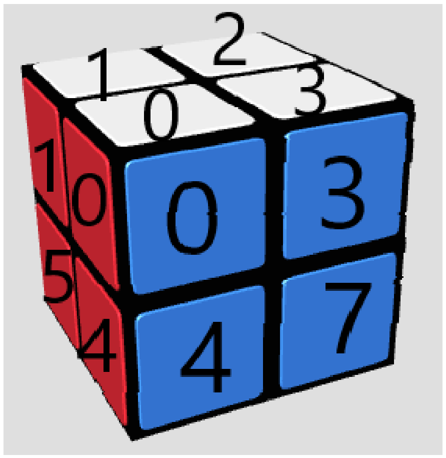
Andrew Davison
CSC 372: Survey of Artificial Intelligence
Dr. Thomas Allen
2/18/2023

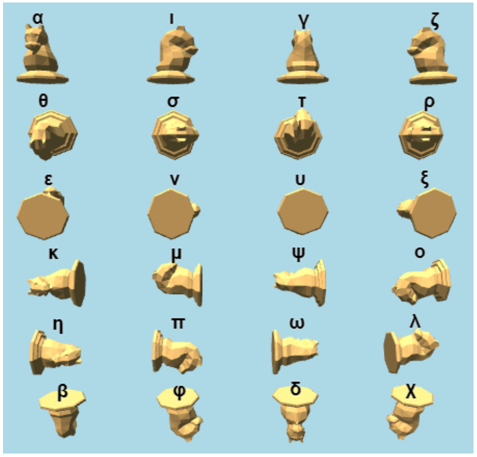
A1: Modeling the 2x2x2 Rubik's Cube

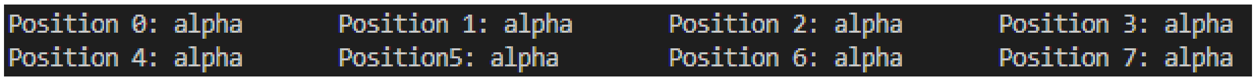
Design and Internal Data Structure

In order to represent a 2x2x2 Rubik's Cube, I use a simple one-dimensional vector of eight elements, each element representing one position or corner of the cube. Each of these elements holds a value from 0 to 23 inclusive, enumerated by Greek letters, to represent the orientation of the cubelet in that position. To better explain how this data structure can uniquely identify a cube, let us consider a visual example alongside how the code represents it.

First, let us define a few of the important concepts. Below is a model of a cube with each position labeled with a number from 0 to 7. These positions are distinct from the cubelets; that is, as the cube turns, the positions do not change. In fact, the starting positions are entirely arbitrary and not tied to specific colors, but for this example, we will assign the positions as thus. To the right of that is a set of 24 different orthogonal orientations of a knight piece in chess. This illustrates the orientations that each cubelet, or subsection of the cube, can take. A default cube begins with its cubelets in the α orientation in the representation.

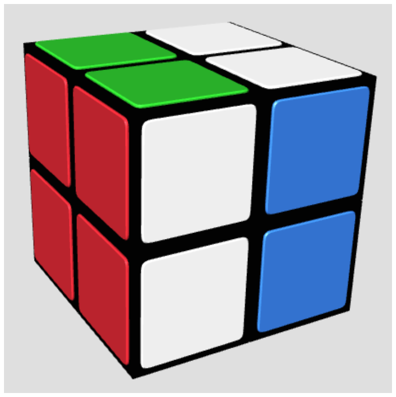


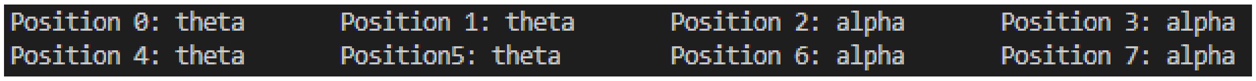




```
Position 0: alpha       Position 1: alpha       Position 2: alpha       Position 3: alpha
Position 4: alpha       Position5: alpha        Position 6: alpha       Position 7: alpha
```

Consider, then, how this internal representation and the cube would correspond as we make a turn. Below is an example of a left clockwise turn in a cube. Notice that the cubelets on the right side of the cube (those in positions 2, 3, 6, and 7) did not move, and that all of the cubelets on the left side of the cube (those in positions 0, 1, 4, and 5) have moved and thus changed orientation. Comparing this to the knights diagram, in relation to the starting orientation, each of these cubelets are now in the θ orientation. This is thus reflected in the internal data structure.

