

알튜비튜

동적 계획법

오늘은 '다이나믹 프로그래밍'이라고도 불리는 동적 계획법 알고리즘에 대해 배웁니다.
과거에 구한 해를 현재 해를 구할 때 활용하는 알고리즘이죠. 문제에 많이 나오는 굉장히 중요한 알고리즘 중 하나예요.

동적 계획법

- 특정 범위까지의 값을 구하기 위해 이전 범위의 값을 활용하여 효율적으로 값을 얻는 기법
- 이전 범위의 값을 저장(Memoization)함으로써 시간적, 공간적 효율 얻음

/<> 2240번 : 자두나무 - Gold 5

문제

- 매 초마다, 두 개의 나무 중 하나의 나무에서 열매가 떨어짐
- 매 초마다 어느 나무에서 자두가 떨어질지에 대한 정보가 주어졌을 때, 자두가 받을 수 있는 최대 자두의 개수 출력

제한 사항

- 초 $T(1 \leq T \leq 1,000)$
- 움직일 수 있는 횟수 $W(1 \leq W \leq 30)$

풀이 방법

- 1. 저장할 요소들에 대해 파악 : 매 초마다의 상황, 현재 몇 번 움직였는지
- 2. 2개의 나무가 있음 → 인덱스로 처리 가능(이동횟수가 홀수이면 무조건 2번 나무 아래, 짝수이면 무조건 1번 나무 아래에 있기 때문)
- 3. 가능한 경우에 대해 생각

- i는 초에 대한 것, j는 이동 횟수에 대한 것

```
for (int i = 1; i <= t; i++)
{
    if (li[i] == 1)
        dp[i][0] = dp[i - 1][0] + 1;
    else
        dp[i][0] = dp[i - 1][0];

    for (int j = 1; j <= w; j++)
    {
        if (li[i] == 2 && j % 2 == 1)
            dp[i][j] = max(dp[i - 1][j - 1], dp[i - 1][j]) + 1;
        else if (li[i] == 1 && j % 2 == 0)
            dp[i][j] = max(dp[i - 1][j - 1], dp[i - 1][j]) + 1;
        else
            dp[i][j] = max(dp[i - 1][j - 1], dp[i - 1][j]);
    }
}
```

- i초에 자두가 2번 나무에서 떨어지고, 현재 2번 나무 일 때 (이동 횟수가 홀수면 2번 나무)

```
if (li[i] == 2 && j % 2 == 1)
    dp[i][j] = max(dp[i - 1][j - 1], dp[i - 1][j]) + 1;
```

- i초에 자두가 1번 나무에서 떨어지고, 현재 1번 나무 일 때 (이동 횟수가 짝수면 1번 나무)

```
else if (li[i] == 1 && j % 2 == 0)
    dp[i][j] = max(dp[i - 1][j - 1], dp[i - 1][j]) + 1;
```

- 현재 서있는 곳에서 자두를 받을 수 없는 경우

```
else  
    dp[i][j] = max(dp[i - 1][j - 1], dp[i - 1][j]);
```

/<> 12865번 : 평범한 배낭 - Gold 5

문제

- 최대 무게(k)가 정해진 배낭에 물건을 넣는다.
- 각 물건은 무게(w)와 가치(v)가 있다.
- 배낭에 넣을 수 있는 물건들의 가치합의 최댓값을 구하는 문제

제한 사항

- 물품의 수 N ($1 \leq N \leq 100$)
- 배낭 무게 K ($1 \leq K \leq 100,000$)
- 물건 무게 W ($1 \leq W \leq 100,000$)
- 물건 가치 V ($0 \leq V \leq 1,000$)

예제 입력

```
4 7
6 13
4 8
3 6
5 12
```

예제 출력

```
14
```

Hint

1. 전 시간들에 배운 알고리즘으로 풀기엔 시간이 부족해보여요.
2. 부분 문제에 대한 정답을 어떻게 활용할 수 있을까요? 이 문제의 전체 정답은 최대 무게 K 일 때의 최대 가치합이죠. 그렇다면 부분 문제는 무엇일까요?

문제

- 최대 무게(k)가 정해진 배낭에 물건을 넣는다.
- 각 물건은 무게(w)와 가치(v)가 있다.
- 배낭에 넣을 수 있는 물건들의 가치합의 최댓값을 구하는 문제

접근

- 물품의 가능한 조합을 모두 구한 후, 무게가 K 이내이면서 가치합이 최대인 경우를 찾는 브루트 포스 접근
- $O(2^n)$ 이고, 물품의 수(n)가 최대 100이므로 절대 불가능!

문제

- 최대 무게(k)가 정해진 배낭에 물건을 넣는다.
- 각 물건은 무게(w)와 가치(v)가 있다.
- 배낭에 넣을 수 있는 물건들의 가치합의 최댓값을 구하는 문제

접근

- 물품의 가능한 조합을 구하는데, 중간에 무게가 K 를 초과하는 경우를 모두 쳐내며 가치합이 최대인 경우를 찾는 백트래킹 접근
 - à 웬지 가능해 보이지만, 이 풀이도 최악의 경우 K 를 초과하는 경우가 없으면 결국 브루트 포스와 동일. 즉, 불가능

문제

- 최대 무게(k)가 정해진 배낭에 물건을 넣는다.
- 각 물건은 무게(w)와 가치(v)가 있다.
- 배낭에 넣을 수 있는 물건들의 가치합의 최댓값을 구하는 문제

접근

- 오늘 배운 동적 계획법을 활용해 보자. 이전에 구한 답을 활용?
 - à K 이전의 무게들에 대한 정답(가치합의 최댓값)을 저장하며 풀면 어떨까?
 - à 무게를 인덱스로!

K이전 무게들에 대한 정답은 어떻게 계산?


접근

- K 까지의 무게를 인덱스로 나타냄
- 현재 물품을 배낭에 넣는 경우 or 안 넣는 경우 중 최대값을 저장하자
- 배낭에 넣으려면?
 - 현재 물품 무게만큼 배낭에 추가되는 것! 그런데 현재 인덱스가 배낭의 최대 무게인데?
 - $[\text{현재 배낭 무게} - \text{물품 무게}]$ 인 배낭 무게에서의 최대 가치값 + 현재 물품 가치값
- 배낭에 안 넣는 경우는?
 - à 현재 배낭 무게에 저장된 정답을 그대로 사용하면 됨!

점화식을 세워봅시다

- $n = 4$
- $k = 7$
- product(물품) $w = 6, v = 13$
- 현재 물품을 배낭에 넣는 경우 or 안 넣는 경우 중 최댓값을 저장하자

	0	1	2	3	4	5	6	7
dp[1]	0	0	0	0	0	0	13	13



- 배낭에 넣는 경우: (현재 배낭 무게 - 물품 무게)를 인덱스로 가지는 값 + 물품 가치 $\rightarrow dp[0][0] + 13$
- 배낭에 안 넣는 경우: 0

점화식을 세워봅시다

- $n = 4$
- $k = 7$
- product(물품) $w = 4, v = 8$
- 현재 물품을 배낭에 넣는 경우 or 안 넣는 경우 중 최댓값을 저장하자

	0	1	2	3	4	5	6	7
dp[1]	0	0	0	0	0	0	13	13
dp[2]	0	0	0	0	8	8	13	13




● 배낭에 넣는 경우: $dp[1][6 - 4] + 8 = 8$

● 배낭에 안 넣는 경우: $dp[1][6] = 13$

점화식을 세워봅시다

- $n = 4$
- $k = 7$
- product(물품) $w = 3, v = 6$
- 현재 물품을 배낭에 넣는 경우 or 안 넣는 경우 중 최댓값을 저장하자

	0	1	2	3	4	5	6	7
dp[1]	0	0	0	0	0	0	13	13
dp[2]	0	0	0	0	8	8	13	13
dp[3]	0	0	0	6	8	8	13	14



- 배낭에 넣는 경우: $dp[2][7 - 3] + 6 = 14$
- 배낭에 안 넣는 경우: $dp[2][7] = 13$

점화식을 세워봅시다

- $n = 4$
- $k = 7$
- product(물품) $w = 5, v = 12$
- 현재 물품을 배낭에 넣는 경우 or 안 넣는 경우 중 최댓값을 저장하자

	0	1	2	3	4	5	6	7
dp[1]	0	0	0	0	0	0	13	13
dp[2]	0	0	0	0	8	8	13	13
dp[3]	0	0	0	6	8	8	13	14
dp[4]	0	0	0	6	8	12	13	14

점화식

- 현재 물품을 배낭에 넣는 경우 or 안 넣는 경우 중 최댓값을 저장하자

$$\therefore DP[i][j] = \text{MAX}(DP[i-1][j-w[i]] + v[i], DP[i-1][j])$$

(단, $w[i] \leq j$)

왜 2차원 DP?

- 1차원 DP로 하면 안 되는 이유는?
à 그 전 물품까지의 정보만 사용해야 하기 때문

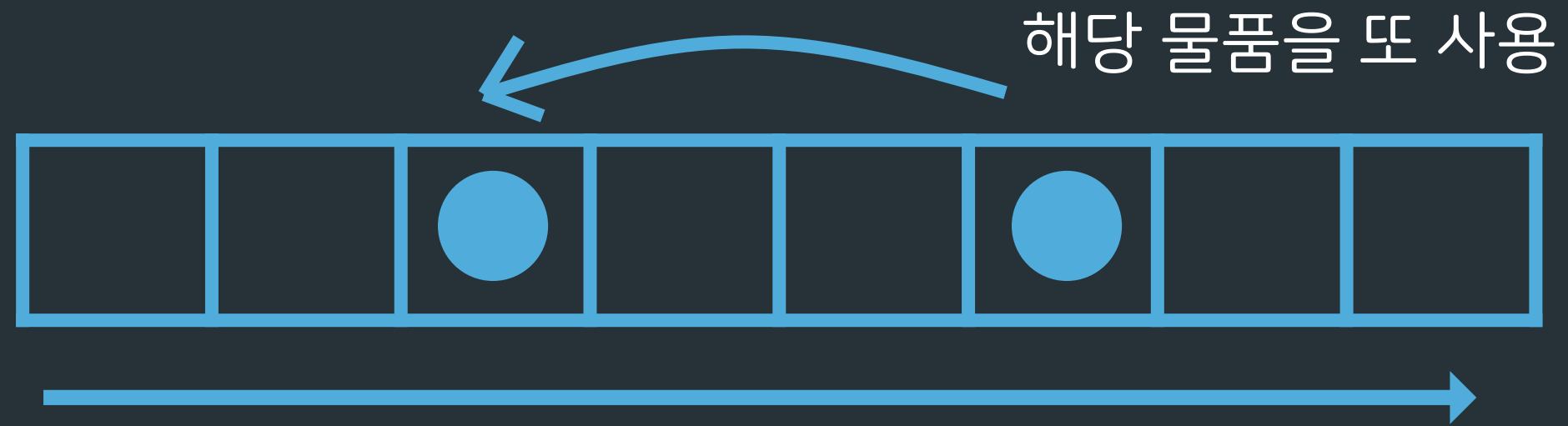
왜 2차원 DP?

