

타임스크립트와 캔버스로 개발한, 토이 프로젝트

컨스택츠 김대현



발표 내용

0. 무엇을 왜 만드나?
1. 타입스크립트 특징 | 동적 타입 vs. 정적 타입 선택
2. 캔버스 | WebGL three.js
3. 개발 구현 상세 이야기
4. 토이 프로젝트

타입스크립트와 캔버스로
개발한, 토이 프로젝트

컨스택츠 김대현



0. 무엇을 왜 만드느냐?

잠깐! 토이 프로젝트...

토이 프로젝트 (1) - **과정 자체가 놀이 거리**인 프로젝트

토이 프로젝트 (2) - **결과물이 장난감 수준**인 프로젝트

토이 프로젝트 (3) - **장난감에 대한** 프로젝트

놀이로 배우고 연습하자

루빅스 큐브 퍼즐 장난감

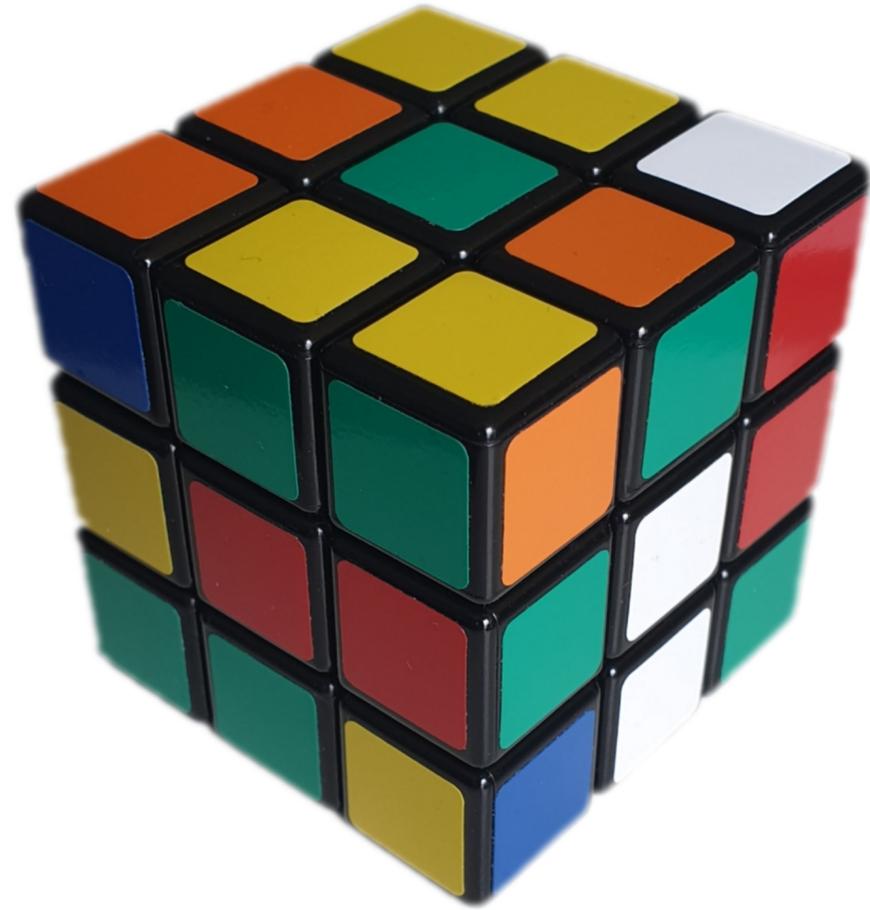


사진 출처: Wikipedia



스피드 큐브 | Speed Cube

기본 알고리즘은 단계별 풀이 알고리즘을 외워서 반복 풀이

외울 패턴 적고, 움직임 회전수 많음 → 푸는 시간이 **오래 걸림**

중급 알고리즘은 다양한 패턴마다 최소 회전을 외워서 풀이

외울 패턴 많고, 움직임 회전수 적음 → **빨리 풀 수 있다.**

문제는, 그걸 다 외우고, 재빨리 판단할 수 있나? → 훈련 필요

(중급) 큐브 퍼즐 풀이법 | F2L 일부

Basic Inserts



$U (R U' R')$



$y' (R' U' R)$
 $y (L' U' L)$

$y' U' (R' U R)$
 $y U' (L' U L)$



$(R U R')$



F2L Case 1



$U' (R U' R' U) y' (R' U' R)$
 $y' U (R' U' R U') (R' U' R)$

$U' (R U R' U) (R U R')$



$U' (R U 2' R' U) y' (R' U' R)$
 $U' (R U 2' R') d (R' U' R)$

$R' U 2' R 2 U R 2' U R$
 $y' U (R' U 2 R) U' y (R U R')$
 $(R U' R' U) (R U' R') U 2 (R U' R')$

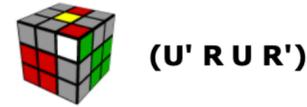


$y' U (R' U R U') (R' U' R)$

$U' (R U' R' U) (R U R')$

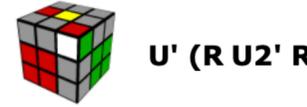


F2L Case 2



$(U' R U R') U 2 (R U' R')$

$y' (U R' U' R) U 2' (R' U R)$
 $d (R' U' R) U 2' (R' U R)$
Note - $(y' U)$ and (d) are interchangeable



$U' (R U 2' R') U 2 (R U' R')$

$y' U (R' U 2 R) U 2' (R' U R)$
 $d (R' U 2 R) U 2' (R' U R)$



F2L Case 3



$U (R U 2 R') U (R U' R')$

$y' U' (R' U 2 R) U' (R' U R)$



$U 2 (R U R' U) (R U' R')$
 $(R U' R') U 2 (R U R')$

$y' U 2 (R' U' R) U' (R' U R)$
 $F' L' U 2 L F$
Note - The second algorithm is fewer moves, but less intuitive and less finger-friendly.



Incorrectly Connected Pieces



$y' (R' U R) U 2' y (R U R')$
 $(R U R') U 2 (R U' R' U) (R U' R')$

$(R U' R' U 2) y' (R' U' R)$
 $U F (R U R' U') F' (U R U' R')$



$(R U 2 R') U' (R U R')$

$y' (R' U 2 R) U (R' U' R)$



$U (R U' R' U') (R U' R' U) (R U' R')$
 $(R U R' U 2') (R U R' U') (R U R')$

$y' U' (R' U R U) (R' U R U') (R' U R)$
 $F (U R U' R') F' (R U' R')$



Corner in Place, Edge in U Face



$U' F' (R U R' U') R' F R$
 $R' F' R U (R U' R') F$

$U (R U' R') U' (F' U F)$
 $U (R U' R') (F R' F' R)$



$(R U' R' U) (R U' R')$

$y' (R' U R U') (R' U R)$



$y' (R' U' R U) (R' U' R)$
 $(R' F R F') U (R U' R')$

$(R U R' U') (R U R')$



Edge in Place, Corner in U face



$(R U' R' U) y' (R' U R)$
 $U' (R' F R F') (R U' R')$

$(U R U' R') (U R U' R') (U R U' R')$



$(U' R U' R') U 2 (R U' R')$

$U (R U R') U 2 (R U R')$

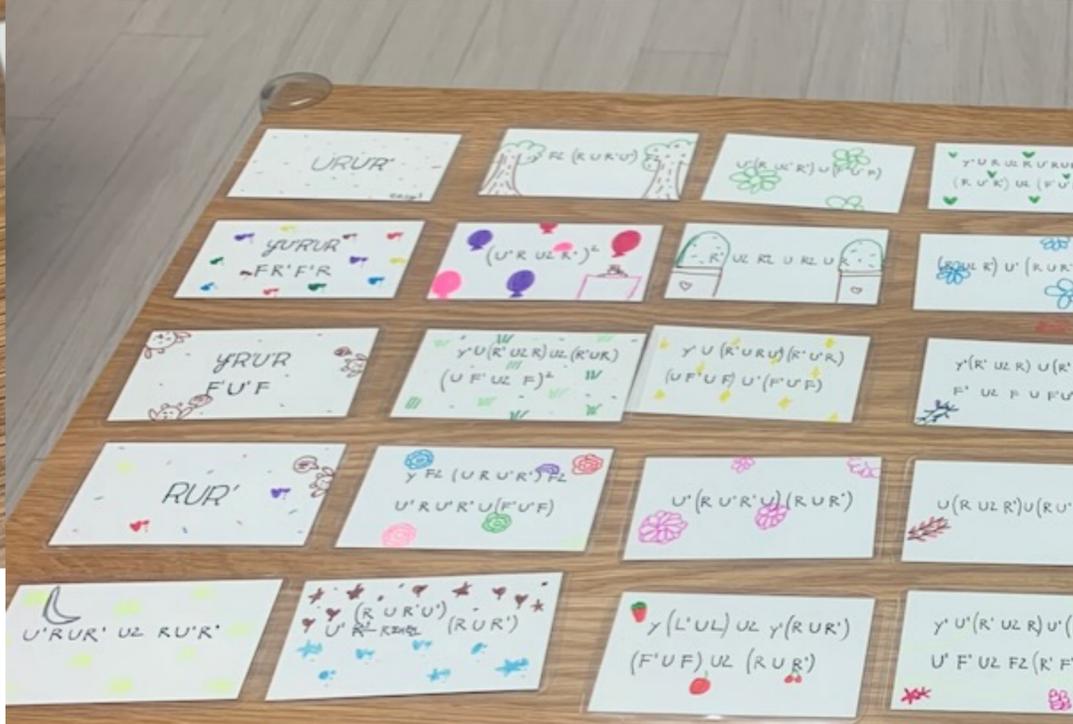
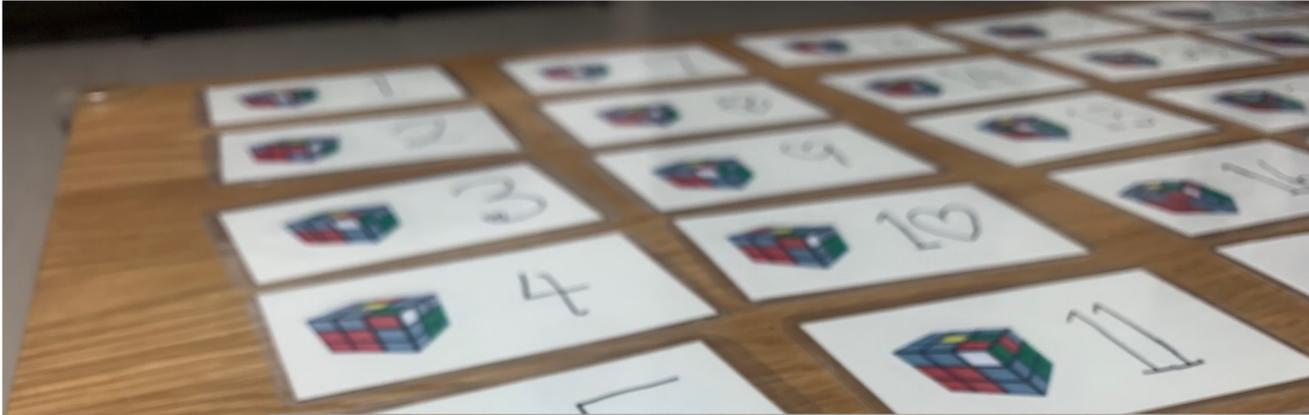


