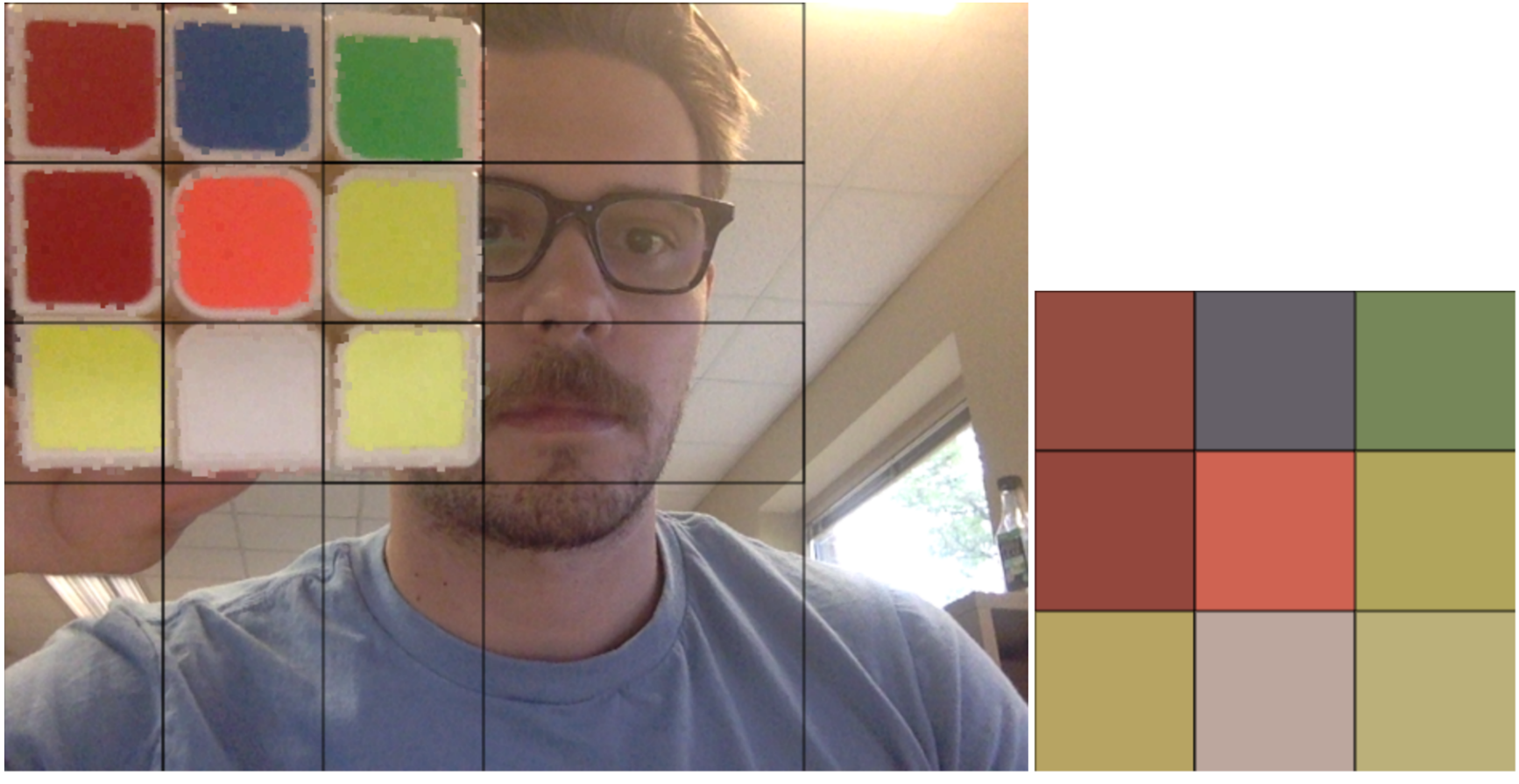
Rubik's Cube Solver

Stuart Downing
Tyler Finateri
Clayton Zimmerman
Austin Dumm
Derek Nelson

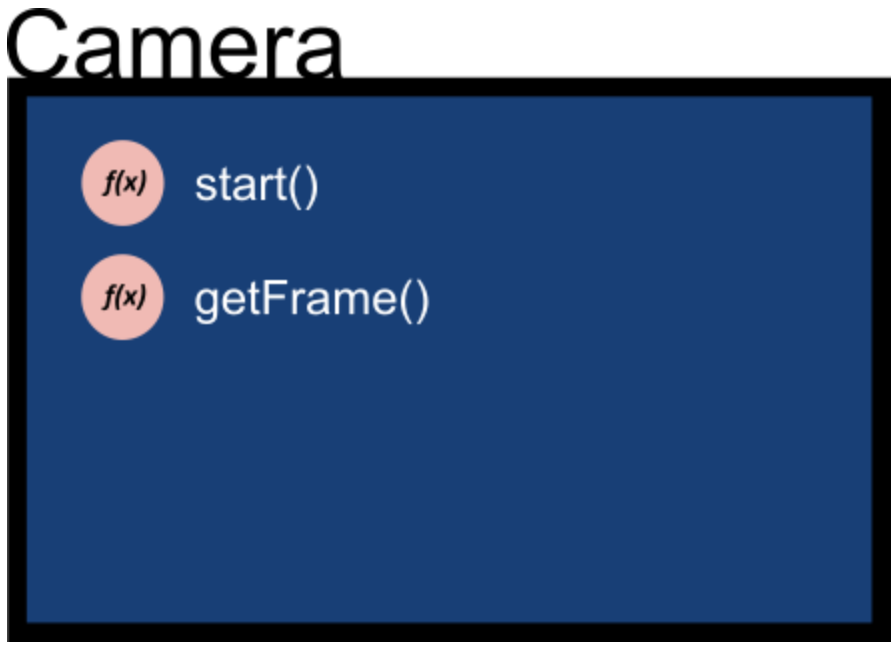
