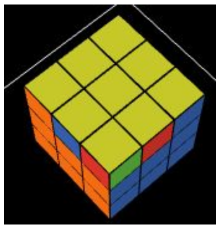
Yellow Cross

[2]


Our next step was the Yellow Cross. In this step, you have two-thirds of the cube solved and you want to create a Yellow Cross so that the following steps are easier. This step was one of the easiest to solve. It only has eight possible positions and there is an easy and quick algorithm. But there was one problem: we had to teach our code to able to tell if a piece was correct or not using only one center. This was very difficult, but with our mentors' help, we managed to overcome this challenge. After that, we added to our algorithm a loop in which the program repeats and senses if all of the pieces are correct, in which case it stops.

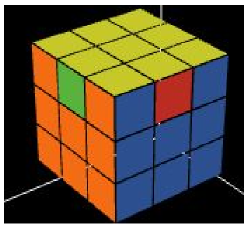
Yellow Corners

Yellow Corners

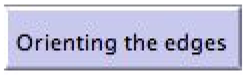
[2]


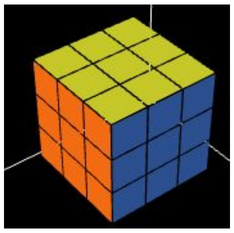
The next step to solve was the Yellow Corners. In this step, you use a certain algorithm to get all of the Yellow Corner pieces facing up, completing the yellow side. This step used the basics of the Yellow Cross, but was much more complicated. The challenge in this step was that the Yellow Cross method found edge pieces, while the Yellow Corners needed to find corner pieces. We tried converting the sensing algorithm over, but we had some bugs. We thought about using the White Corners algorithm, but then we noticed what the bug was and fixed it. We started coding the algorithms and it was working until we found another bug. We searched all through the algorithm that was incorrect, but we couldn’t find anything wrong. Then we saw that there were two algorithms for the same case, which messed it up. We fixed this bug and the Yellow Corners then worked.

Last Layer

[2] 

The next step was the Last Layer. In this step, you move the corner pieces of the top layer so that only the yellow edge pieces are not completely solved. We had found that if you kept iterating a certain corner algorithm, the algorithm would eventually converge to solved corners. The only problem was checking if the algorithm reached the solved state or not. Unlike other sensing algorithms, we had to sense the position of two corners instead of one, such as the two blue corners in the picture above. The built-in Netlogo variable “color” can only be used for one corner at a time. However, we eventually solved the problem by using “color of.” Once it was solved, we had to rotate the top layer such that the color of the corners matches the color at the center piece. We did this by sensing the color of one corner (since it was “solved” already) and seeing where it was compared to its center.

Orienting Edge Pieces 

[2] 

This was the final step to solving the Rubik's Cube. This step is called Orienting the Edge Pieces, and you put the edge pieces in their correct place, thus completely solving the Rubik's Cube. To do so, we had to do three steps. First, we had to find whether it was solved (skip to step three), had one piece solved, or had no pieces solved. Second, we would use an edge algorithm depending on the outcome of step one. If it had one piece solved, we would move that piece so it was on the backside. Then we would repeat the same algorithm until it was ready for positioning. If it had had no pieces solved, we would do the same algorithm to make it have one piece solved, and since the code looped, it would eventually solve itself. Third, we would see if it was in the correct position. By doing so we used the same method of the Last Layer positioning: find a yellow edge piece and match it to its color's center.

Conclusion

Doing this project was very challenging, but also enjoyable. We created a Rubik's Cube that can solve itself completely using sensing algorithms that we created. It can solve the White Cross, White Corners, Middle Layer, Yellow Cross, Yellow Corners, Last Layer, and Orienting Edge Pieces steps. We have finished the Rubik's Cube, and now it is able to solve itself. We are very proud of our program. We encountered many challenges programming the algorithms that we learned into this Rubik's Cube, but persevered and solved the Rubik's Cube.

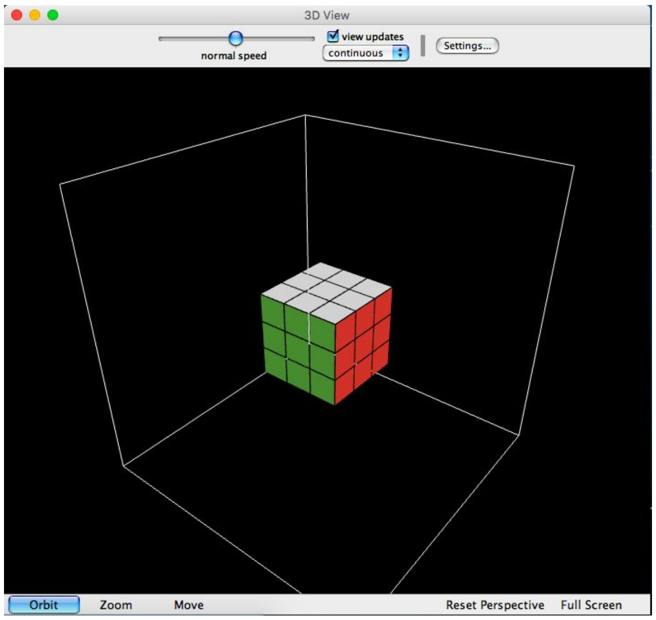
Significant Achievement

We used the Rubik's Cube we created in Netlogo 3-D last year, and this year we programmed in all the steps to solve the Rubik's Cube and made it so that the Rubik's Cube can solve itself.