- 1차원 DP로 하면 안 되는 이유는?
 - à 그 전 물품까지의 정보만 사용해야 하기 때문
- 지금처럼 증가하면서 검사할 때 1차원 DP를 사용하게 되면?
 - à 해당 물품을 또 사용하는 경우 생길 수 있음!
 - à 따라서 2차원을 사용하며 각 물품을 행으로 구분해서 중복 방지

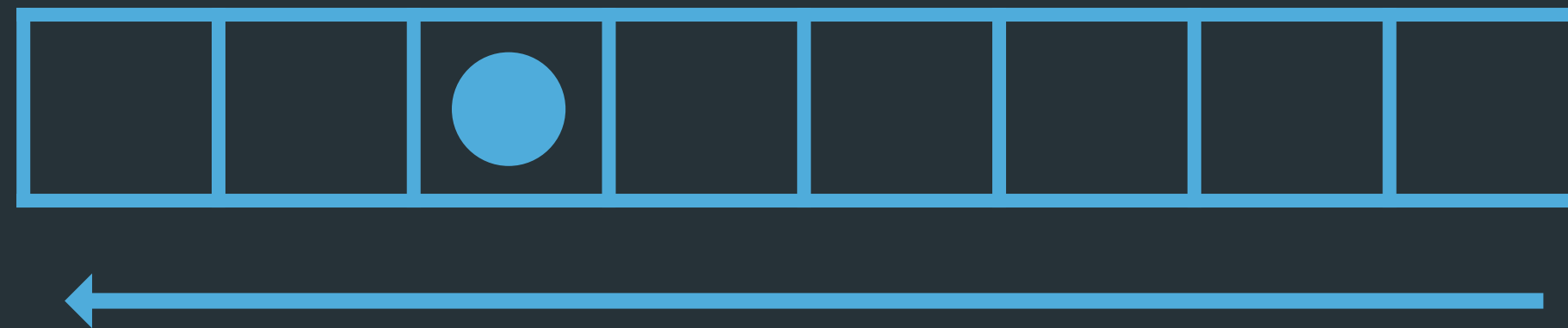
1차원 DP도 가능

- 어떻게 하면 1차원 DP로 풀이가 가능할까
- 해당 물품을 여러 번 사용하는 걸 방지하기 위해 2차원을 사용함
- 그렇다면.. 지금처럼 증가하는게 아니라 무게를 감소하며 계산하면 어떨까?

기존 증가 방식 dp



감소 방식 dp



뒤에서부터 채워지므로 중복 사용 방지

/<> 20923번 : 숫자 할리갈리 게임 - Silver 1

문제

1. 게임 시작 시 도도와 수연이는 N장의 카드로 구성된 덱을 받음 (그라운드는 비어 있음)
2. 도도와 수연이는 차례대로 덱의 가장 위에 있는 카드를 그라운드에 내려놓음
3. 종을 치는 사람이 그라운드의 카드 더미를 모두 가져감
 - a. 수연: 그라운드 위의 카드의 숫자 합이 5가 될 때 종을 침
 - b. 도도: 카드의 숫자가 5가 나올 때 종을 침
4. 카드 더미를 가져갈 때는 상대방의 그라운드, 자신의 그라운드 순으로 가져와 자신의 덱 아래에 합침
5. M번 게임을 진행한 후 더 많은 카드를 가진 사람이 승리, 카드의 수가 같은 경우 비김
 - a. 게임 도중 덱에 있는 카드가 다 떨어지면 상대가 승리
6. 2~4 과정을 진행하는 것을 한 번 진행한 것으로 볼 때 M번 진행 후 승리한 사람은?

/<> 20923번 : 숫자 할리갈리 게임 - Silver 1

제한 사항

- 도도와 수연이가 가지는 카드의 개수의 범위는 $1 \leq N \leq 30,000$
- 게임 진행 횟수의 범위는 $1 \leq M \leq 2,500,000$
- 각각의 카드는 1이상 5이하의 자연수가 적혀 있음

예제 입력1

```
10 12
1 2
2 2
1 2
2 3
3 1
2 2
2 5
2 1
5 1
2 3
```

예제 출력1

```
do
```

예제 입력2

```
1 1
5 2
```

예제 출력2

```
su
```

예제 입력3

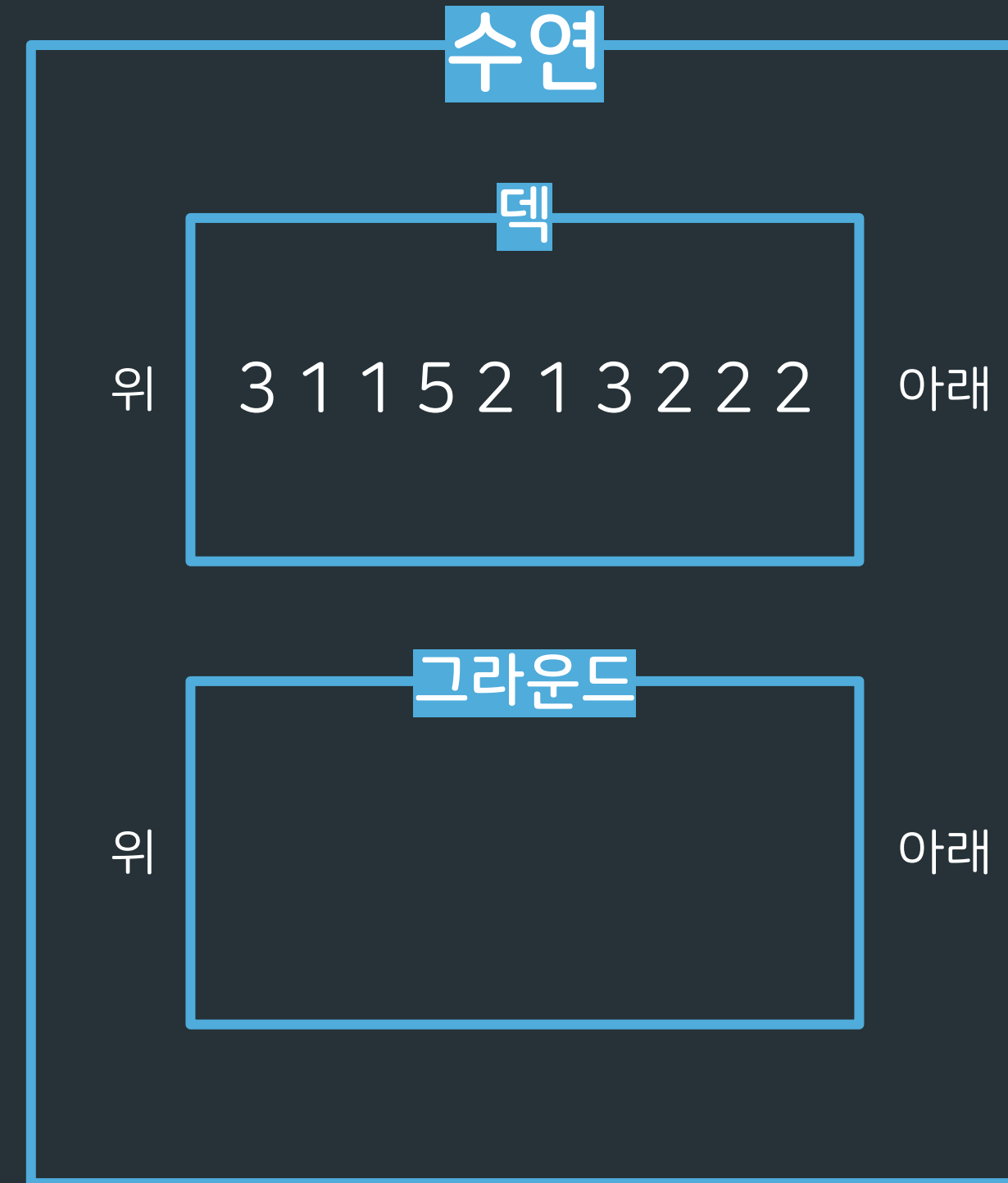
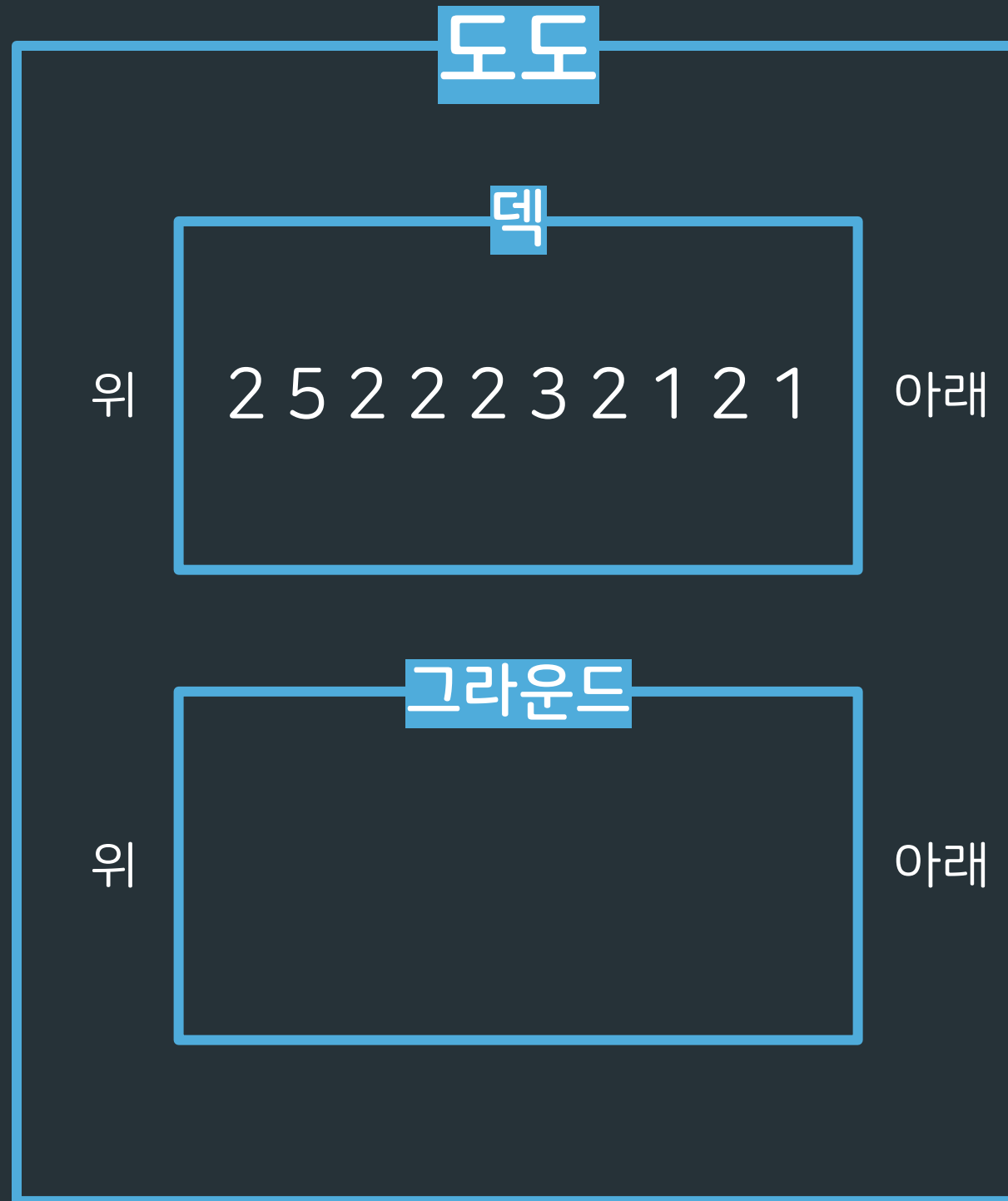
```
3 4
1 2
2 2
1 1
```