$(U' R U R') U y' (R' U' R)$

$U (F' U' F) U' (R U R')$



(중급) 큐브 퍼즐 풀이법 | 암기 카드



그래서 프로젝트로 하려는 건...

풀이 패턴을 암기하기 위한 **큐브 연습용 웹페이지**를 만들자.

아직 외우지 못한 패턴을 문제로 내고, 풀어보게 연습.

적어도 내게는 유용할 거라는 기대 (합리화)

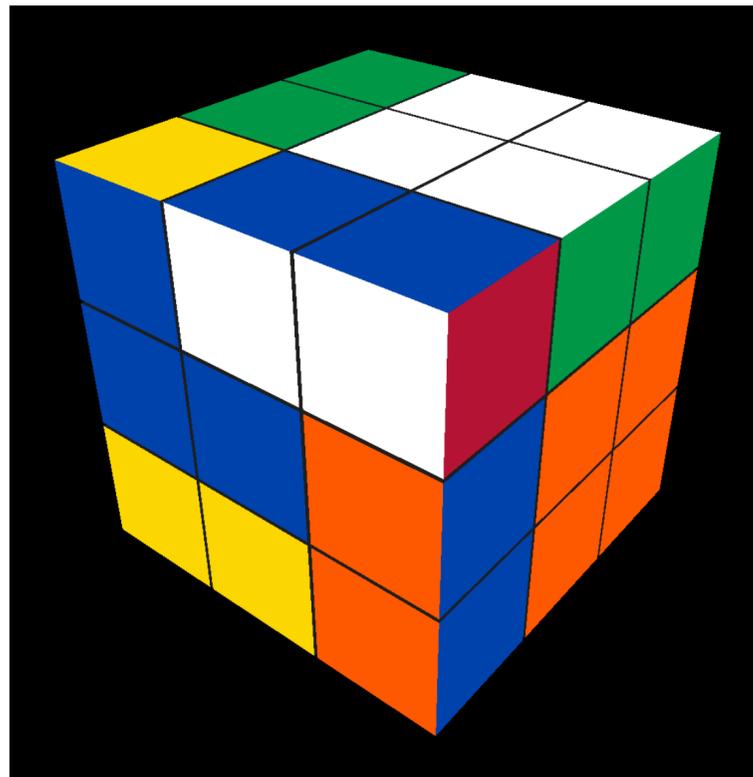
Step1. 일단 **큐브 기본 움직임**을 구현해 보자

프로젝트 데모

<https://hatemogi.github.io/rurucube>

큐브가 보이면, R, U, L, F 키 등을 눌러보세요

ESC | 최초 상태로 복귀, SPC | 카메라 위치 복귀, RET | 섞기



1. TypeScript

타입스크립트 | TypeScript

JavaScript에 **타입 시스템**을 입혀서 기능을 확장한 언어

컴파일 시점에 정적^{static} 타입 검사

실행 시점을 위한 JavaScript 코드 생성

.ts → 컴파일 → .js → (브라우저) JS 실행기

가장 큰 특징은 아무래도 **"정적 타입 시스템"**일 터

예제 코드 | JavaScript

```
src > JS sample.js > ...  
1   var a = 1;  
2  
3   if (a < 3) {  
4     |   a = "Jeju";  
5   }  
6  
7   console.log(a.length);  
8
```

예제 코드 | TypeScript

```
src > TS sample.ts > ...
```

```
1   var a = 1;
```

```
2
```

```
3   if (a < 3) {  
4     a = "Jeju";  
5   }
```

```
6
```

```
7   console.log(a.length);  
8  
9  
10  
11
```

any

'number' 형식에 'length' 속성이 없습니다. ts(2339)

문제 보기 빠른 수정을 사용할 수 없음

단순하게 보자면...

정적static 타입 검사

컴파일 타임에 미리 타입을 확인

타입 오류를 컴파일 타임에 발견 → **컴파일 실패**

동적dynamic 타입 검사

런타임에 타입 확인

타입 오류를 런타임에 발견 → **런타임 오류**

동적 타입 언어, 정적 타입 언어

JavaScript

Python

Ruby

Perl

Clojure

TypeScript

Java

Kotlin

Scala

Haskell

타입 시스템 장단점

정적static 타입 언어

타입을 일일이 맞춰야 하는 번거로움 | 타입 추론

런타임에 타입 오류 발생 거의 없음

동적dynamic 타입 언어

변수나 함수 선언시 타입 선언 안 하고 편하게 씀

런타임에 타입 오류 발생 가능 | Linter등으로 정적 분석

장단점 명확. 흔히 각자 주력 언어에 따라 호불호 갈리더라

둘 다 해보고 골라도 늦지 않다.

2. Canvas | Three.js

HTML5 Canvas

웹페이지 안에 `<canvas id={} width={} height={} />` 태그

JavaScript로 캔버스 안에 그릴 내용을 코딩

2차원 그래픽 컨텍스트로 평면 그림을 그리거나,

WebGL 컨텍스트에 3D 그래픽 그릴 수 있다.

Three.js | JavaScript 3D Library

JavaScript 3차원 그래픽 라이브러리

WebGL을 이용해서, Canvas에 3D 그래픽 렌더링

인프런 강의 | three.js로 시작하는 3D 인터랙티브 웹

three.js r141

Learn

[documentation](#)
[examples](#)
[editor](#)

Community

[questions](#)
[discord](#)
[forum](#)
[slack](#)
[twitter](#)