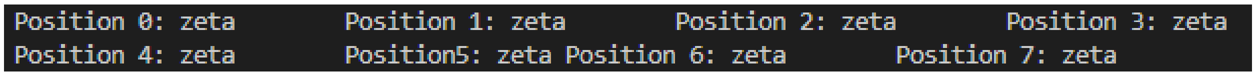




```
Position 0: theta        Position 1: theta        Position 2: alpha        Position 3: alpha
Position 4: theta        Position5: theta         Position 6: alpha        Position 7: alpha
```

This raises the question of how each individual cubelet is then identified. Interestingly, the combination of position and orientation is enough to uniquely identify which cubelet, from the original starting state, is in that position. Even without an extensive proof, it is not difficult to convince yourself of this. Consider the starting state again, where all cubelets are in the α orientation. Further, consider Position 0. Is it possible to maintain the orientations of any of the other cubelets on the cube and transpose it into Position 0? No, doing so would require disassembling the cube. In fact, each position has three different orientations per cubelet that identifies which one is in the position – think of it in the case that each cubelet can fit into each position with the colors in three different arrangements, and with eight cubelets, this spans the 24 different possible orientations.

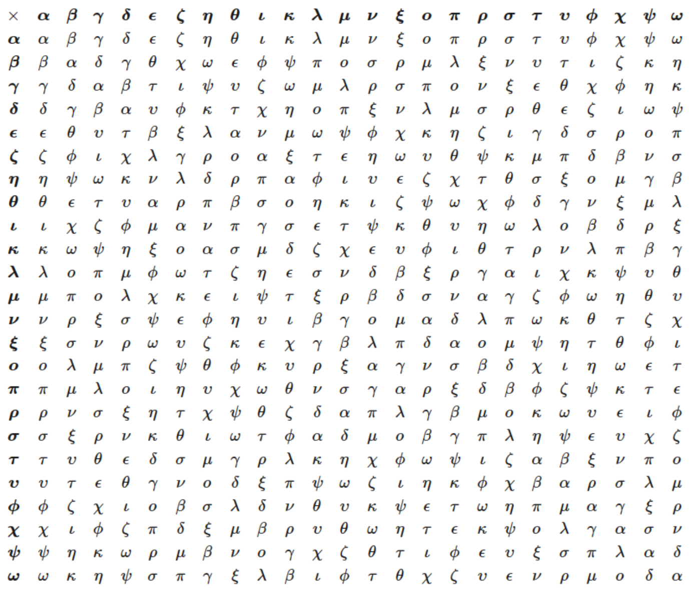
Thus, we know that this representation can uniquely identify each different possible permutation of the cube. This representation, however, allows for many redundant states. Consider the case where were rotate the entire cube to the left. The data looks like this:



```
Position 0: zeta         Position 1: zeta         Position 2: zeta         Position 3: zeta
Position 4: zeta         Position5: zeta Position 6: zeta         Position 7: zeta
```

Which is clearly a different state in the representation, but in the real world, the exact same state. This is a shortcoming of this representation, but an important observation can be made that in the case where all positions have cubelets aligned into the same orientation, the cube is then in a solved state. With such a small representation of a cube and quick state evaluation, I find that this representation will likely still be useful and effective in A2 despite the presence of redundant states.

Finally, it is worth going over the mechanics of making rotations in a bit more detail. Each of the 12 different rotations is handled by a function that changes the values of the affected elements in a rotation. The vital part of this is the multiplication table between orientations. The term “multiplication” can be misleading – really, this table shows how orientations map onto each other. The orientations in the left column are the starting orientations, and the orientations across the top are the rotations being affected upon these orientations. The orientation produced in that operation is found in the table.