예제 출력3

```
dosu
```

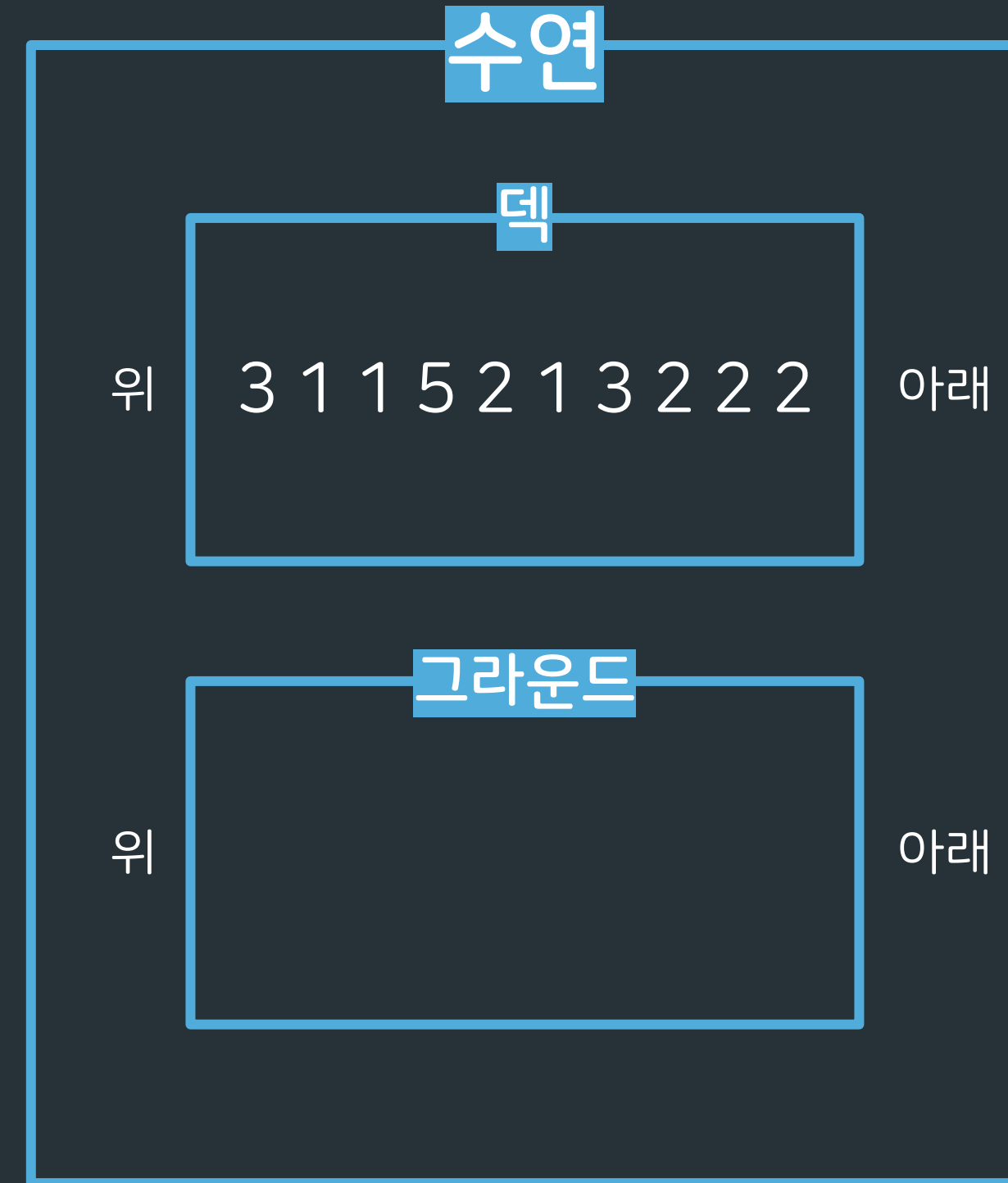
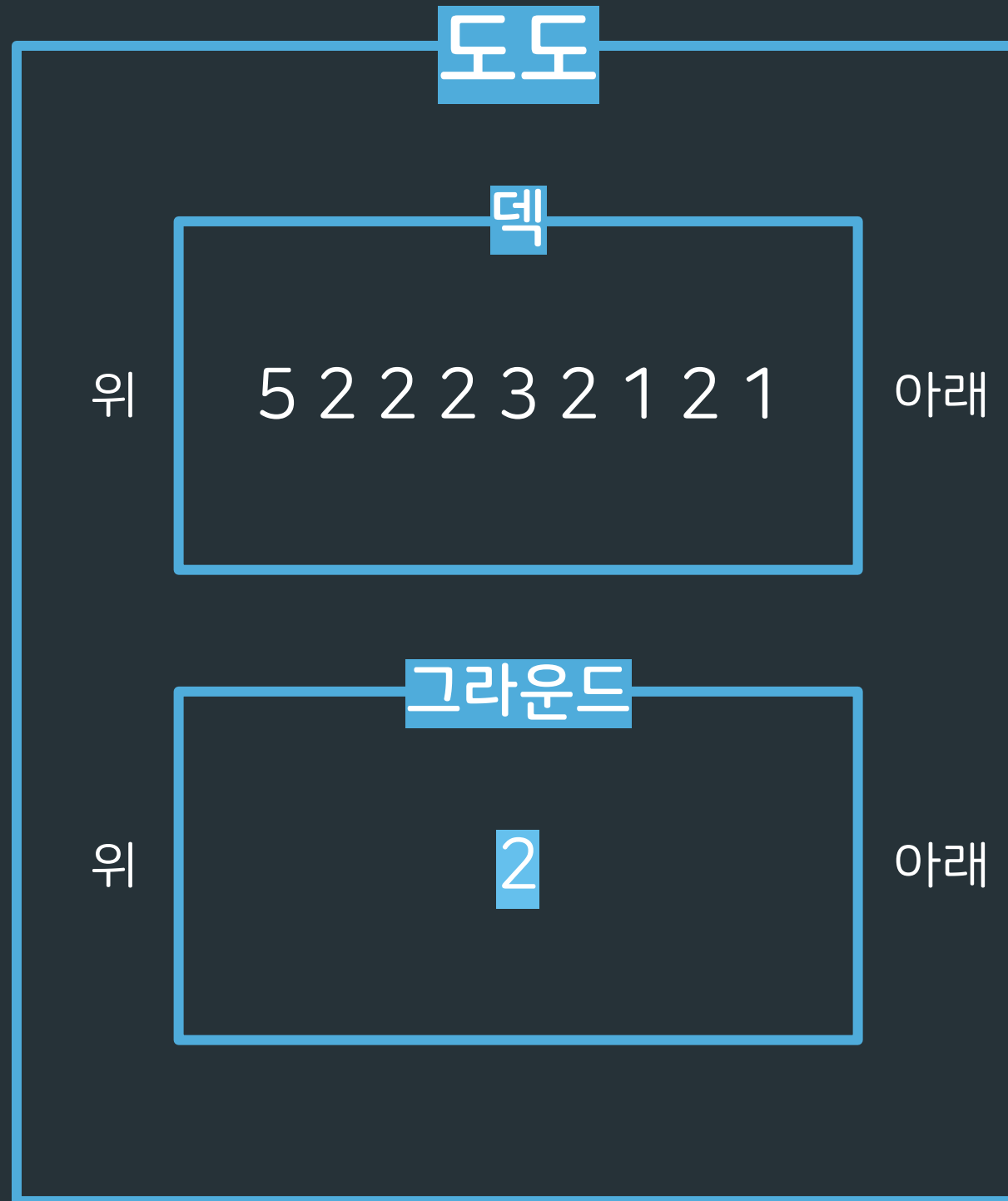
순서대로 게임을 진행해보자