Code

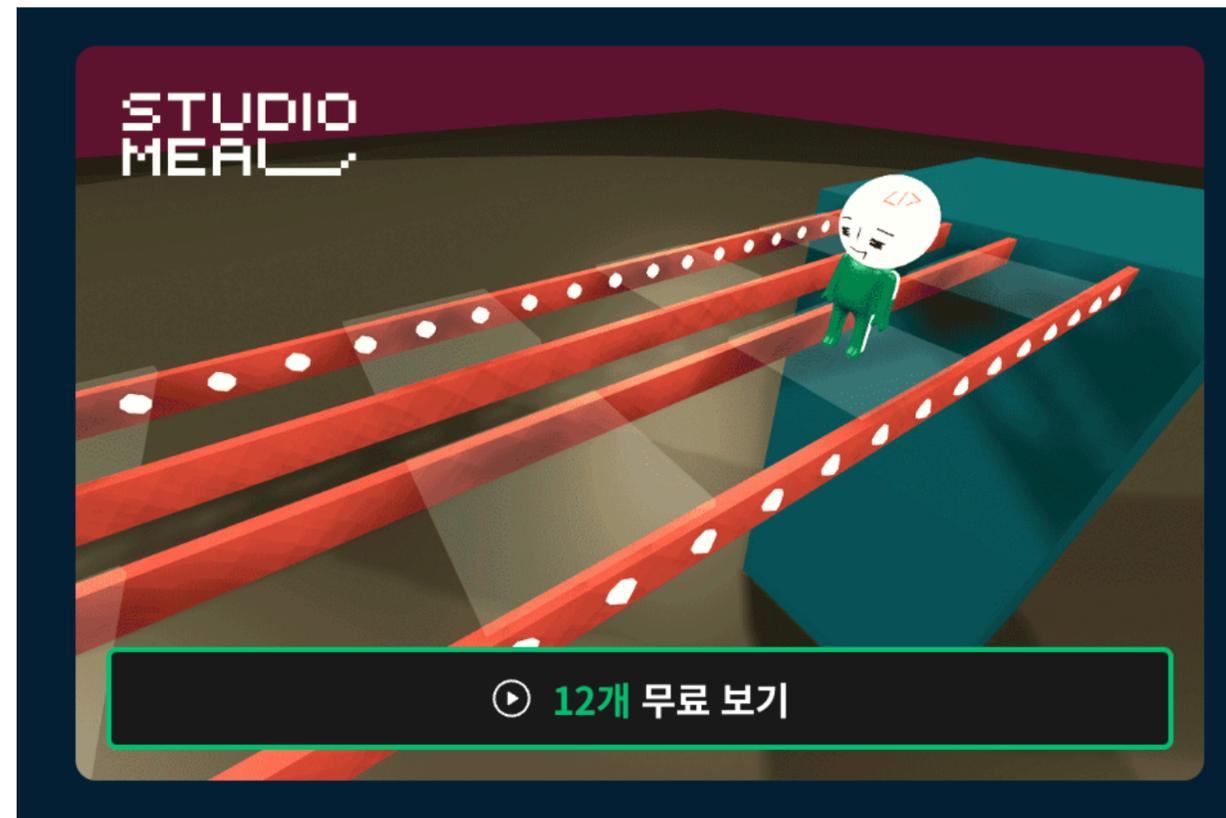
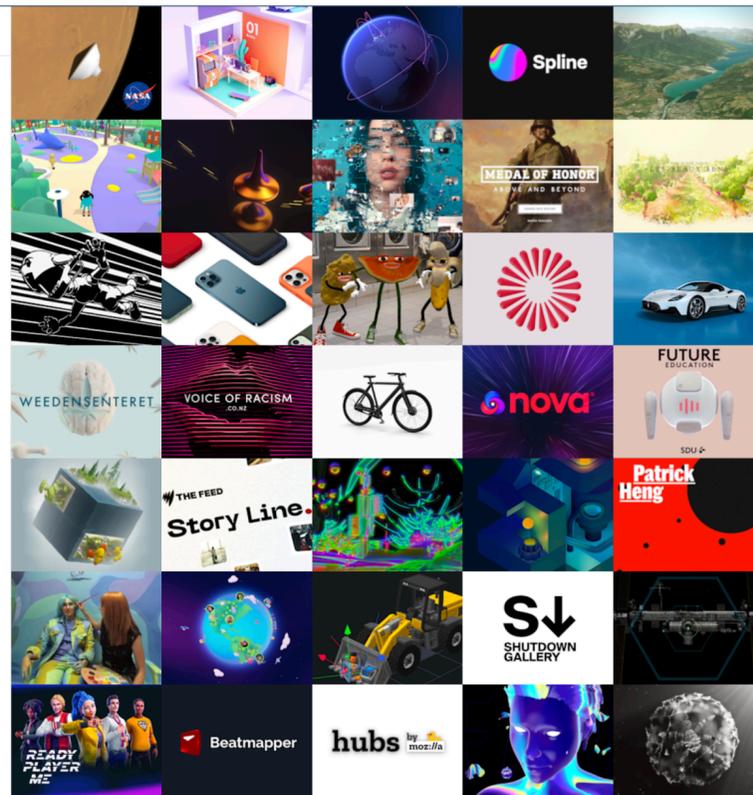
[github](#)
[download](#)
[devtools](#)

Resources

[Three.js Fundamentals](#)
[Three.js Journey](#)
[Learn Three.js](#)
[初めてのThree.js](#)

Merch

[T-Shirts](#)



Three.js 구성요소

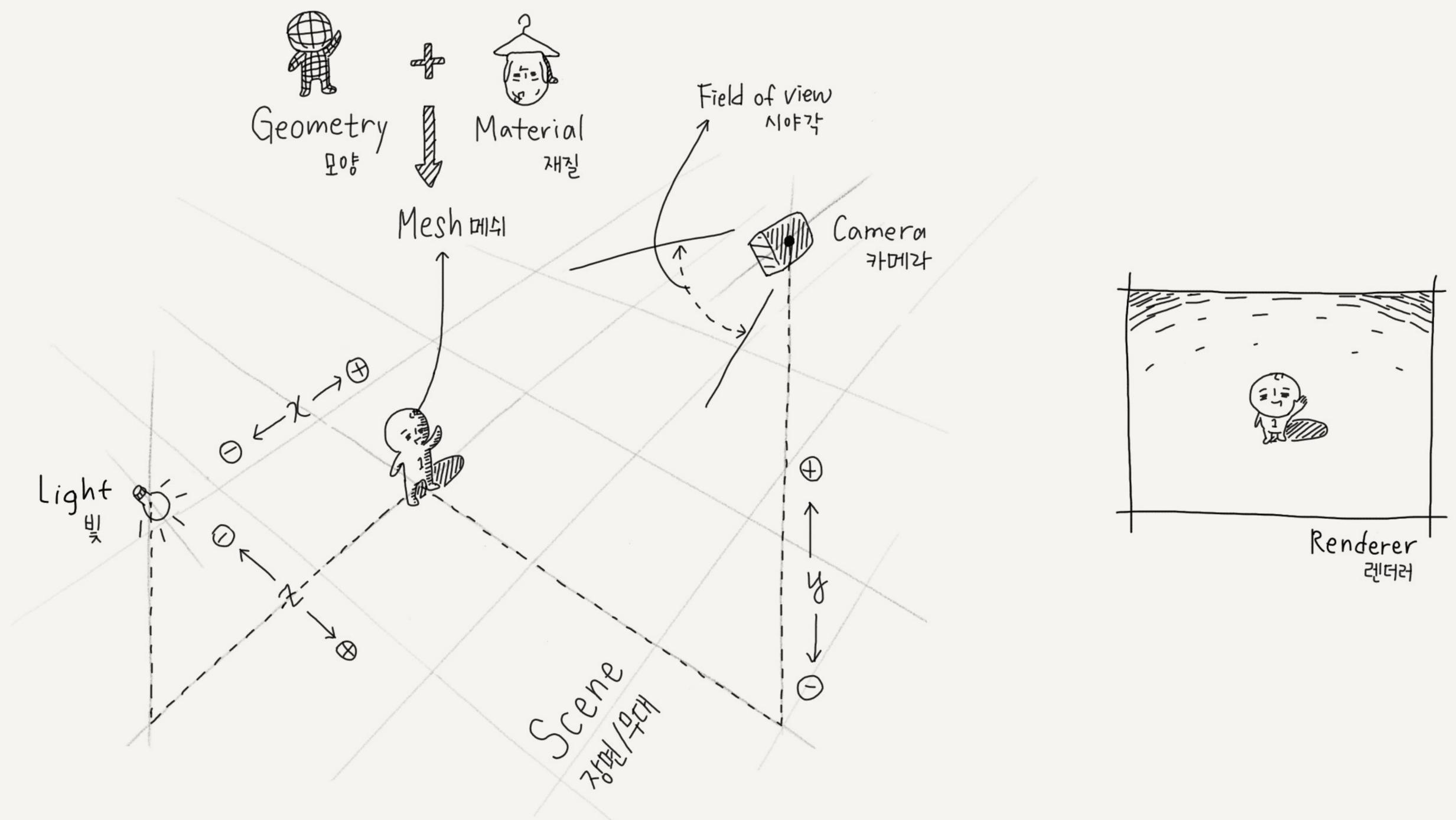
Mesh | 물체 Geometry + Material

Camera | 카메라 시점

Light | 조명

Scene | 무대

위 구성 요소들을 준비하고 배치해서, `<canvas/>`에 렌더링



그리고, 물론 TypeScript로도 활용 가능

```
npm install @types/three
```

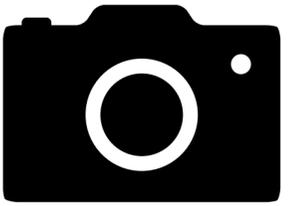
<https://www.npmjs.com/package/@types/three>

3. 개발 구현 이야기

개발의 묘미 하나

마음껏 상상한 무언가를
모델로 추상화해서, 본질을 추려내고
만들어 보이고, 움직이기도 한다

내가 상상하면 현실이 된다



역할을 (잘게) 나누고, 담당 역할을 책임진다

MVC 모델/뷰/컨트롤 **역할 분리**.

해야할 일을 책임지고, 각자 할 일에 집중.

결과가 잘 합쳐지는가? ← 좋은 설계 디자인에 달려있다.