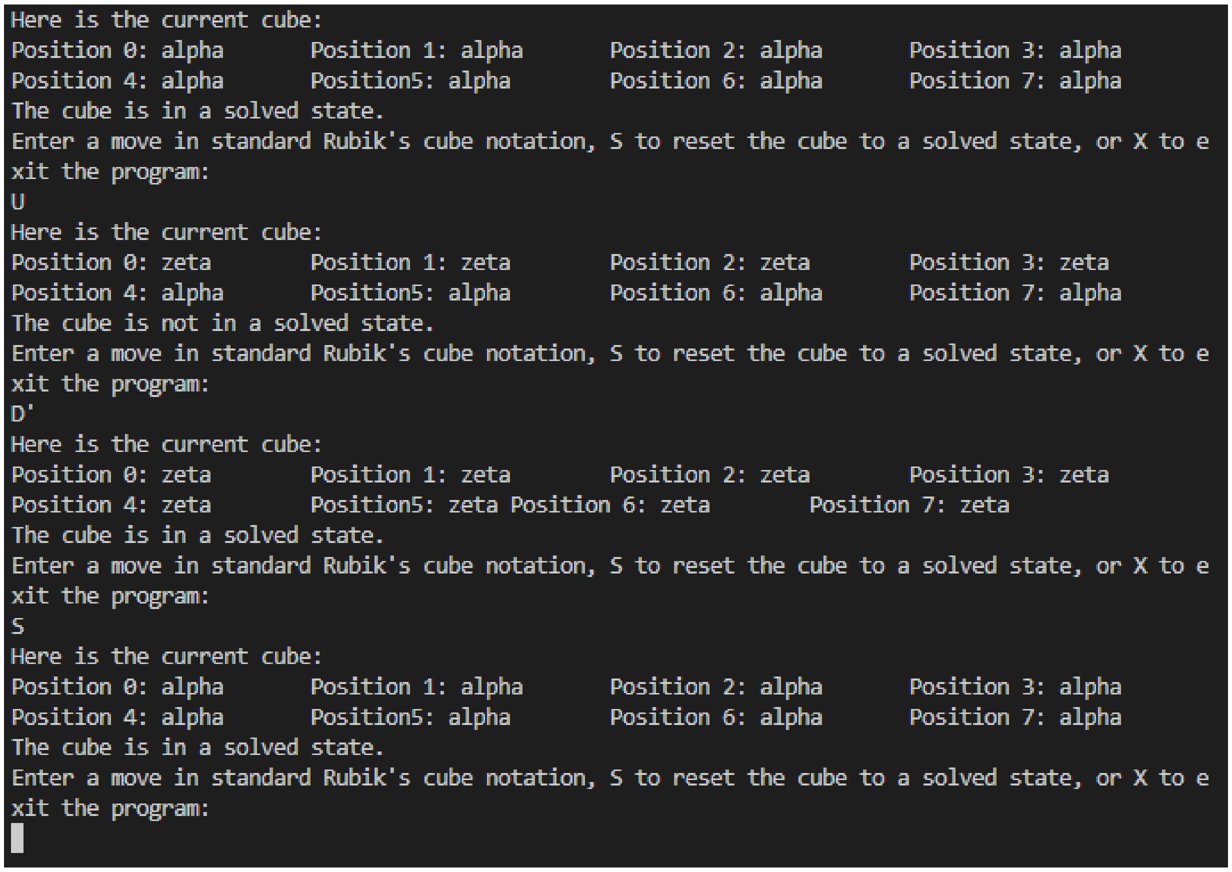


×	α	β	γ	δ	ϵ	ζ	η	θ	ι	κ	λ	μ	ν	ξ	ο	π	ρ	σ	τ	υ	ϕ	χ	ψ	ω
α	α	β	γ	δ	ϵ	ζ	η	θ	ι	κ	λ	μ	ν	ξ	ο	π	ρ	σ	τ	υ	ϕ	χ	ψ	ω
β	β	α	δ	γ	θ	χ	ω	ϵ	ϕ	ψ	π	ο	σ	ρ	μ	λ	ξ	ν	υ	τ	ι	ζ	κ	η
γ	γ	δ	α	β	τ	ι	ψ	υ	ζ	ω	μ	λ	ρ	σ	π	ο	ν	ξ	ϵ	θ	χ	ϕ	η	κ
δ	δ	γ	β	α	υ	ϕ	κ	τ	χ	η	ο	π	ξ	ν	λ	μ	σ	ρ	θ	ϵ	ζ	ι	ω	ψ
ϵ	ϵ	θ	υ	τ	β	ξ	λ	α	ν	μ	ω	ψ	ϕ	χ	κ	η	ζ	ι	γ	δ	σ	ρ	ο	π
ζ	ζ	ϕ	ι	χ	λ	γ	ρ	ο	α	ξ	τ	ϵ	η	ω	υ	θ	ψ	κ	μ	π	δ	β	ν	σ
η	η	ψ	ω	κ	ν	λ	δ	ρ	π	α	ϕ	ι	υ	ϵ	ζ	χ	τ	θ	σ	ξ	ο	μ	γ	β
θ	θ	ϵ	τ	υ	α	ρ	π	β	σ	ο	η	κ	ι	ζ	ψ	ω	χ	ϕ	δ	γ	ν	ξ	μ	λ
ι	ι	χ	ζ	ϕ	μ	α	ν	π	γ	σ	ϵ	τ	ψ	κ	θ	υ	η	ω	λ	ο	β	δ	ρ	ξ
κ	κ	ω	ψ	η	ξ	ο	α	σ	μ	δ	ζ	χ	ϵ	υ	ϕ	ι	θ	τ	ρ	ν	λ	π	β	γ
λ	λ	ο	π	μ	ϕ	ω	τ	ζ	η	ϵ	σ	ν	δ	β	ξ	ρ	γ	α	ι	χ	κ	ψ	υ	θ