M = 0



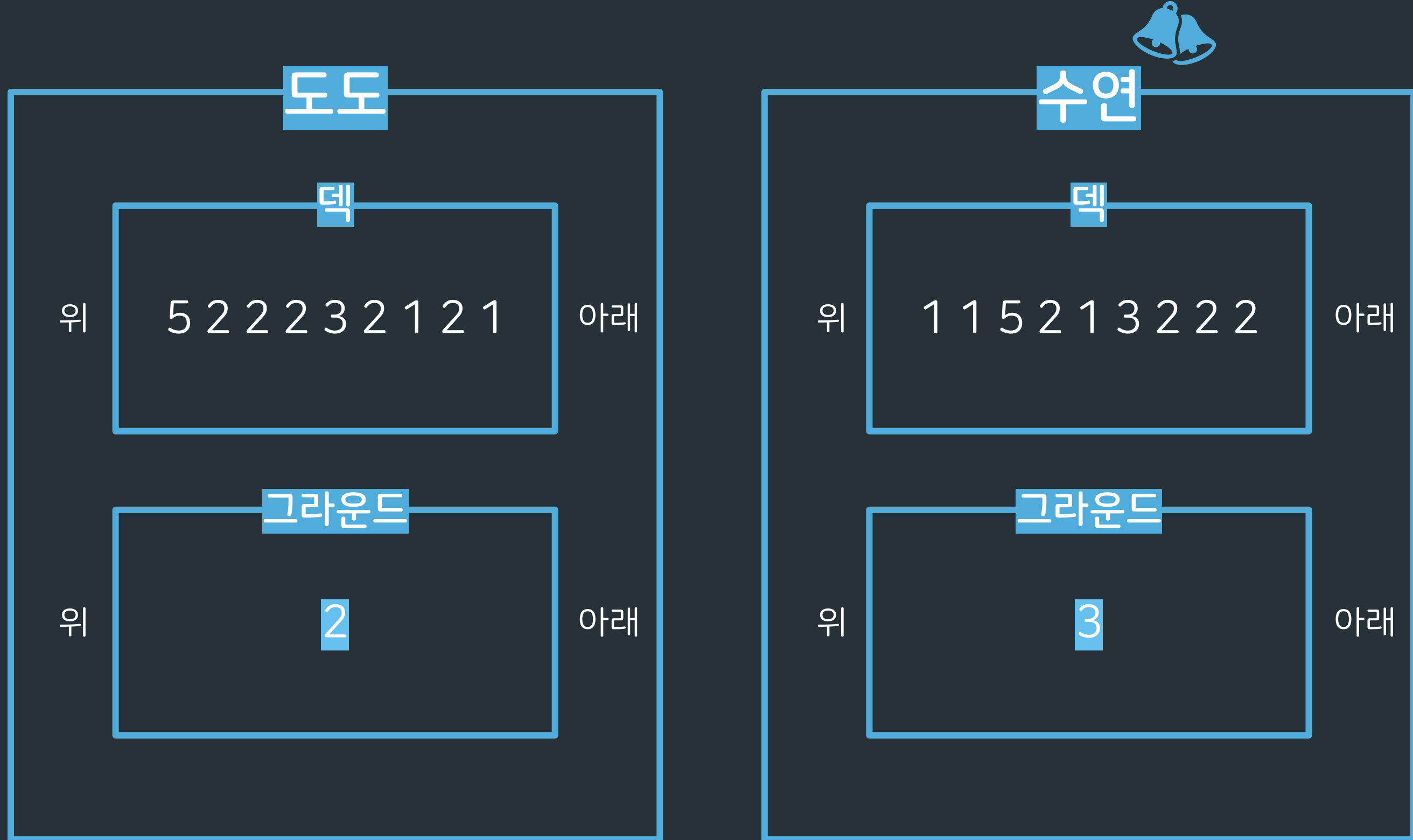
순서대로 게임을 진행해보자

M = 1



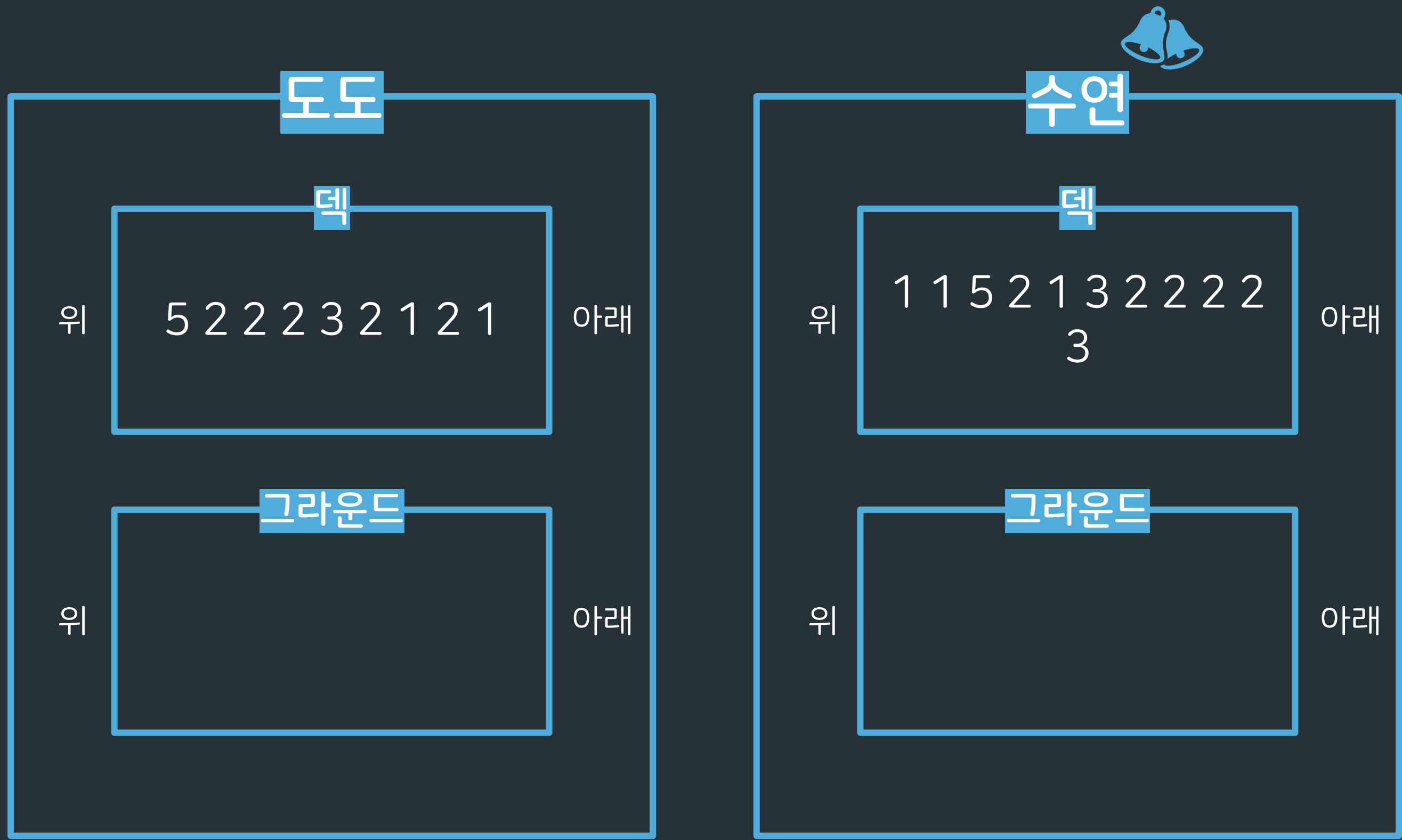
순서대로 게임을 진행해보자

M = 2



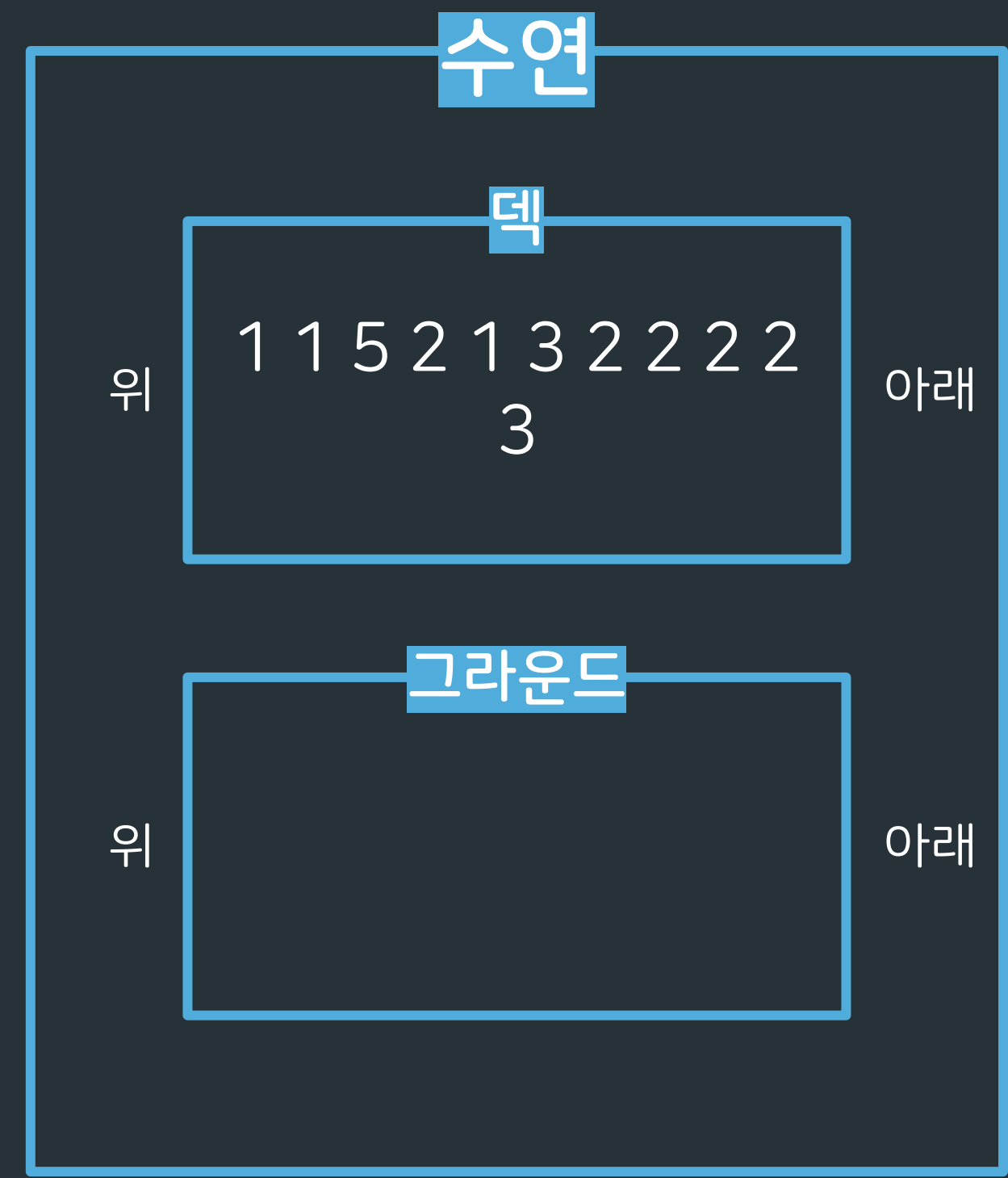