큐브 모델 | 큐브의 **현재 상태**와, 회전 후 **다음 상태**만 관리

애니메이션 뷰 | 화면에 그려야할 내용과, 상태들 사이 애니메이션 처리

컨트롤 모델 | 키 입력 등의 제어에 집중

큐브 모델의 본질 (1) 색상

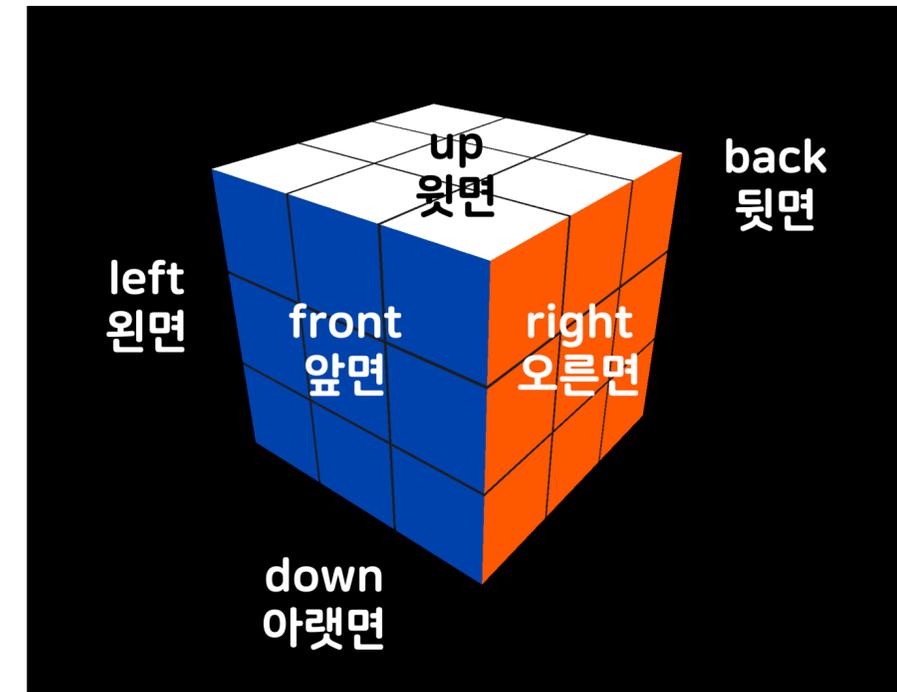
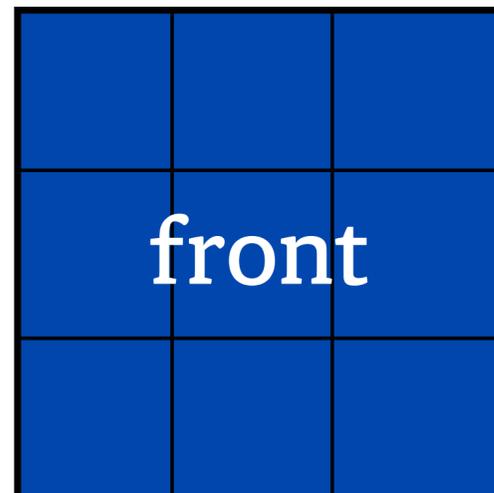
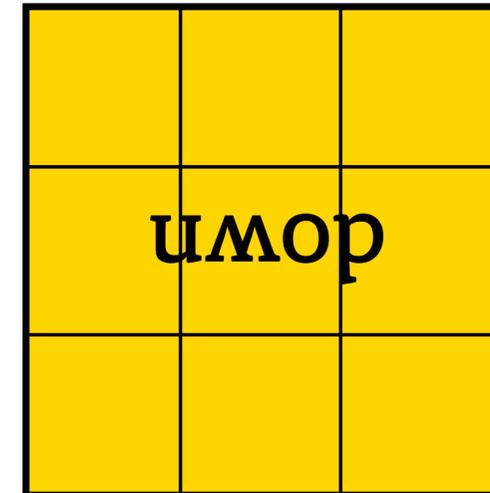
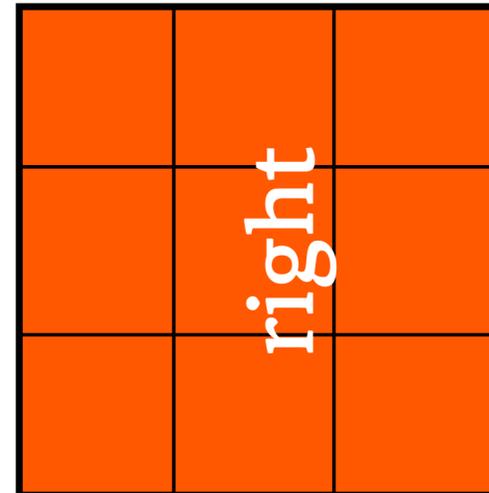
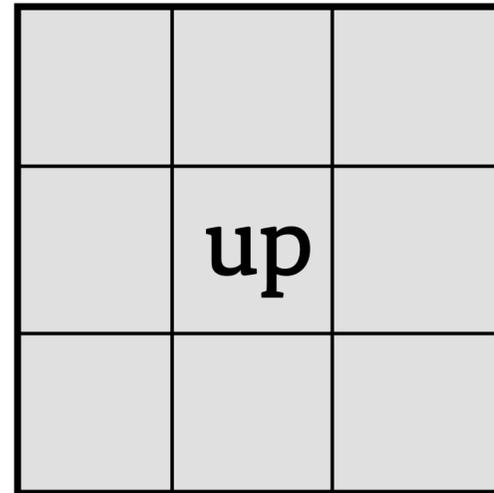
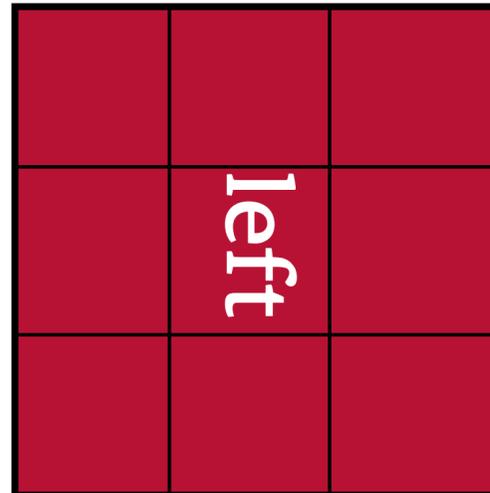
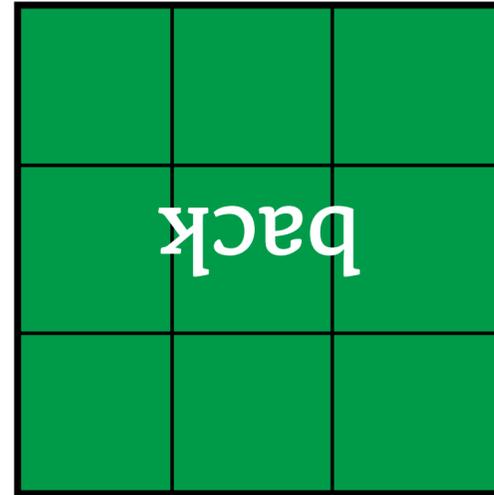
큐브 조각 단면 하나는 6가지 색상 중 하나다.

~~그리고, 조각 안쪽 면들은 색이 보이지 않는다 | NONE~~

```
src > TS model.ts > ...  
1   enum FaceColor {  
2   |   WHITE, YELLOW, BLUE, GREEN, RED, ORANGE, NONE  
3   };  
4
```



큐브 모델의 본질 (2) 겉 면



큐브 모델의 본질 (3) 한 면 | 색상 9개 배열

6	7	8
3	4	5
0	1	2

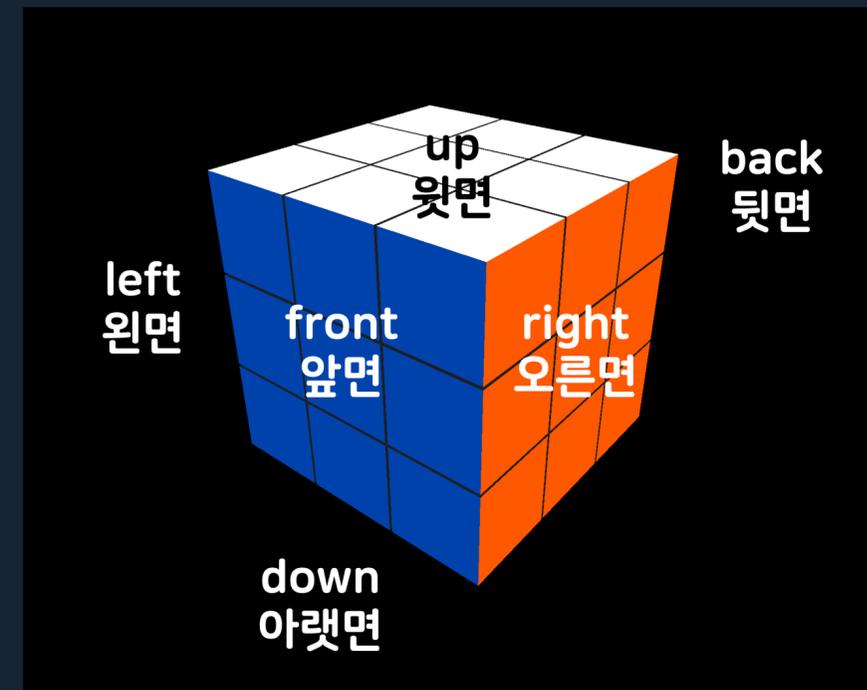
```
5 | type FaceA = FaceColor[];  
6 |  
7 | // OR ...  
8 |  
9 | type FaceB = [FaceColor, FaceColor, FaceColor,  
10 | | FaceColor, FaceColor, FaceColor,  
11 | | FaceColor, FaceColor, FaceColor];  
12 |
```