The idea

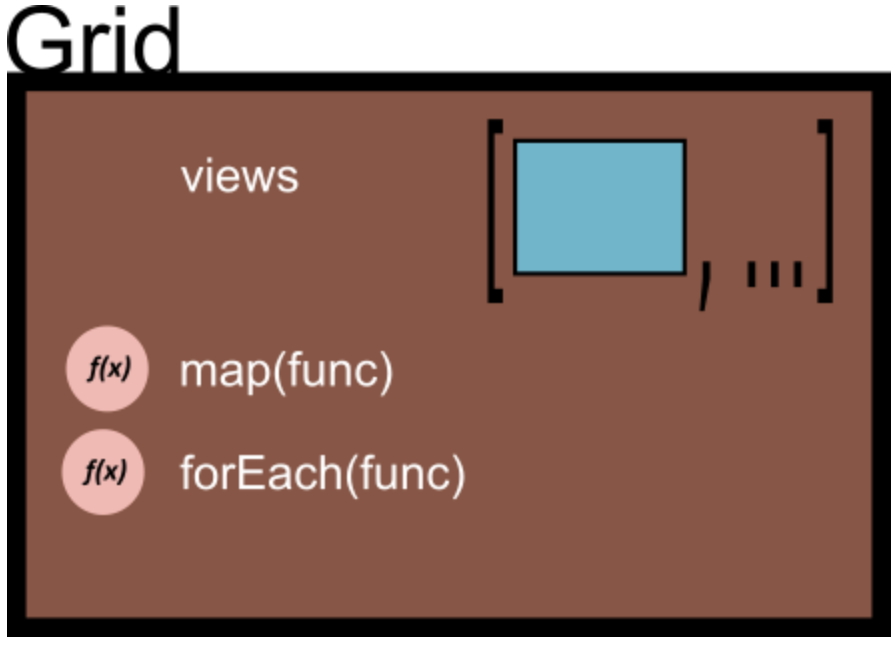
We wanted to solve a Rubiks cube in Javascript getting the initial state from a webcam.