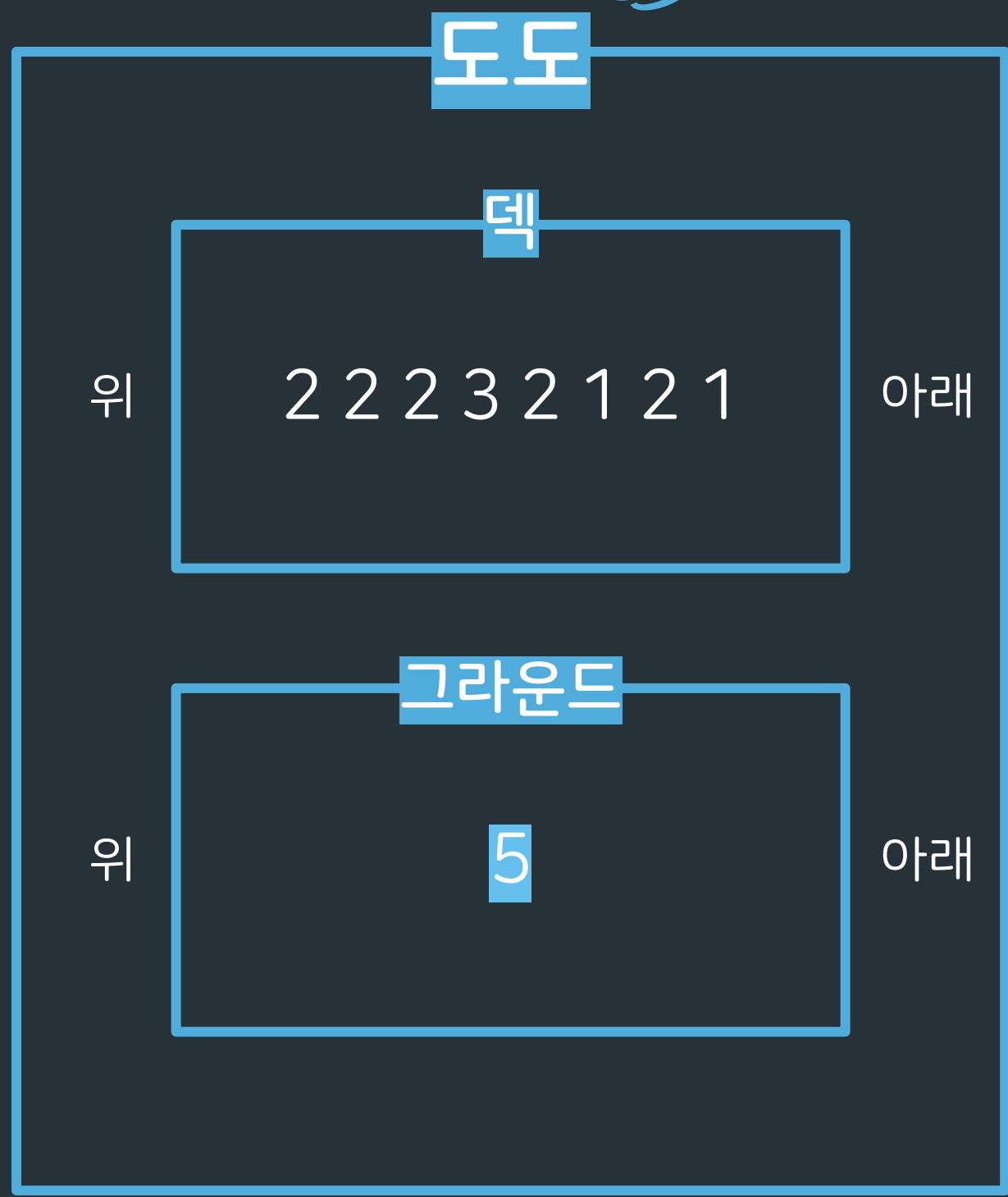
순서대로 게임을 진행해보자

M = 2



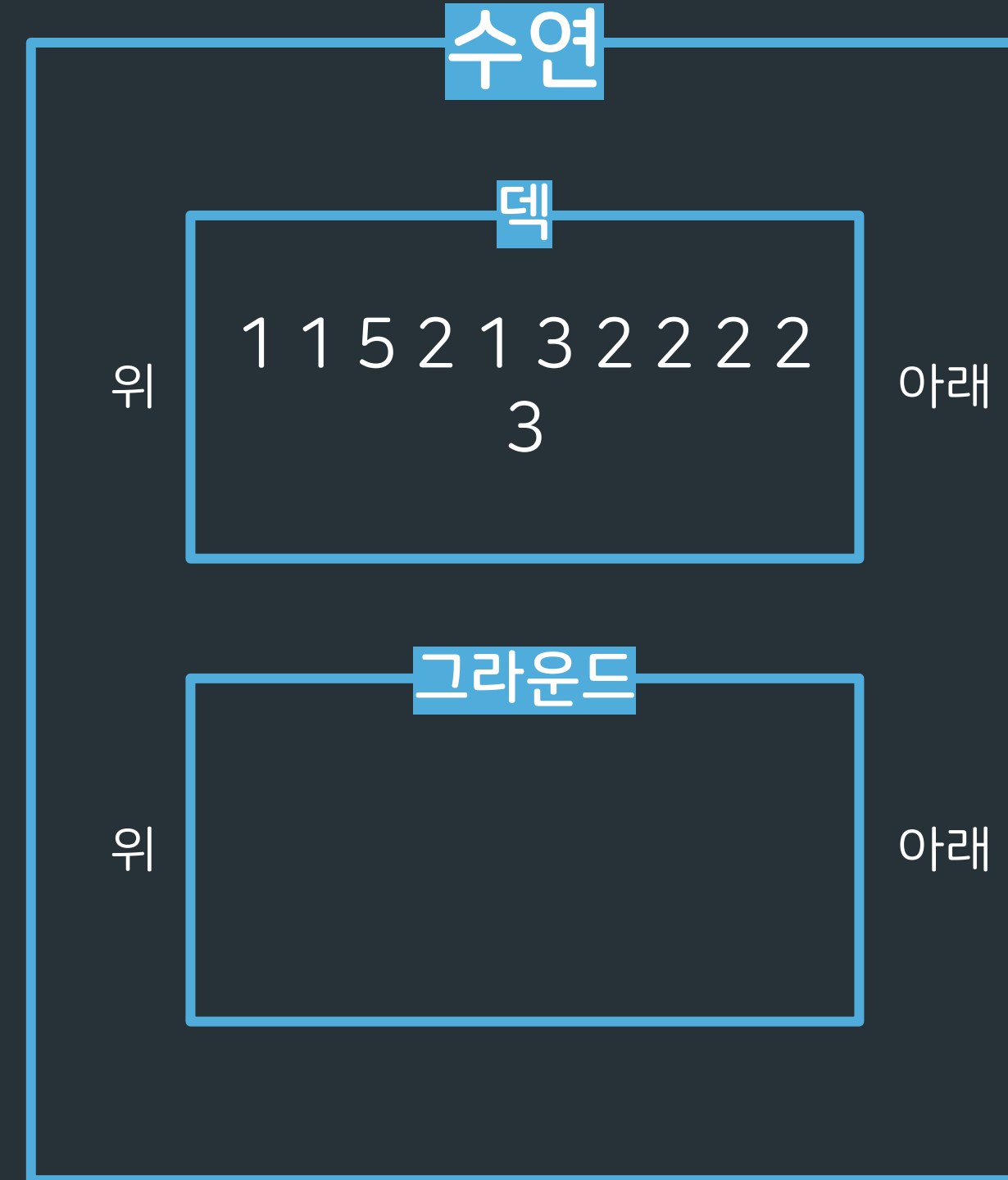
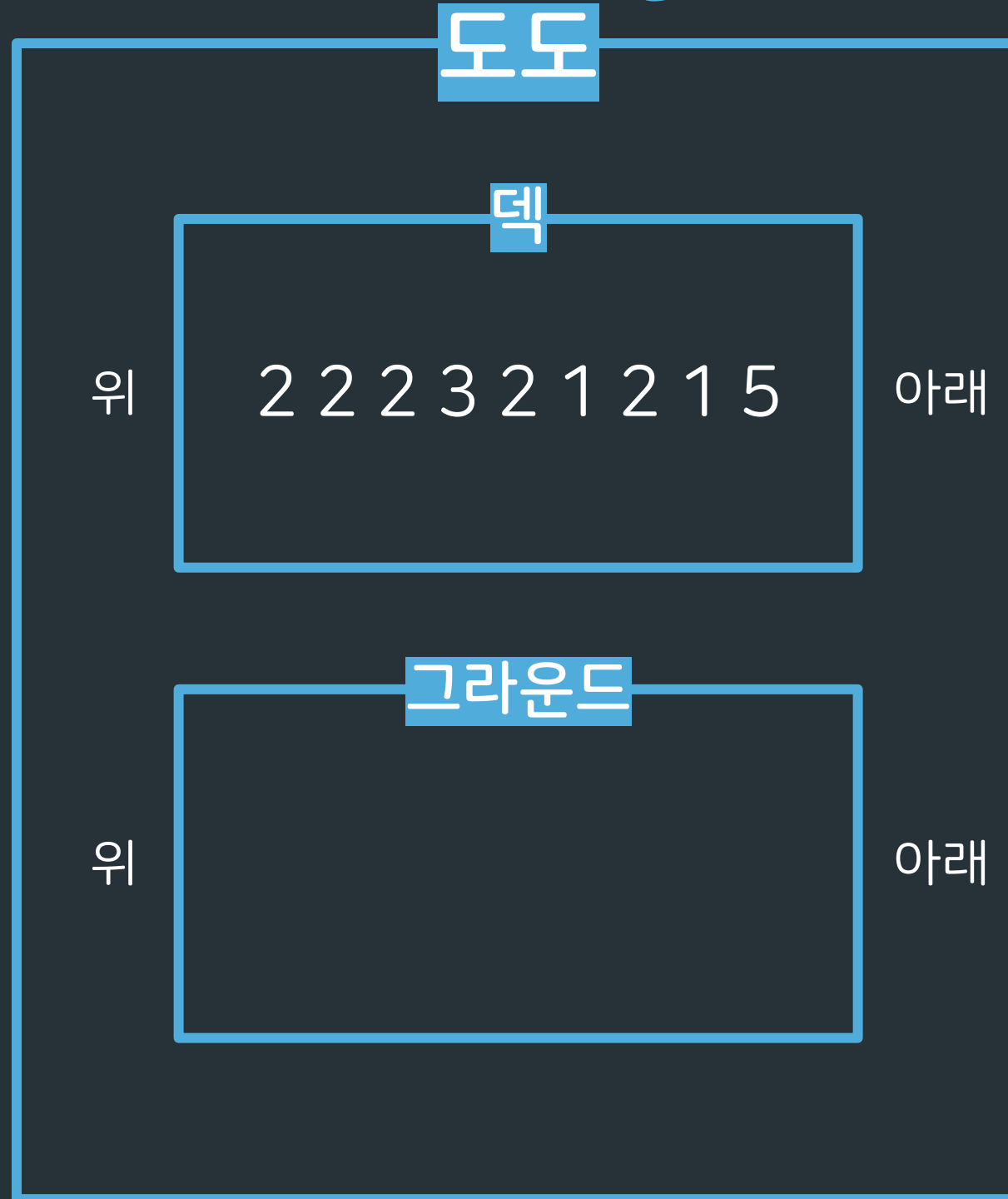
순서대로 게임을 진행해보자