큐브 모델의 본질 | 모델 코드

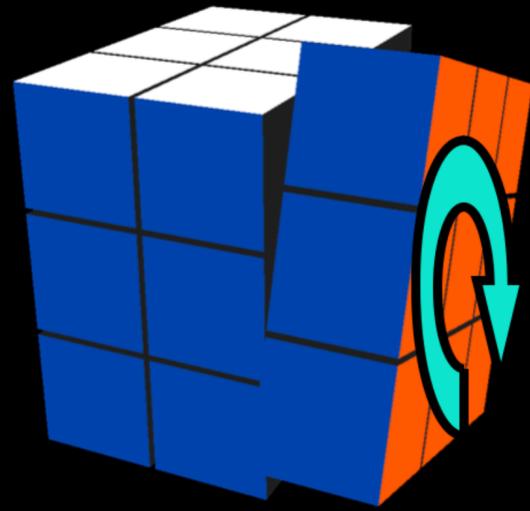
```
1 enum FaceColor {  
2     WHITE, YELLOW, BLUE, GREEN, RED, ORANGE, NONE  
3 };
```

```
4  
5 type Face = FaceColor[];
```

```
6  
7 ▶ type Cube = {  
8     front : Face,  
9     back  : Face,  
10    up    : Face,  
11    down  : Face,  
12    left  : Face,  
13    right : Face  
14 };  
15
```

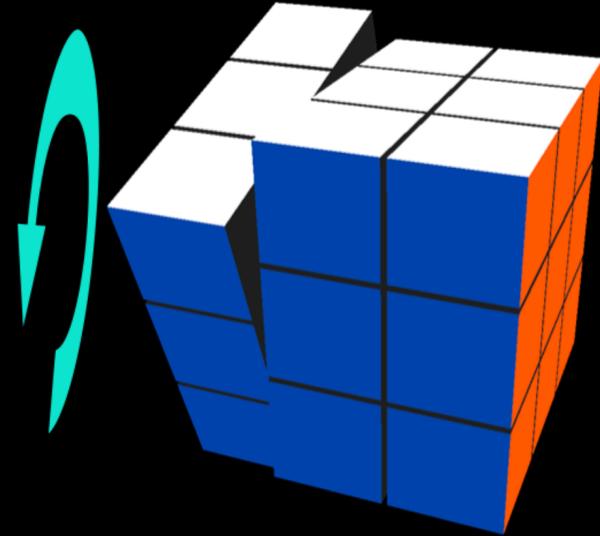
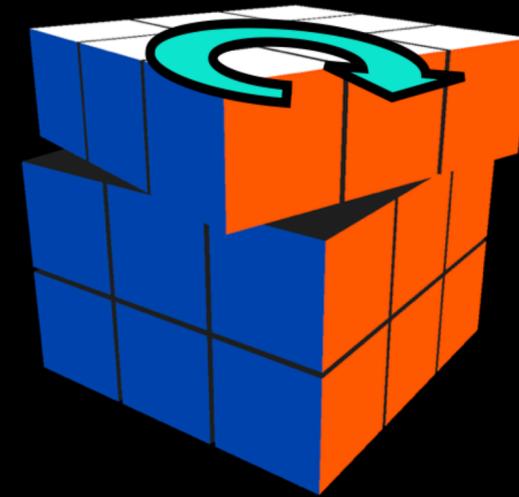


큐브 모델 회전



R
오른쪽면 90°
시계방향 회전

U
윗면 90°
시계방향 회전



L
왼쪽면 90°
시계방향 회전



F
앞면 90°
시계방향 회전

큐브 모델 회전 | U U' R R'

표기	의미	설명
U	윗면 회전	맨 윗면을 위에서 바라본다고 치고, 시계 방향으로 90도 회전
U'	윗면 회전'	맨 윗면을 위에서 바라볼 때, 반시계 방향으로 90도 회전
R	오른면 회전	내 오른손 쪽을 정면에서 바라볼 때 시계 방향으로 90도 회전
R'	오른면 회전'	내 오른손 쪽을 정면에서 볼 때, 반시계 방향으로 90도 회전
L, L'	왼쪽면 회전	내 왼손쪽 면을 시계/반시계 방향으로 90도 회전
F, F'	앞면 회전	내가 보고 있는 앞면을 시계/반시계 방향으로 90도 회전
U2	윗면 두 번 회전	3층을 180도 회전

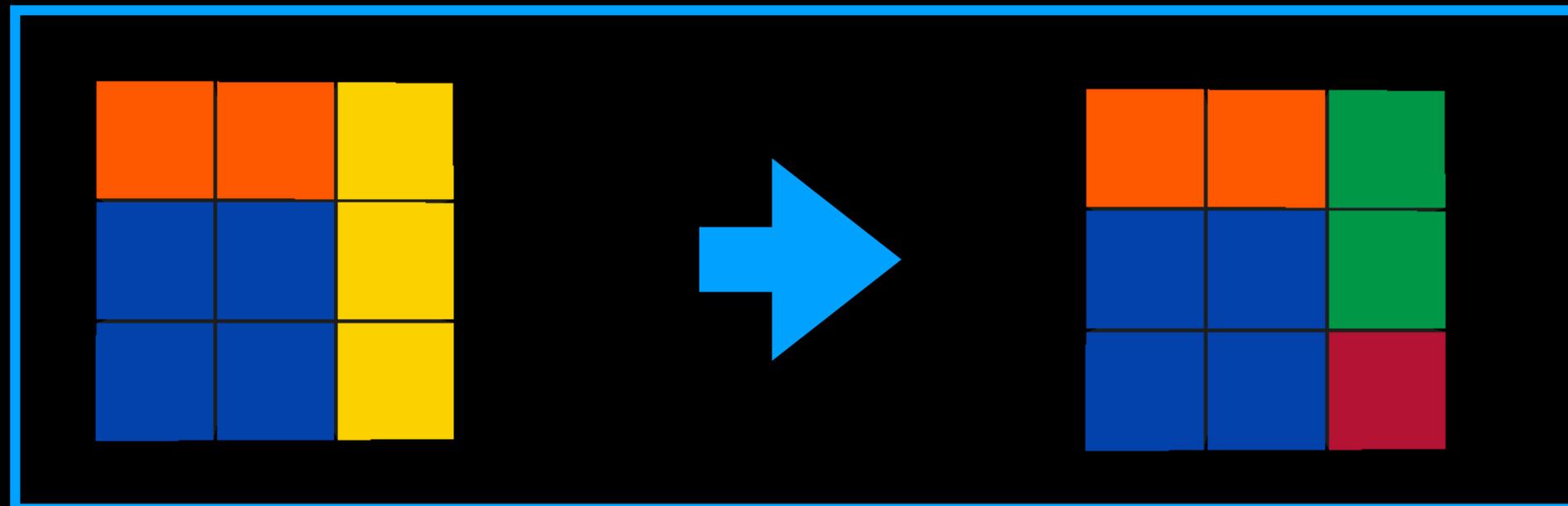
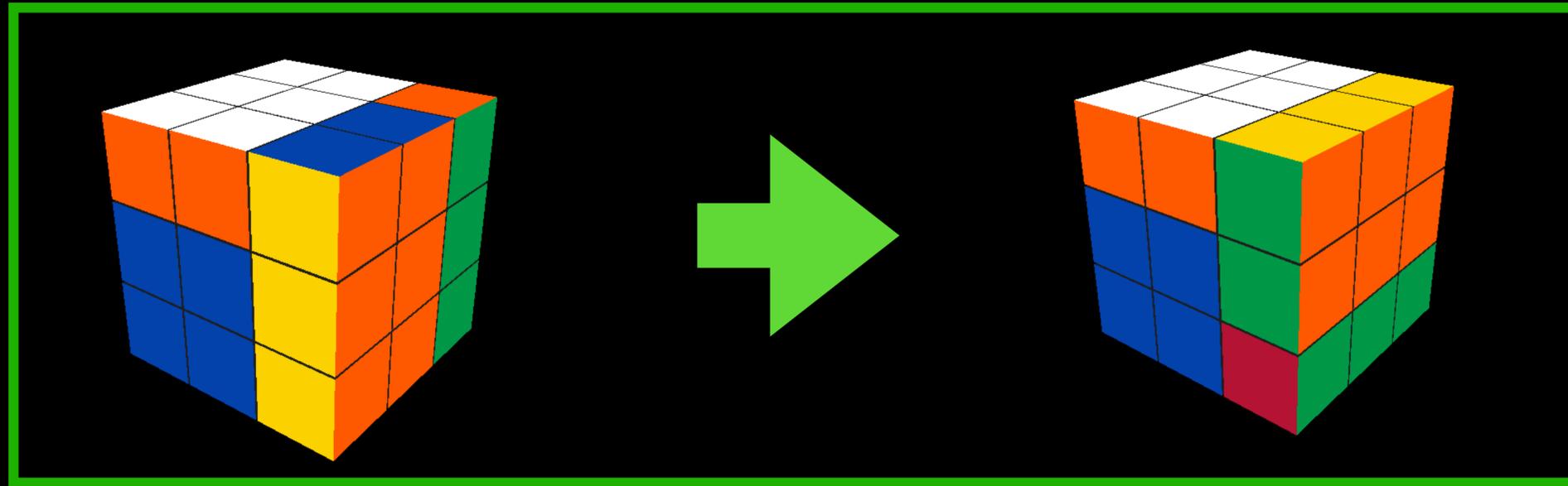
함수 타입 선언 | 큐브 변환, 면 변환