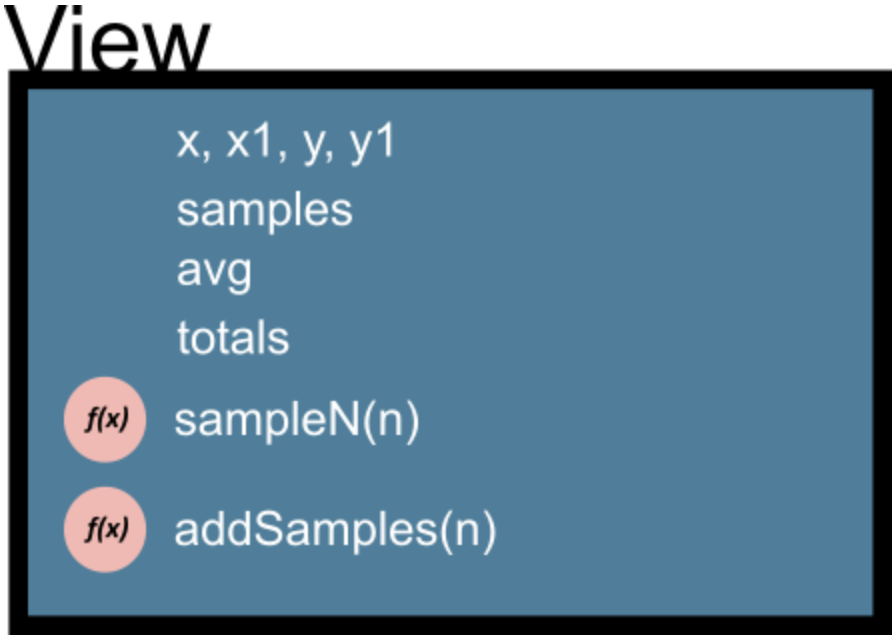


Tech

- Javascript (ES6)
- Node & NPM
- Webpack 2
- Chrome
- ThreeJS


Camera
f(x)
start()
f(x)
getFrame()


Grid
views
f(x)
map(func)
f(x)
forEach(func)

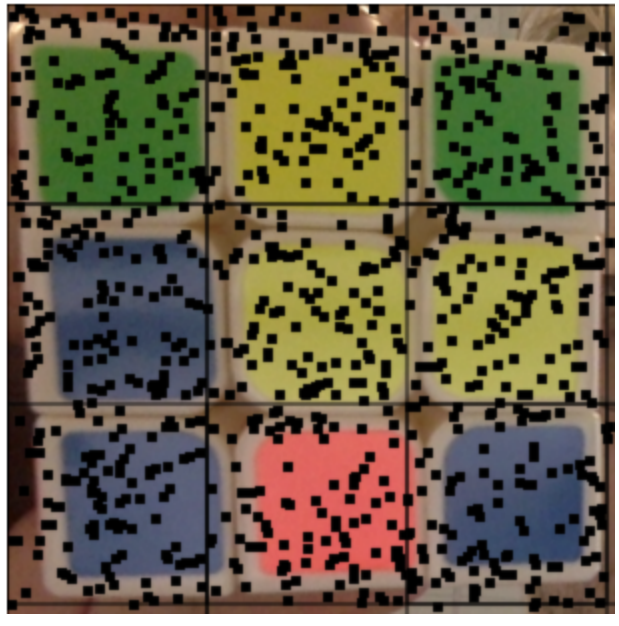

View
x, x1, y, y1
samples
avg
totals
f(x)
sampleN(n)
f(x)
addSamples(n)

Computer Vision - Webcam

Webcam support in *modern* browsers is decent but we did use a polyfill created by Google for the ImageCapture API so we could draw to Canvas.

We next added a loop to update the frames at **30fps**.

Lastly we added a 3x3 grid of "views" with the ability to take an arbitrary number of sampling coordinates.