M = 3



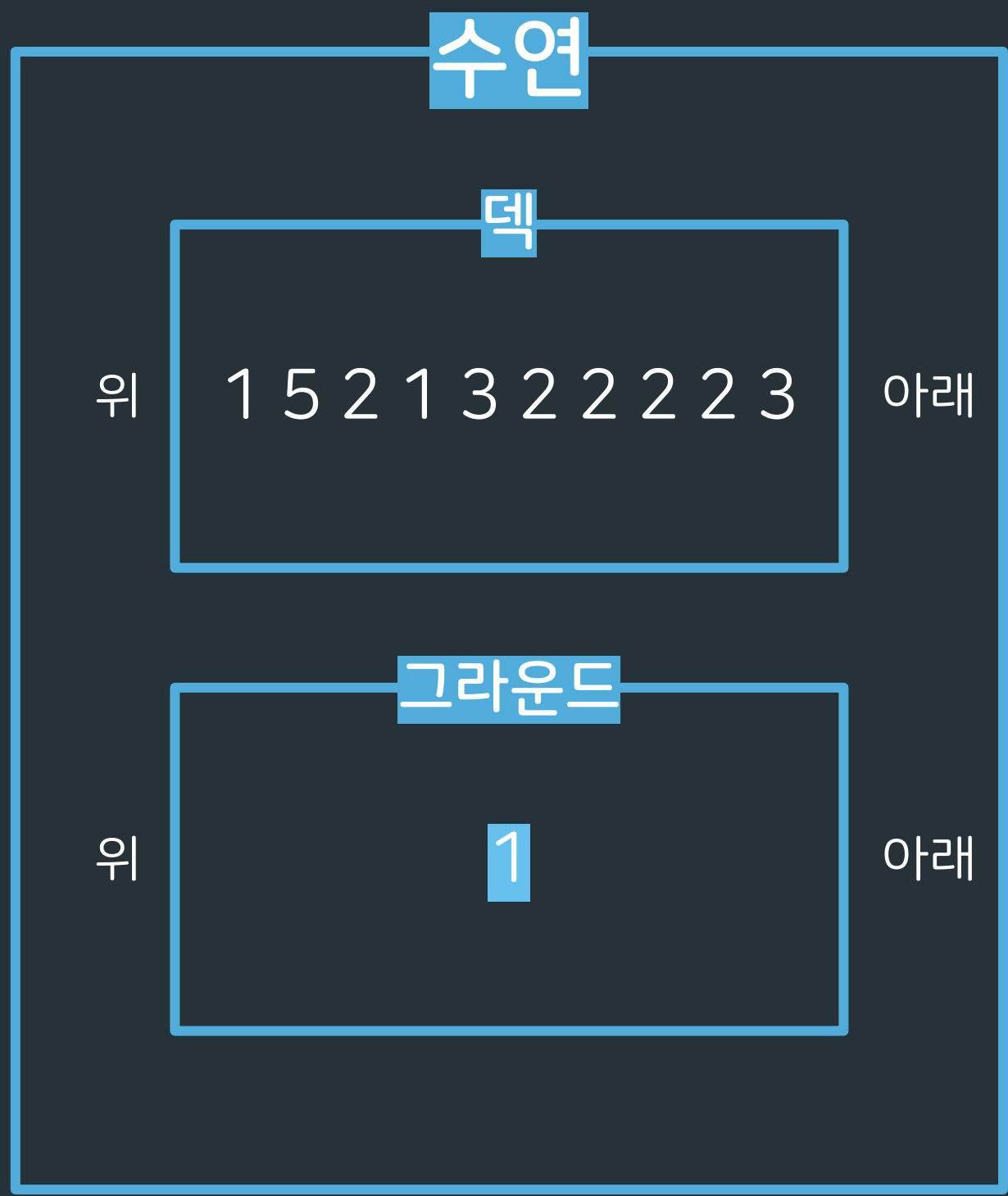
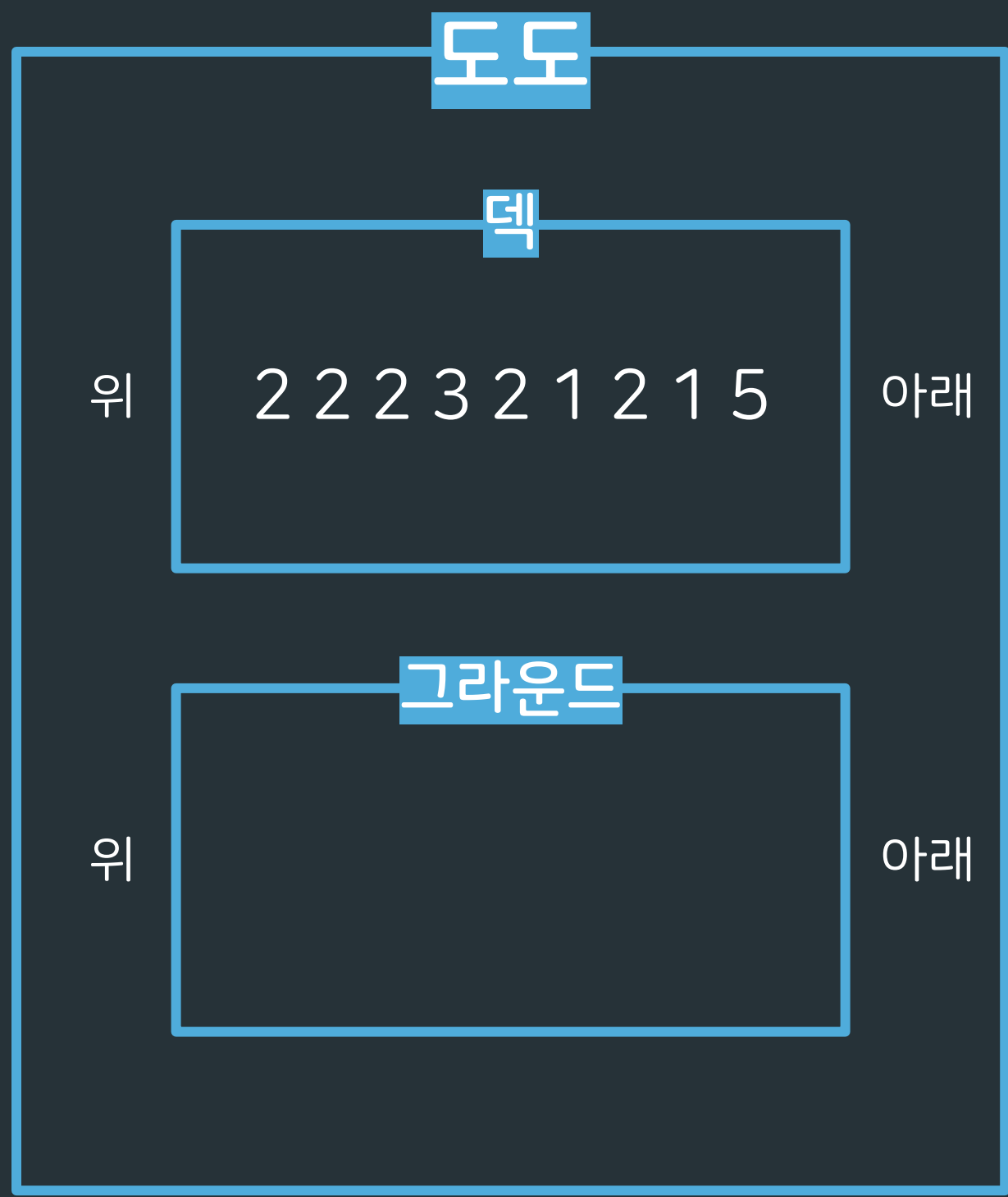
순서대로 게임을 진행해보자