64

```
type CubeFunc = (c: Cube) => Cube;
```

65

```
type FaceFunc = (f: Face) => Face;
```



기본 함수 | 큐브 면을 다루는 보조 함수

```
64 type CubeFunc = (c: Cube) => Cube;
```

```
65 type FaceFunc = (f: Face) => Face;
```

```
66
```

```
67 function copyFace(from: Face, to: Face, toIndexes: number[]): Face {
```

```
68     const result = to.concat([]);
```

```
69     toIndexes.forEach(ti => result[ti] = from[ti]);
```

```
70     return result;
```

```
71 }
```

다른 면의 일부를 복사

```
72
```

```
73 function reorderFace(from: (n: number) => number): FaceFunc {
```

```
74     return (face: Face) => {
```

```
75         const target = face.concat([]);
```

```
76         face.forEach((x, n) => target[from(n)] = x);
```

```
77         return target;
```

```
78     }
```

```
79 }
```

한 면 내의 조각들을 재배치

```
80
```

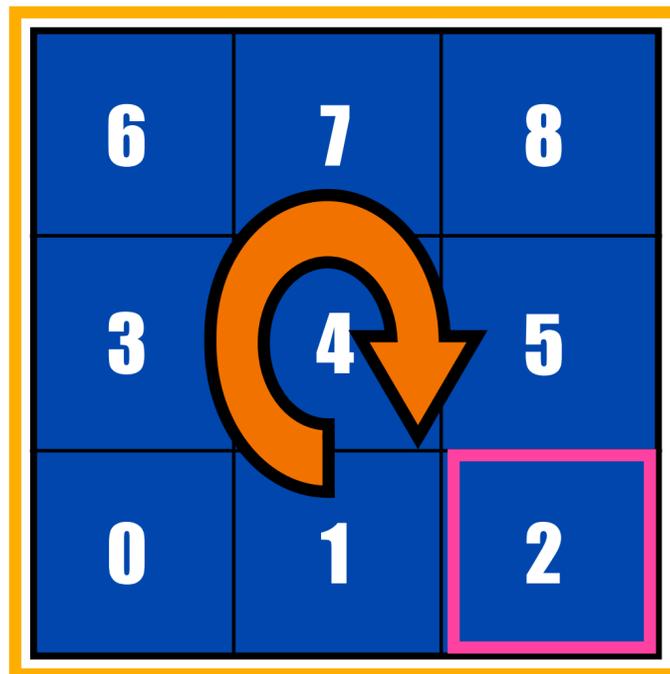
reorderFace() 이용한 "면" 회전 함수

```
81 // const clockwiseF      = (x: number) => (7 * x + 6) % 10;  
82 // const counterclockwiseF = (x: number) => (3 * x + 2) % 10;  
83 // const halfTurnF       = (x: number) => 8 - x;
```

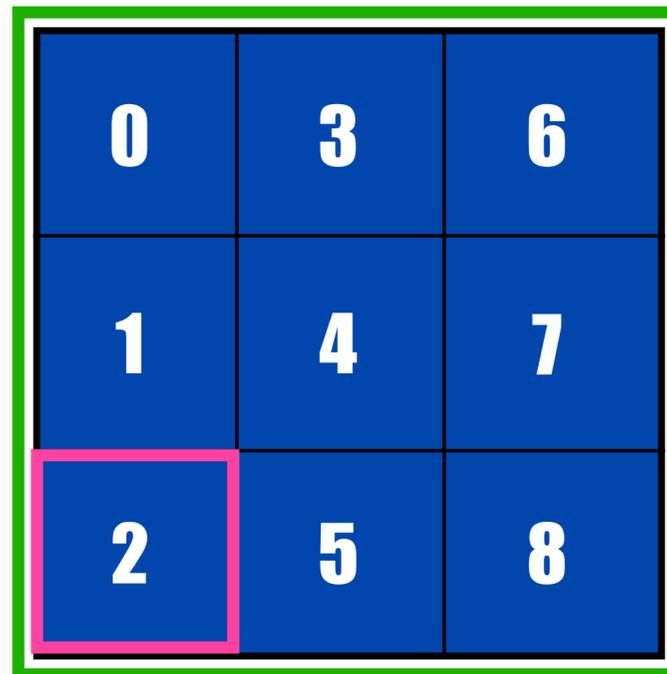
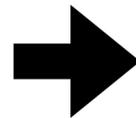
재배치 공식

```
84  
85 // const clockwise      = reorderFace(clockwiseF);  
86 // const counterclockwise = reorderFace(counterclockwiseF);  
87 // const halfTurn       = reorderFace(halfTurnF);
```

면 회전 함수



6	7	8
3	4	5
0	1	2



0	3	6
1	4	7
2	5	8

2	0
5	1
8	2
1	3
4	4
7	5
0	6
3	7

면 회전 함수를 이용한 "큐브 회전" 함수

```
89 const moveU: CubeFunc = ({front, back, up, down, left, right}) =>
90   ({ front: copyFace(right, front, [6,7,8]),
91     back : copyFace(left,  back, [6,7,8]),
92     up   : clockwise(up),
93     down,
94     left : copyFace(front, left, [6,7,8]),
95     right: copyFace(back,  right, [6,7,8])
96   });
```

```
97
98 const moveR: CubeFunc = ({front, back, up, down, left, right}) =>
99   ({ front: copyFace(down,      front, [2,5,8]),
100     back : copyFace(halfTurn(up), back, [0,3,6]),
101     up   : copyFace(front,      up,    [2,5,8]),
102     down : copyFace(halfTurn(back), down, [2,5,8]),
103     left,
104     right: clockwise(right)
105   });
```

```
106
```

U, R 등 회전 구현 후, 반시계 방향도 만든다...

모든 시계방향 회전에 대응하는 반시계 방향 회전을 구현했는데...

$$R' = RRR$$

$$U' = UUU$$

$$F' = FFF$$

-90도 회전은 270도 회전과 결과가 같다

따로 만들 필요 없었다. 아깝지만, 과감히 버리자!

머리가 나쁘면, 손발이 고생한 케이스

큐브 단위 테스트 | 'UU' 이동은 원복된다

```
66
67 test('시계 방향 회전과 반시계 방향 회전을 둘 다 하면 원래 상태로', () => {
68   allSlices.forEach(slice => {
69     const clockwise: Move = { slice, prime: false };
70     const counterclockwise: Move = { slice, prime: true };
71     expectNoChange(move(clockwise, counterclockwise));
72     expectNoChange(move(counterclockwise, clockwise));
73   })
74 });
75
```

```
19
20 function expectNoChange(fn: ((c: Cube) => Cube)) {
21   expect(eqCube(testCube, fn(testCube))).toBe(true);
22 }
23
```

큐브 단위 테스트 | U4 이동도 원복된다

```
62
63 test('같은 회전을 4번 하면 원래 상태로', () => {
64     | allMoves.forEach(m => expectNoChange(move(m, m, m, m)));
65 });
66
```

큐브 불변성 검사 | 가운데 조각들은 대칭색

```
32
33 test('어떻게 이동해도 가운데 조각은 서로 반대색', () => {
34     const expectOppositeColor = (given: FaceColor, expected: FaceColor) =>
35         expect(oppositeColor(given)).toBe(expected);
36     allMoves.forEach(m => {
37         const {front, back, up, down, left, right} = move(m)(testCube);
38         const center = 4;
39         expectOppositeColor(front[center], back[center]);
40         expectOppositeColor(left[center], right[center]);
41         expectOppositeColor(up[center], down[center]);
42     });
43 });
44
```

큐브 불변성 검사 | 동일 색상은 총 9조각

```
44
45 test('어떻게 이동해도 한 색상은 9조각씩 있다', () => {
46   allMoves.forEach(m => {
47     const { front, back, up, down, left, right } = move(m)(testCube);
48     const count = (color: FaceColor, face: Face) =>
49       face.filter((c: FaceColor) => c === color).length;
50     const countAll = (color: FaceColor) =>
51       count(color, front) + count(color, back) +
52       count(color, left) + count(color, right) +
53       count(color, up) + count(color, down);
54     [W, Y, R, O, B, G].forEach(color => expect(countAll(color)).toBe(9));
55   });
56 });
57
```

큐브 단위 테스트 | 안심 피드백!