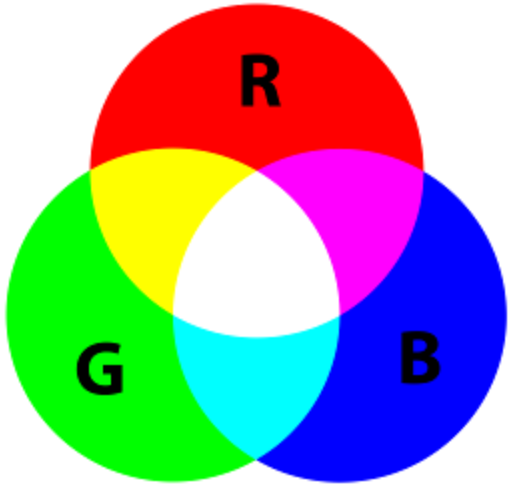


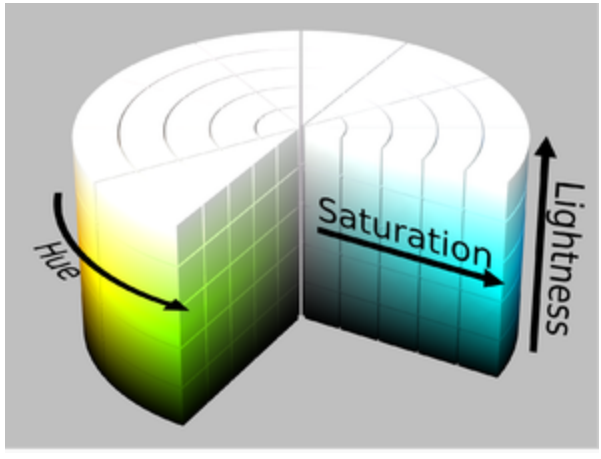
Computer Vision - ImageData

With the Canvas we get RGB values for each sample from the `Uint8ClampedArray` provided by ImageData.

RGB colors are then sent back to the `View` where they are they are added to `total` and a new `avg` is calculated.