M = 3



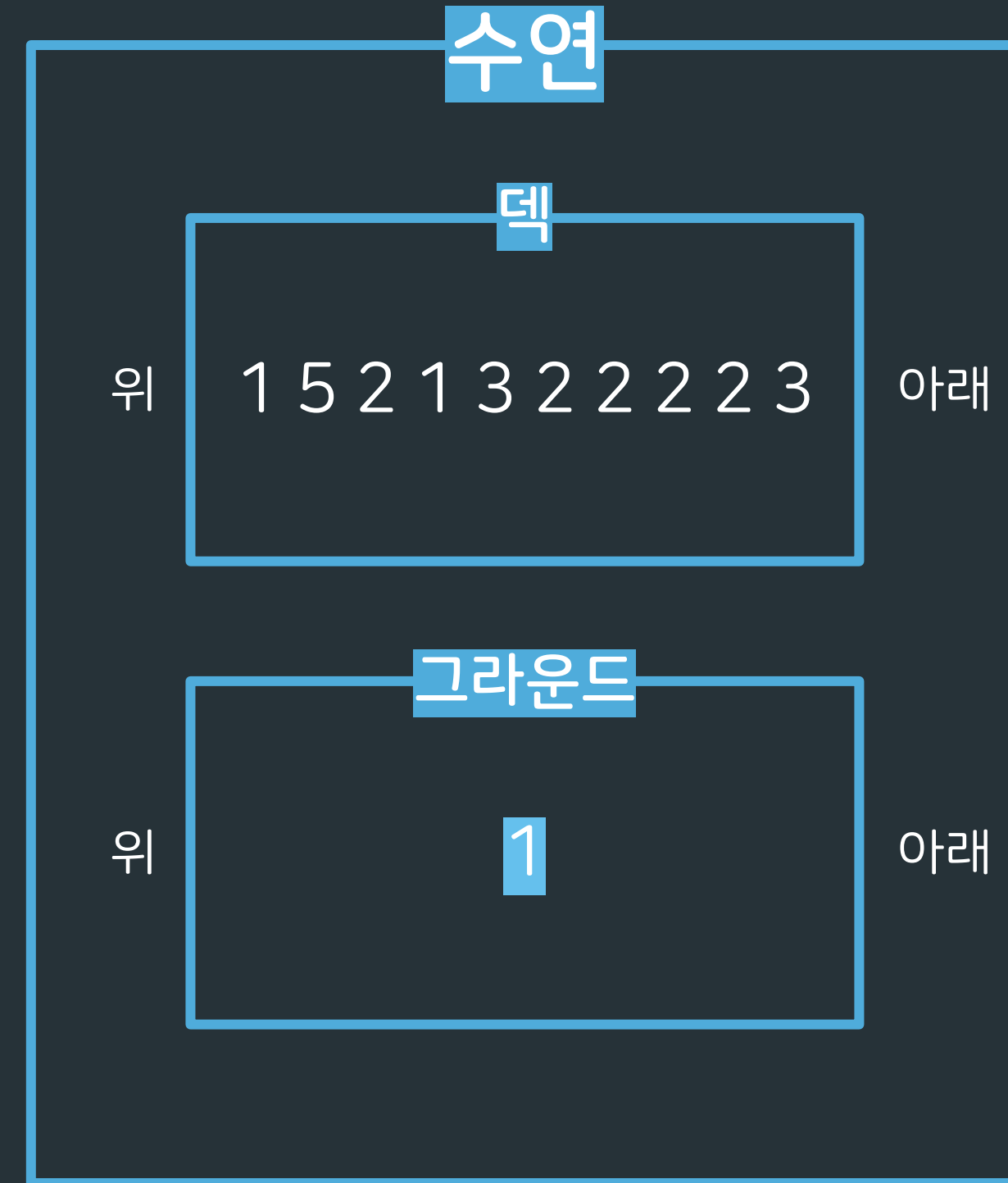
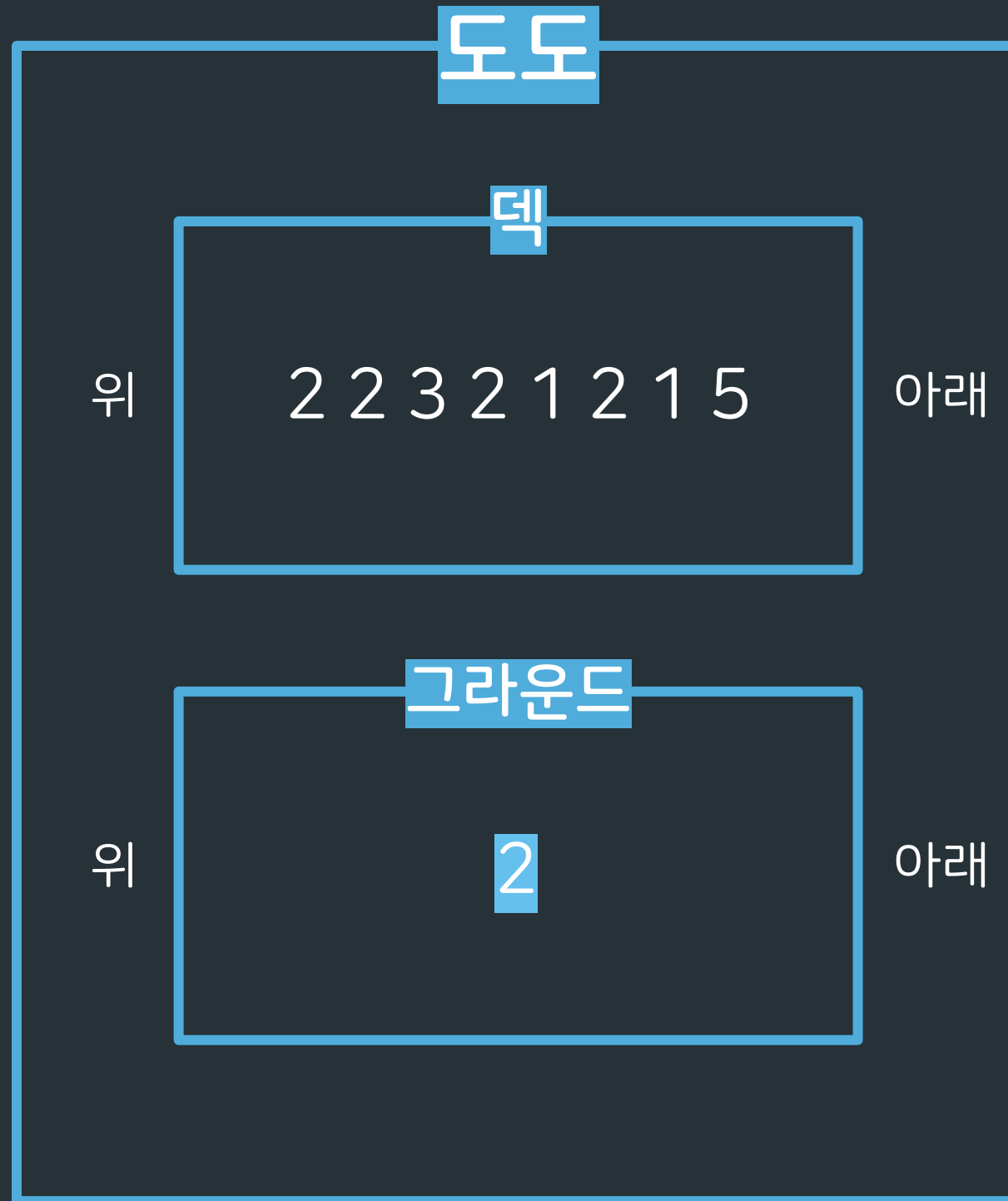
순서대로 게임을 진행해보자