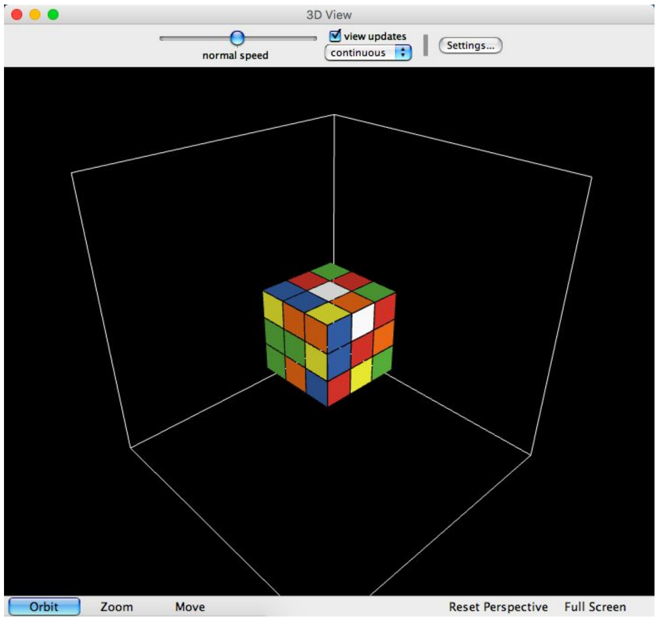
Acknowledgements

We would like to acknowledge our mentors, Latchesar Ionkov and Li-Ta Lo, for helping us when we had trouble with our project, and also giving us ideas and go-arounds when we were stuck. We would also like to thank our sponsor, Susan Koehn, for teaching us about Rubik's Cubes and getting us interested in them.

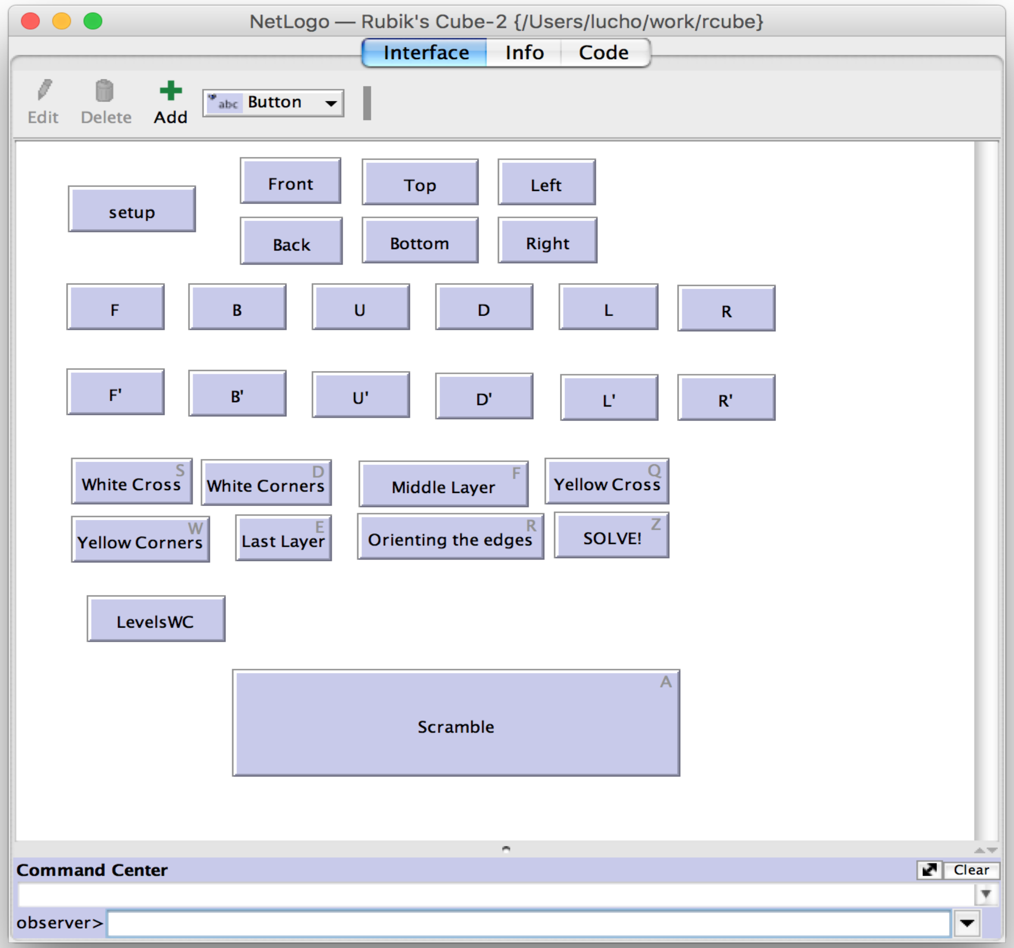
Screenshots



The Rubik's Cube when set up.



The Rubik’s Cube when scrambled.



The user interface to control the execution of the program.

References

[1] http://www.cube20.org

[2] Youcandothecube.com (Also, all algorithms are based off algorithms found here)

[3] Netlogo Dictionary