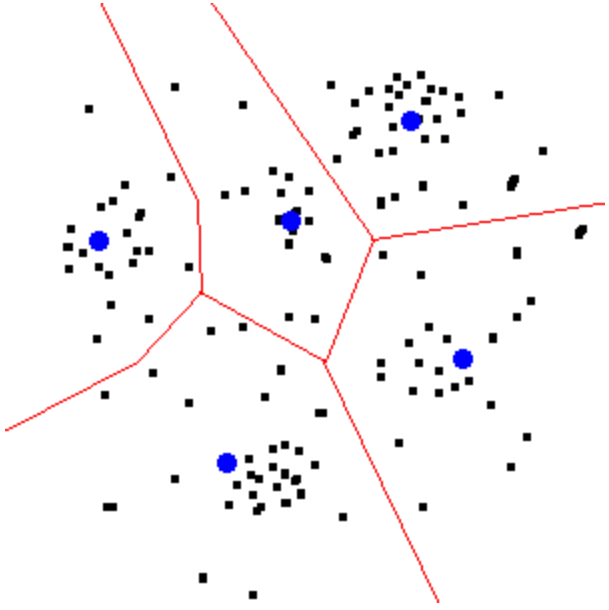
Computer Vision - HSL vs. RGB



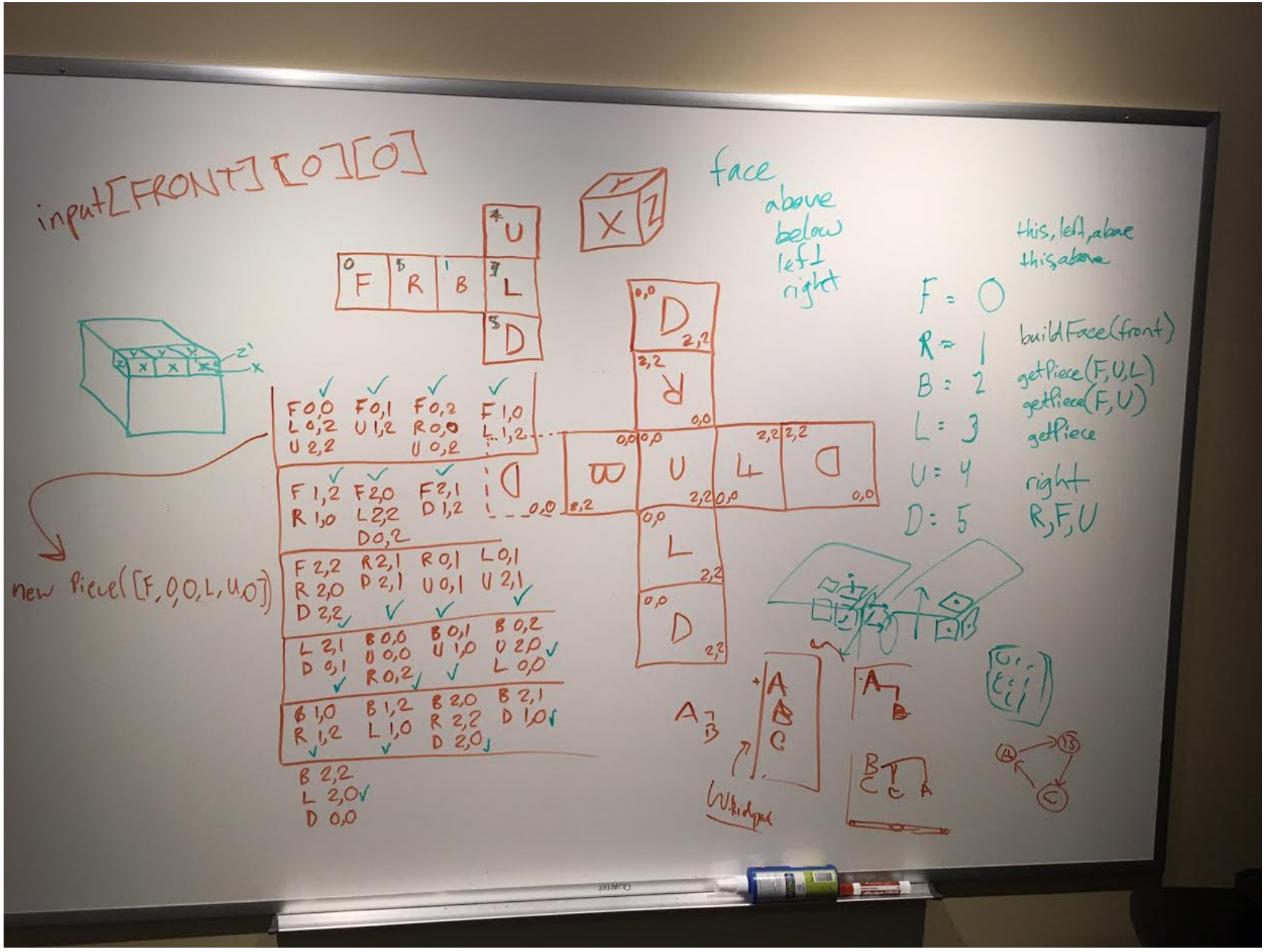


Computer Vision - K-Means Clustering

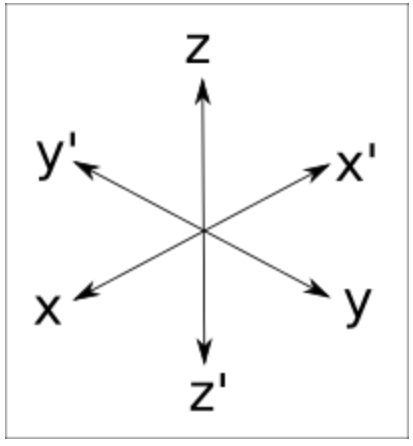
We finally use K-Means Clustering to match the final colors into 6 groups of 9 this data can now be sent to the Cube factory to create a new cube representation.