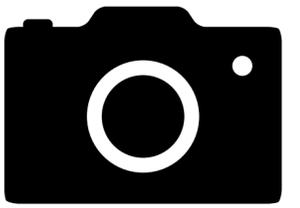
```
> rurucube@0.1.0 test  
> jest
```

PASS test/model.test.ts

- ✓ 반대 색상 (1 ms)
- ✓ 어떻게 이동해도 가운데 조각은 서로 반대색 (4 ms)
- ✓ 어떻게 이동해도 한 색상은 9조각씩 있다 (2 ms)
- ✓ 같은 큐브 비교 (1 ms)
- ✓ 같은 회전을 4번 하면 원래 상태로 (1 ms)
- ✓ 시계 방향 회전과 반시계 방향 회전을 둘 다 하면 원래 상태로 (2 ms)
- ✓ RUR'U' 패턴을 6번 하면 원래 상태로

```
Test Suites: 1 passed, 1 total  
Tests:       7 passed, 7 total  
Snapshots:   0 total  
Time:        1.245 s  
Ran all test suites.
```

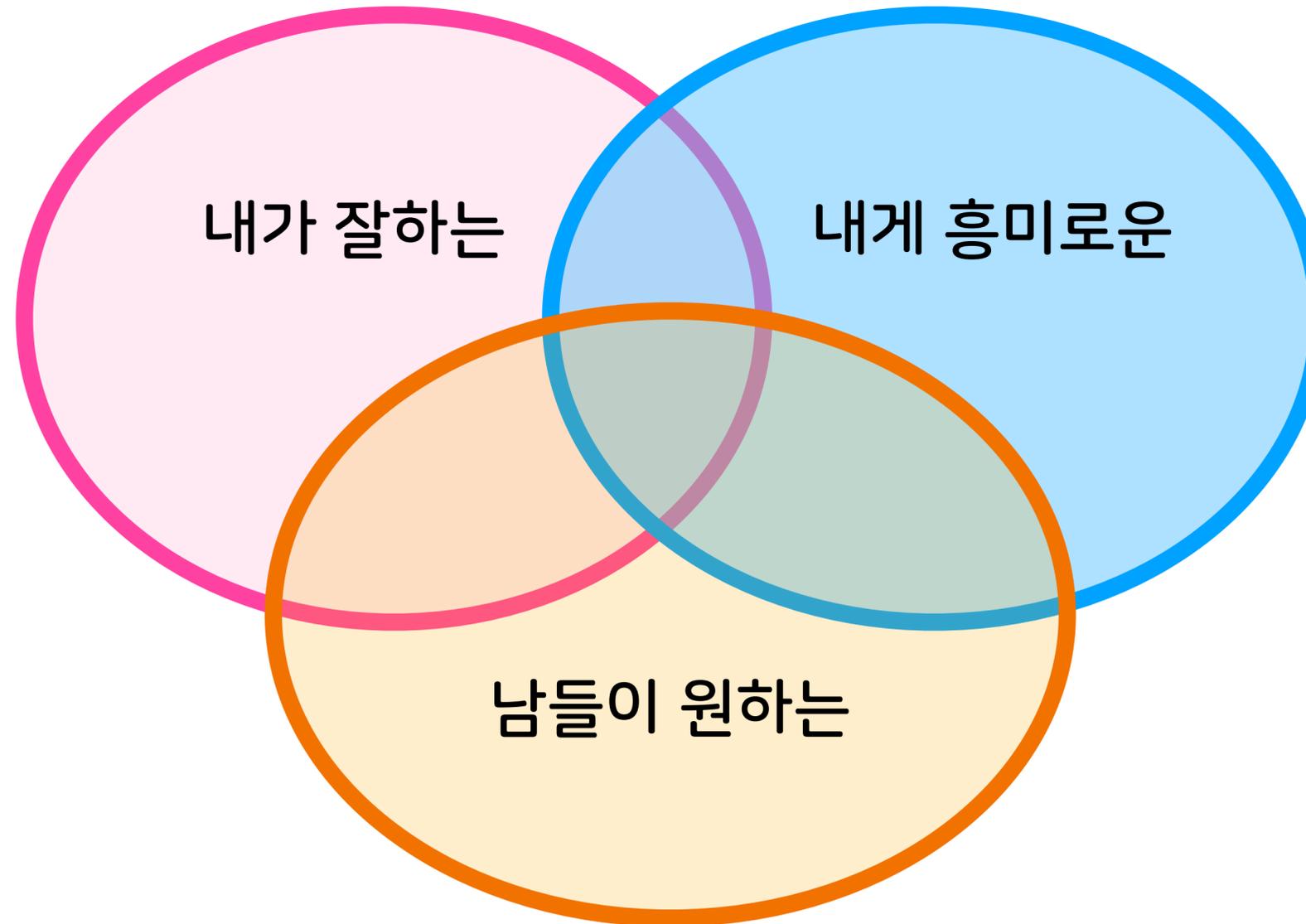
소프트웨어 프로젝트 공통 전략



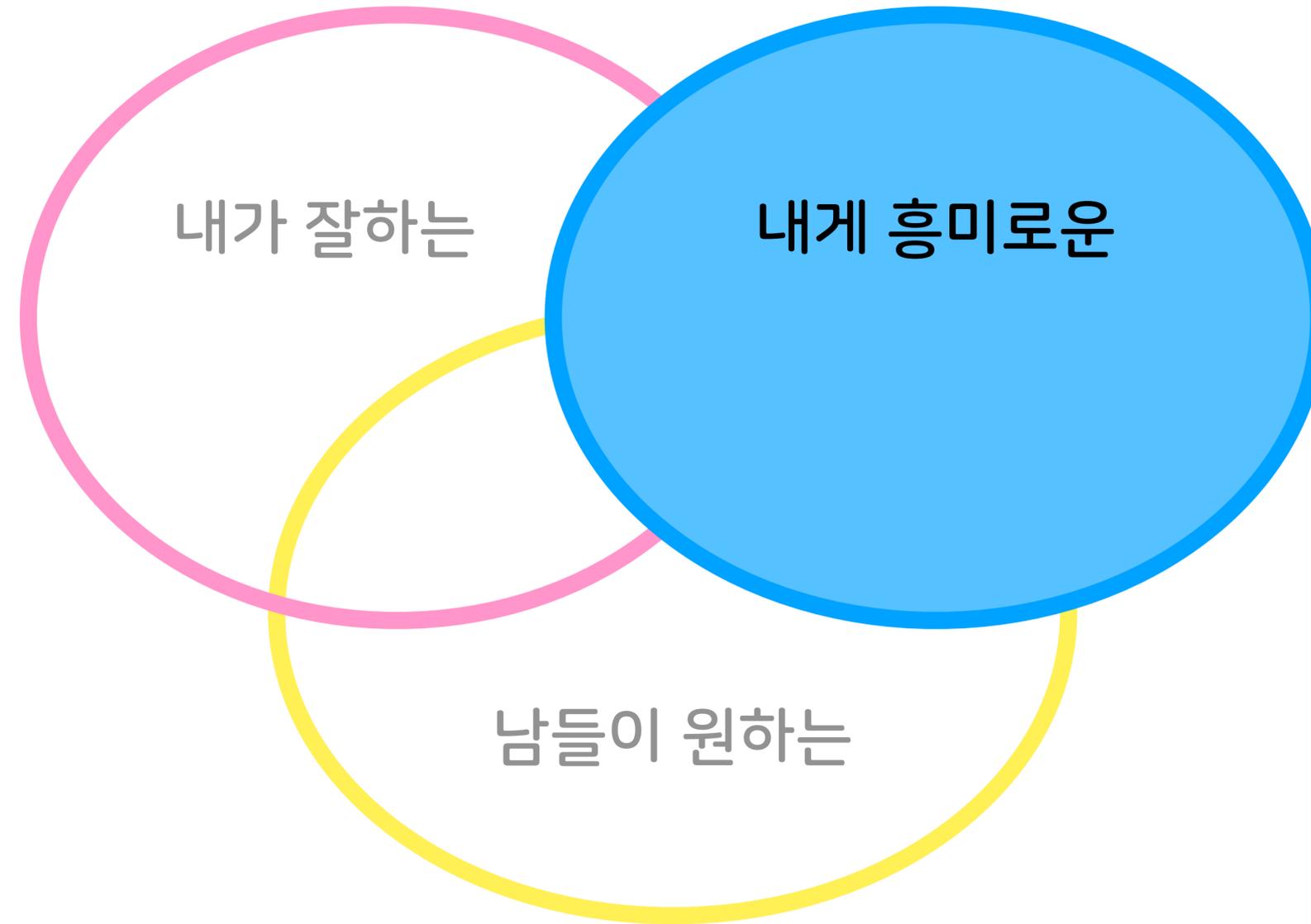
데이터 모델은 핵심을 잘 추려내야 구현이 편하다
용어나 표기법을 정리해 두면 추상화에 도움이 된다
역할을 잘 분리하고 풀고자하는 문제 범위를 줄이자
"열심히" 관찰하면, 더 나은 관점이나 풀이법이 떠오른다
관찰에 충분한 시간과 정성이 필요
계속 관찰만 할 수는 없기에, good-enough 타협

4. 토이 프로젝트

잠깐, 이상적 프로젝트는?



토이 프로젝트는, 내게 흥미로우면 충분



토이 프로젝트 장점 | Pros

개발 **흥미** 유발 | 유지 | 회복

새로운 (언어 프레임워크 라이브러리 등) **학습** 기회

훌륭한 결과물이어야 한다는 부담이 적다

수익에 대한 고민도 필요 없다

글쓰기 주제, 발표 주제로 유용

토이 프로젝트 단점 | Cons

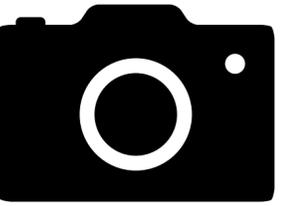
남들에게는 시시할 수 있다

나만의 흥미이기 쉽기 때문에, 혼자 다 만들어야...

흥미가 식으면, 프로젝트도 흐지부지되기 쉽다

결과물 수익화 가능성 낮다 → 왜 하는지 회의감 들 때가 있다

토이 프로젝트 진행 전략



내 **재미와 학습**이라는 목표에 집중 → 남들이 뭐라건 My Way

목표 결과물을 **작고 간단하게** 설정

흥미가 식기 전, **빠르게** 진행해 마무리

수익이 없어도 괜찮다 마인드 | 보통 재밌는 거 하려면 돈 써야...