μ	μ	π	ο	λ	χ	κ	ϵ	ι	ψ	τ	ξ	ρ	β	δ	σ	ν	α	γ	ζ	ϕ	ω	η	θ	υ
ν	ν	ρ	ξ	σ	ψ	ϵ	ϕ	η	υ	ι	β	γ	ο	μ	α	δ	λ	π	ω	κ	θ	τ	ζ	χ
ξ	ξ	σ	ν	ρ	ω	υ	ζ	κ	ϵ	χ	γ	β	λ	π	δ	α	ο	μ	ψ	η	τ	θ	ϕ	ι
ο	ο	λ	μ	π	ζ	ψ	θ	ϕ	κ	υ	ρ	ξ	α	γ	ν	σ	β	δ	χ	ι	η	ω	ϵ	τ
π	π	μ	λ	ο	ι	η	υ	χ	ω	θ	ν	σ	γ	α	ρ	ξ	δ	β	ϕ	ζ	ψ	κ	τ	ϵ
ρ	ρ	ν	σ	ξ	η	τ	χ	ψ	θ	ζ	δ	α	π	λ	γ	β	μ	ο	κ	ω	υ	ϵ	ι	ϕ
σ	σ	ξ	ρ	ν	κ	θ	ι	ω	τ	ϕ	α	δ	μ	ο	β	γ	π	λ	η	ψ	ϵ	υ	χ	ζ
τ	τ	υ	θ	ϵ	δ	σ	μ	γ	ρ	λ	κ	η	χ	ϕ	ω	ψ	ι	ζ	α	β	ξ	ν	π	ο
υ	υ	τ	ϵ	θ	γ	ν	ο	δ	ξ	π	ψ	ω	ζ	ι	η	κ	ϕ	χ	β	α	ρ	σ	λ	μ
ϕ	ϕ	ζ	χ	ι	ο	β	σ	λ	δ	ν	θ	υ	κ	ψ	ϵ	τ	ω	η	π	μ	α	γ	ξ	ρ
χ	χ	ι	ϕ	ζ	π	δ	ξ	μ	β	ρ	υ	θ	ω	η	τ	ϵ	κ	ψ	ο	λ	γ	α	σ	ν
ψ	ψ	η	κ	ω	ρ	μ	β	ν	ο	γ	χ	ζ	θ	τ	ι	ϕ	ϵ	υ	ξ	σ	π	λ	α	δ
ω	ω	κ	η	ψ	σ	π	γ	ξ	λ	β	ι	ϕ	τ	θ	χ	ζ	υ	ϵ	ν	ρ	μ	ο	δ	α