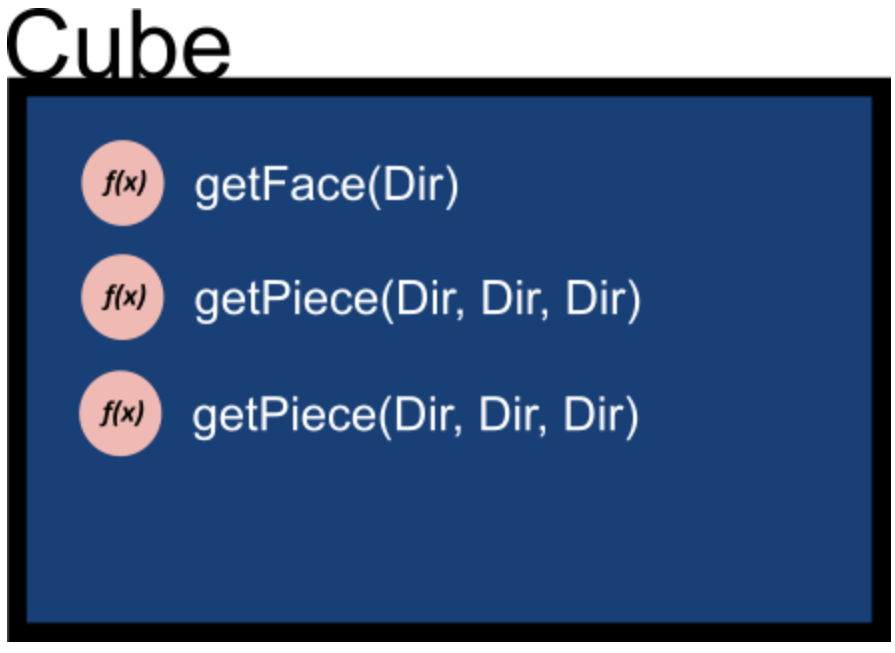
Cube Representation & API



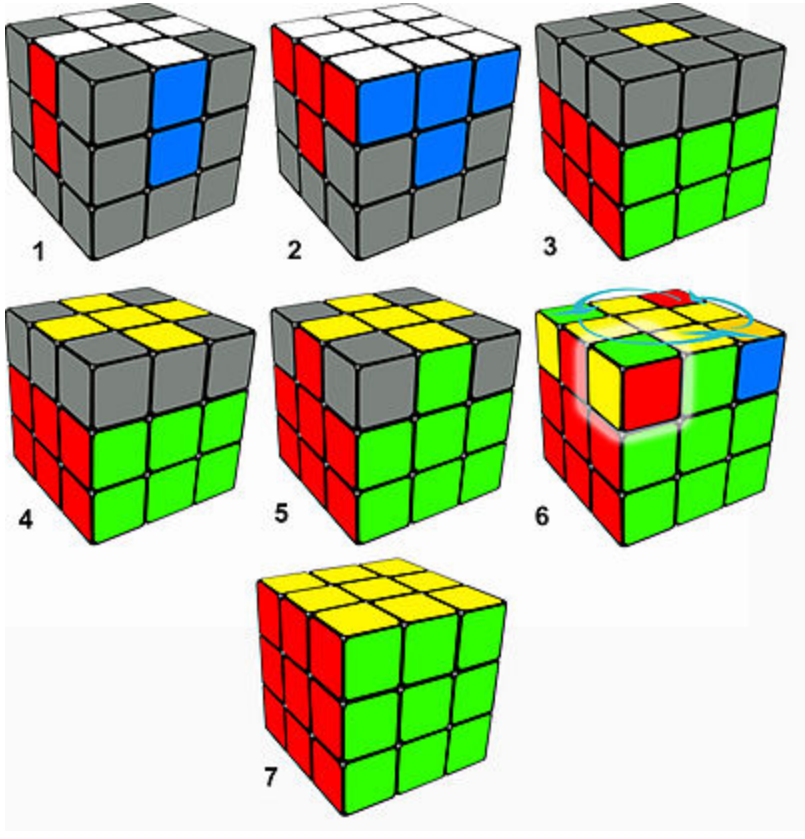
Cube Representation & API



We start by defining faces, and from there we model each of the cubes 26 peices by face.


Cube
f(x)
getFace(Dir)
f(x)
getPiece(Dir, Dir, Dir)
f(x)
getPiece(Dir, Dir, Dir)

Solver & Strategies - Solving in layers



Solver - High Level Implementation

```
let solver = new Solver(cube);
let phase = solver.phase();
while (!cube.isSolved()) {
  if(phase.isComplete()) {
      phase = solver.next();
  } else {
      phase.getNextSteps();
  }
}
```

Solver & Strategies - Algorithms

```
if (edgePiece.color1 === edgePiece.face2
    && edgePiece.color2 === edgePiece.face1) {
  this.transformList.push(newTransform(touchesFace, true));
  this.transformList.push(newTransform(faces.DOWN, false));
  this.transformList.push(newTransform(pivotFace, true));
  this.transformList.push(newTransform(faces.DOWN, true));
}
else if (edgePiece.color1 === edgePiece.face2
    || edgePiece.color2 === edgePiece.face1) {
  if (touchesFace === opposite(color1)
      || touchesFace === opposite(color2)) {
    this.transformList.push(newTransform(touchesFace, true));
    this.transformList.push(newTransform(faces.DOWN, true));
    this.transformList.push(newTransform(pivotFace, true));
    this.transformList.push(newTransform(faces.DOWN, false));
}
```

Visualization - ThreeJS

ThreeJS (https://threejs.org/) is a JavaScript 3D library built on WebGL (https://goo.gl/HsyJUh). It supports rendering interactive 2D and 3D graphics inside and HTML5 canvas element. ThreeJS was chosen because Stuart had experience with it.

We started by drawing a cube. We then changed each side to use different colors (imitating a solved cube). Lastly each side was broken into a grid of colors.