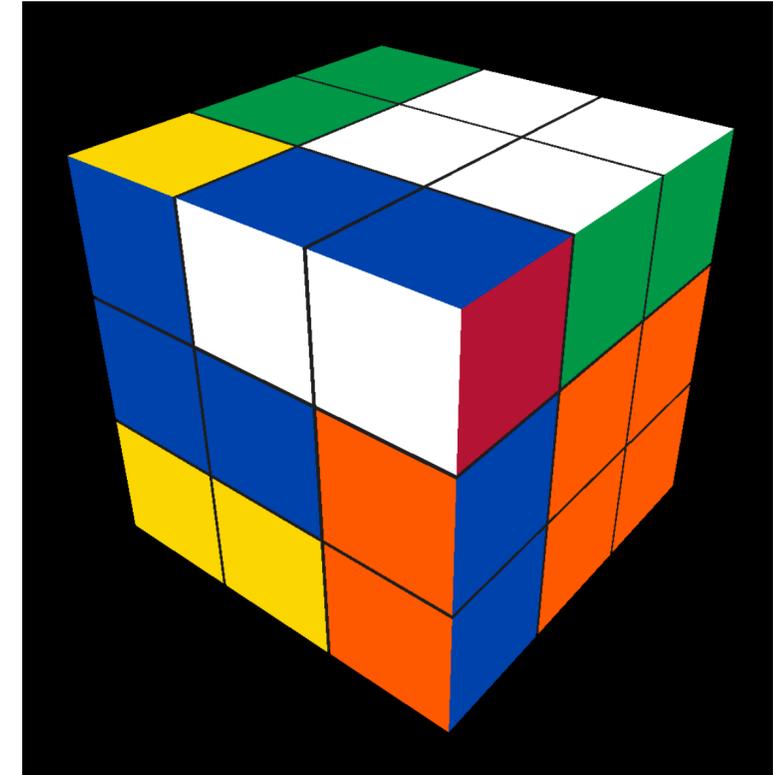
마무리

다시, 프로젝트 데모

<https://hatemogi.github.io/rurucube>

큐브가 보이면, R, U, L, F 키 등을 눌러보세요

ESC를 누르면 최초 상태로 복귀



앞으로 과제

중급 패턴 (F2L) 풀이 기능 추가

온라인 (유료) 강의로 제작 중

최신 알고리즘 적용 풀이

뒤늦은 자기소개 | 김대현

20+년 경력 백엔드 개발자, 개발팀장

(현) 컨스택츠 백엔드 엔지니어 | Haskell

LINE+ 광고 수익 적립 배분 시스템 | Scala

NHN Dooray 메신저 | Java, Kotlin

Daum 카페, 캘린더, 사내 깃헙, 클라우드 | Java, Ruby



<https://medium.com/@hatemogi>

<https://www.youtube.com/c/hatemogi101>

요약 정리

동적 타입 vs. 정적 타입 | 당신의 경험과 선택은?

Canvas에 WebGL 3D 그래픽 다루는 three.js 훌륭하더라

토이 프로젝트 | 흥미, 학습, 포트폴리오, 블로그

개발 프로젝트 | 마음껏 상상하고, 본질을 선별해, 만들어 내자

나만의
오늘부터 토이 프로젝트 시작!

여담들

그럴리 없겠지만, 시간이 남는다면...

큐브 풀이 경우의 수

막 돌리다 보면 풀리지 않을까 기대할 수도 있지만

큐브 패턴 경우의 수는,

$$8! \times 3^7 \times \frac{12!}{2} \times 2^{11} = 43,252,003,274,489,856,000$$

무작위로 불가능. 컴퓨터로도 쉽지 않은 수준.

유튜브의 힘으로 풀 수 있게 됐다



How to Solve a 3x3 Rubik's Cube In No Time | The Easiest Tutorial

조회수 4544만회 · 3년 전

BRIGHT SIDE ✓

If you look up the word "frustration" in the dictionary, you'll probably see a picture of a Rubik's Cube. It takes some bright minds ...

자막



Learn How to Solve a Rubik's Cube in 10 Minutes (Beginner Tutorial)

조회수 3368만회 · 3년 전

J Perm ✓

Learning to solve a Rubik's Cube can be easy! Read the pinned comment for common questions...

자막

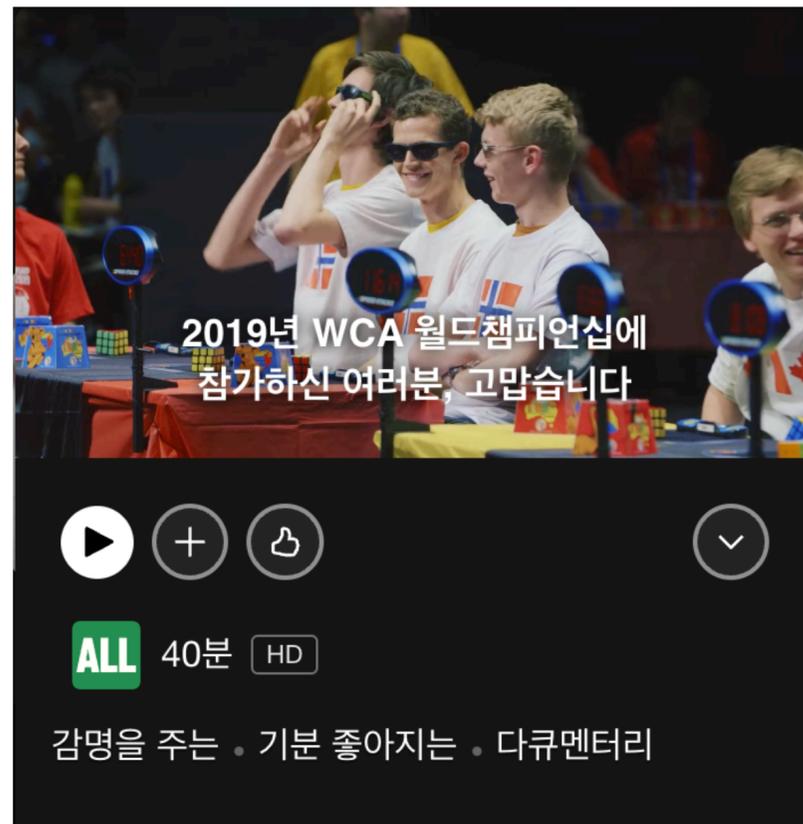
White Cross | White Corners | Second Layer | Top Cross | Match Cros... **챕터 7** ▾

세계 스피드 큐브 대회 기록

세계 기록은 2022년 4월 5회 평균 5.08초 Max Park

내년에 서울에서 세계대회 열린다고...

넷플릭스에 "스피드 큐브의 천재들" 다큐멘터리도 있음



Championship	Year	Host	Date(s)	Nations	Puzzles	Events	Winner (3x3)	Winning time(s)	Ref
I	1982	Budapest	5 June	19	1	1	Minh Thai	22.95 (Single)	[15]
II	2003	Toronto	23-24 August	15	9	13	Dan Knights	20.00 (Average)	[16]
III	2005	Lake Buena Vista	5-6 November	16	9	15	Jean Pons	15.10 (Average)	[17]
IV	2007	Budapest	5-7 October	28	10	17	Yu Nakajima	12.46 (Average)	[18]
V	2009	Düsseldorf	9-11 October	32	12	19	Breandan Vallance	10.74 (Average)	[19]
VI	2011	Bangkok	14-16 October	35	12	19	Michał Pleskowicz	8.65 (Average)	[20]
VII	2013	Las Vegas	26-28 July	35	10	17	Feliks Zemdegs	8.18 (Average)	[21]
VIII	2015	São Paulo	17-19 July	37	11	18	Feliks Zemdegs	7.56 (Average)	[22]
IX	2017	Paris	13-16 July	64	11	18	Max Park	6.85 (Average)	[23]
X	2019	Melbourne	11-14 July	52	11	18	Philipp Weyer	6.74 (Average)	[14]
XI	2021	Almere	Cancelled ^[1]	—	11	17	—	—	[24]
XII	2023	Seoul	12-15 August	—	11	17	—	—	[25]

Journal of Physics A: Mathematical and Theoretical

PAPER

Solving Rubik's cube via quantum mechanics and deep reinforcement learning

Sebastiano Corli^{1,2}, Lorenzo Moro^{1,3}, Davide E Galli² and Enrico Prati^{4,1} 

Published 29 September 2021 • © 2021 IOP Publishing Ltd

[Journal of Physics A: Mathematical and Theoretical](#), [Volume 54](#), [Number 42](#)**Citation** Sebastiano Corli et al 2021 *J. Phys. A: Math. Theor.* **54** 425302[References ▾](#)[+ Article information](#)

Abstract

Rubik's cube is one of the most famous combinatorial puzzles involving nearly 4.3×10^{19} possible configurations. Its mathematical description is expressed by the Rubik's group, whose elements define how its layers rotate. We develop a unitary representation of such group and a quantum formalism to describe the cube from its geometrical constraints. Cubies are described by single particle states which turn out to behave like bosons for corners and fermions for edges, respectively. When in its solved configuration, the cube, as a geometrical object, shows symmetries which are broken when driven away from this configuration. For each of such symmetries, we build a Hamiltonian operator. When a Hamiltonian lies in its ground state, the respective symmetry of the

335 Total downloads

[Turn on MathJax](#)[Get permission to re-use this article](#)

Share this article

[Abstract](#)[References](#)

나만의
오늘부터 토이 프로젝트 시작!