This is implemented by enumerating these orientations and entering the multiplication table above into a two-dimensional array. Then, finding the result of a rotation operation is as trivial as simply accessing an element in that array. It’s worth noting that the 12 different moves only implement 6 different rotations – the ζ, ι, η, κ, θ, and ε rotations – but that each of the 24 different orientations is reachable for every cubelet.

Interface

Currently, the command line interface is very minimal. Upon running it, the program displays the structure as it is displayed above and asks for a rotation, to reset the cube to a solved state, or to exit out of the program. While I believe this works well for those who are quite familiar with the internal data structure, it is lacking in its visual display of what the cube would look like in some sort of color-aware or 3-dimensional space. Here is a series of commands to demonstrate the interface:



```
Here is the current cube:
Position 0: alpha       Position 1: alpha       Position 2: alpha       Position 3: alpha
Position 4: alpha       Position5: alpha        Position 6: alpha       Position 7: alpha
The cube is in a solved state.
Enter a move in standard Rubik's cube notation, S to reset the cube to a solved state, or X to e
xit the program:
U
Here is the current cube:
Position 0: zeta        Position 1: zeta        Position 2: zeta        Position 3: zeta
Position 4: alpha       Position5: alpha        Position 6: alpha       Position 7: alpha
The cube is not in a solved state.
Enter a move in standard Rubik's cube notation, S to reset the cube to a solved state, or X to e
xit the program:
D'
Here is the current cube:
Position 0: zeta        Position 1: zeta        Position 2: zeta        Position 3: zeta
Position 4: zeta        Position5: zeta Position 6: zeta        Position 7: zeta
The cube is in a solved state.
Enter a move in standard Rubik's cube notation, S to reset the cube to a solved state, or X to e
xit the program:
S
Here is the current cube:
Position 0: alpha       Position 1: alpha       Position 2: alpha       Position 3: alpha
Position 4: alpha       Position5: alpha        Position 6: alpha       Position 7: alpha
The cube is in a solved state.
Enter a move in standard Rubik's cube notation, S to reset the cube to a solved state, or X to e
xit the program:
```

Proposed Heuristic

From my observations of this representation, my proposed heuristic is to consider the number of different orientations in the cube. My general observations are that a solved cube has 1 total orientation, a cube one turn from being solved has two total different orientations (as seen above), and so on. Using the multiplication table, it is possible to find the distance from a given orientation to another orientation, but with 24 different goal states, it's impractical to calculate the heuristic for each possible solved state. Instead, I propose that the heuristic should consider the number of different orientations in the data structure, minus one. Subtracting one is similar to selecting one of those orientations arbitrarily as the orientation for the goal solved state, and counting the number of orientations "out of place" – just like a Hamming distance.

This heuristic should never overestimate the number of turns needed to get to a solved state. Instead, I suspect that this will actually regularly undercount the number of turns needed, but will still be an admissible heuristic.

Lessons Learned from this Assignment

I've learned a lot about 3-dimensional rotations and Euclidean angles in my research for this project, but most importantly, I've learned that in higher level problems like these, spending lots of time on the planning stage is hugely important and can legitimately outweigh the implementation time. I've also learned that I need to be a bit more proactive with these assignments, and I'm very excited to get an early start on assignment A2.

Acknowledgements and References

I found a significant part of research on this blog, which strongly inspired my data structure for this problem and provided the knights graphic and orientation multiplication table:
https://k-l-lambda.github.io/2020/12/14/rubik-cube-notation/

I retrieved my Rubik's cube models from here:
https://www.grubiks.com/puzzles/rubiks-mini-cube-2x2x2/

Additionally, I discussed my ideas for this structure with classmate Michael Liao. We did not go into any specifics, and we talked sparingly, but I'd like to acknowledge his help in clarifying my thought processes.