M = 4



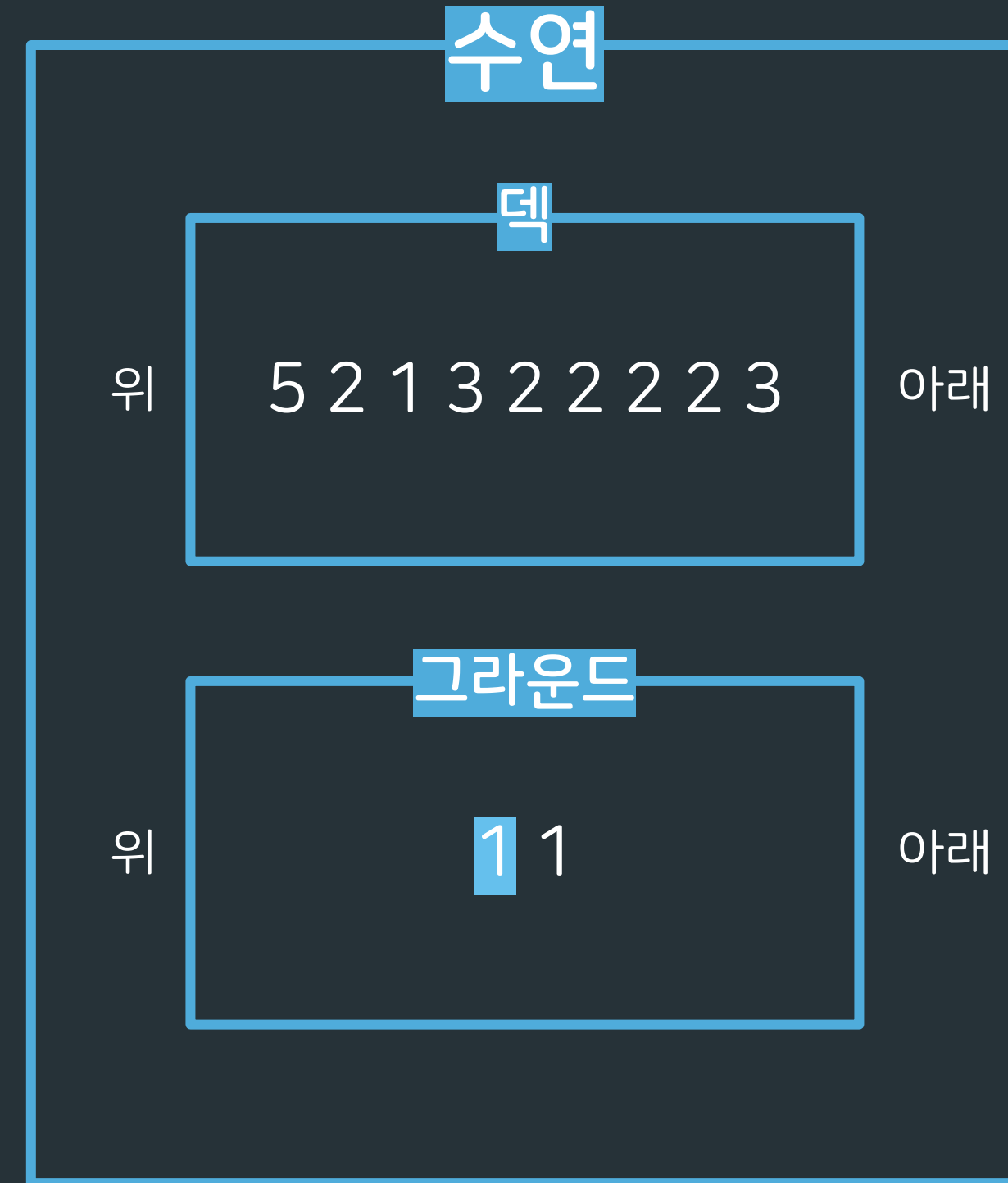
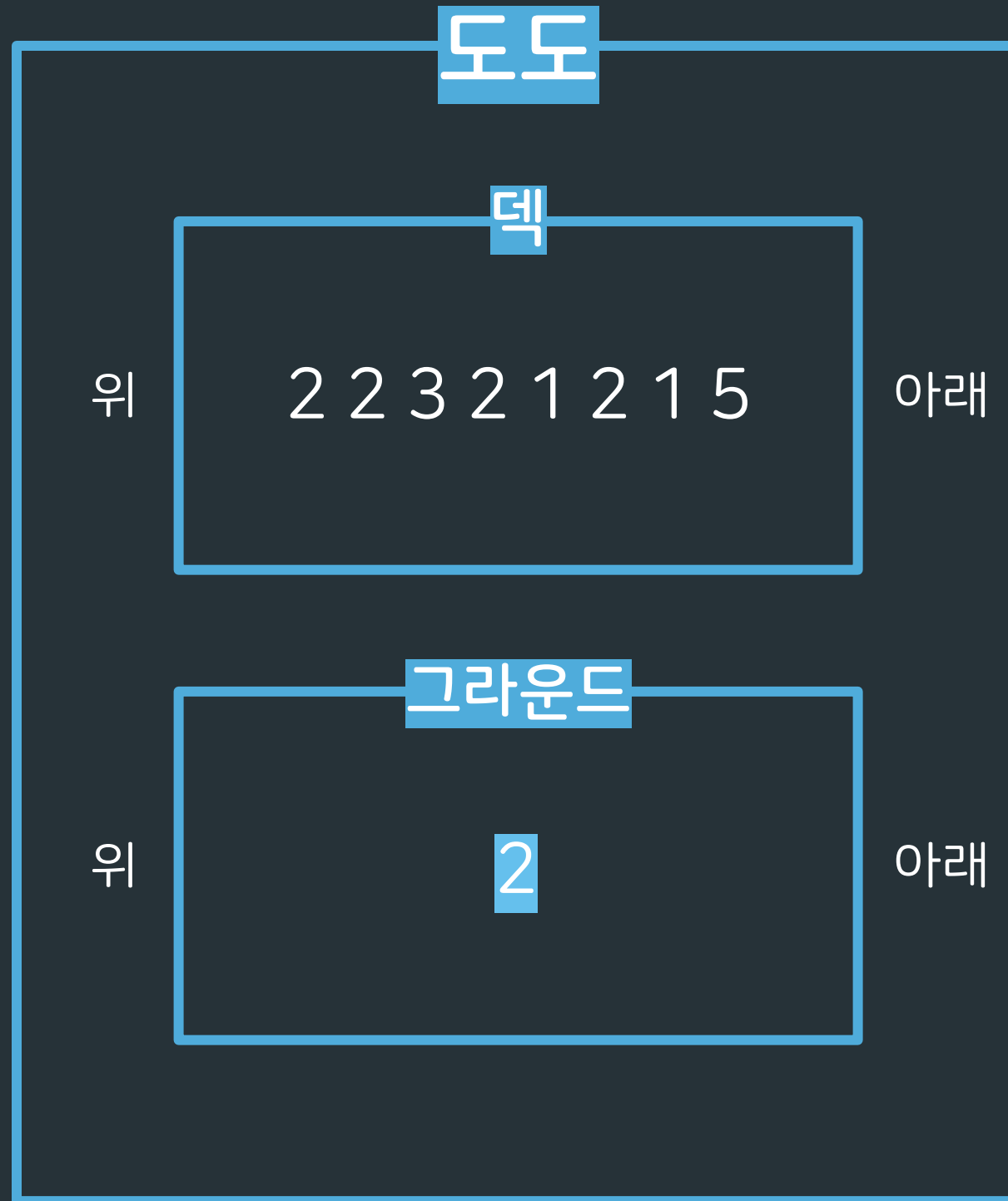
순서대로 게임을 진행해보자

M = 5



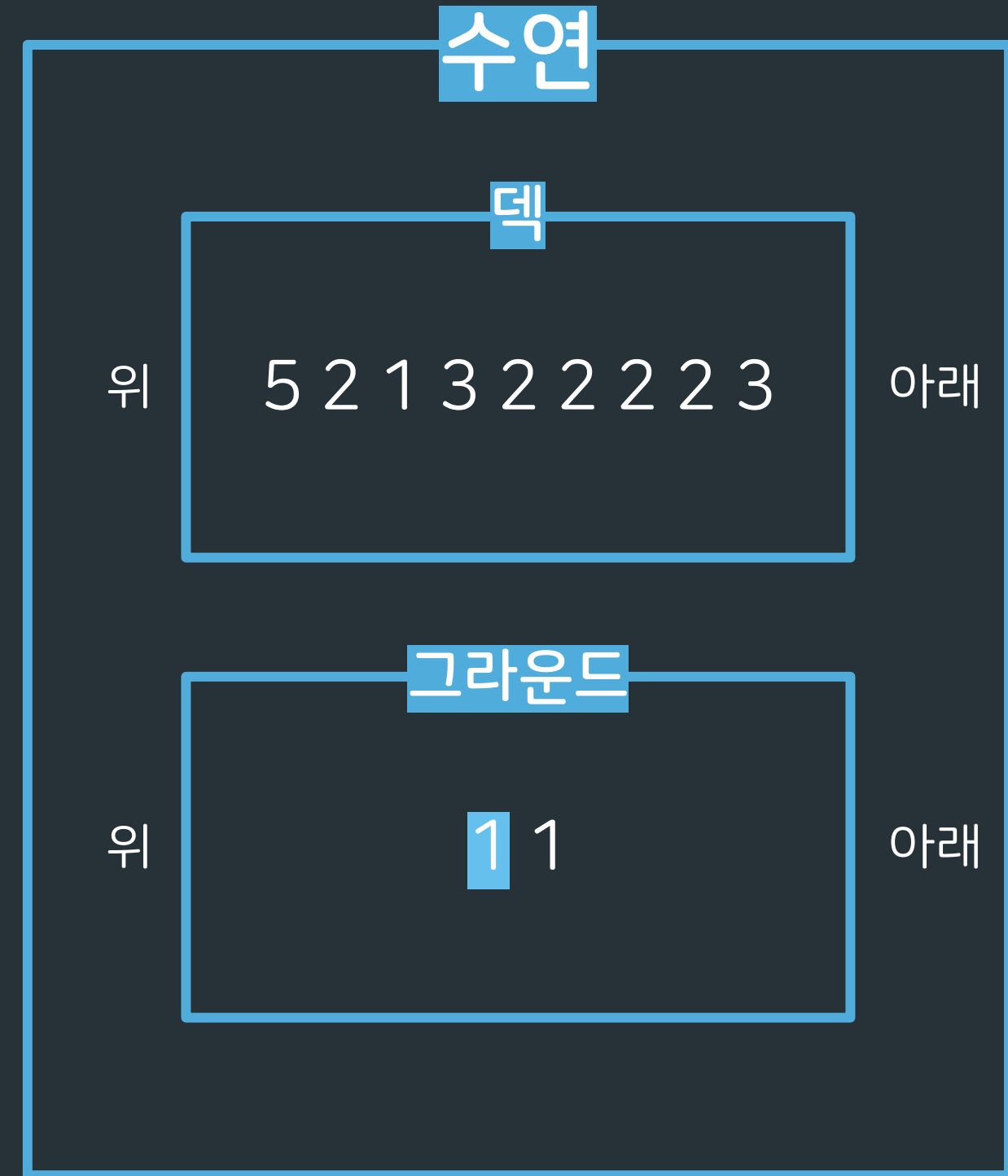
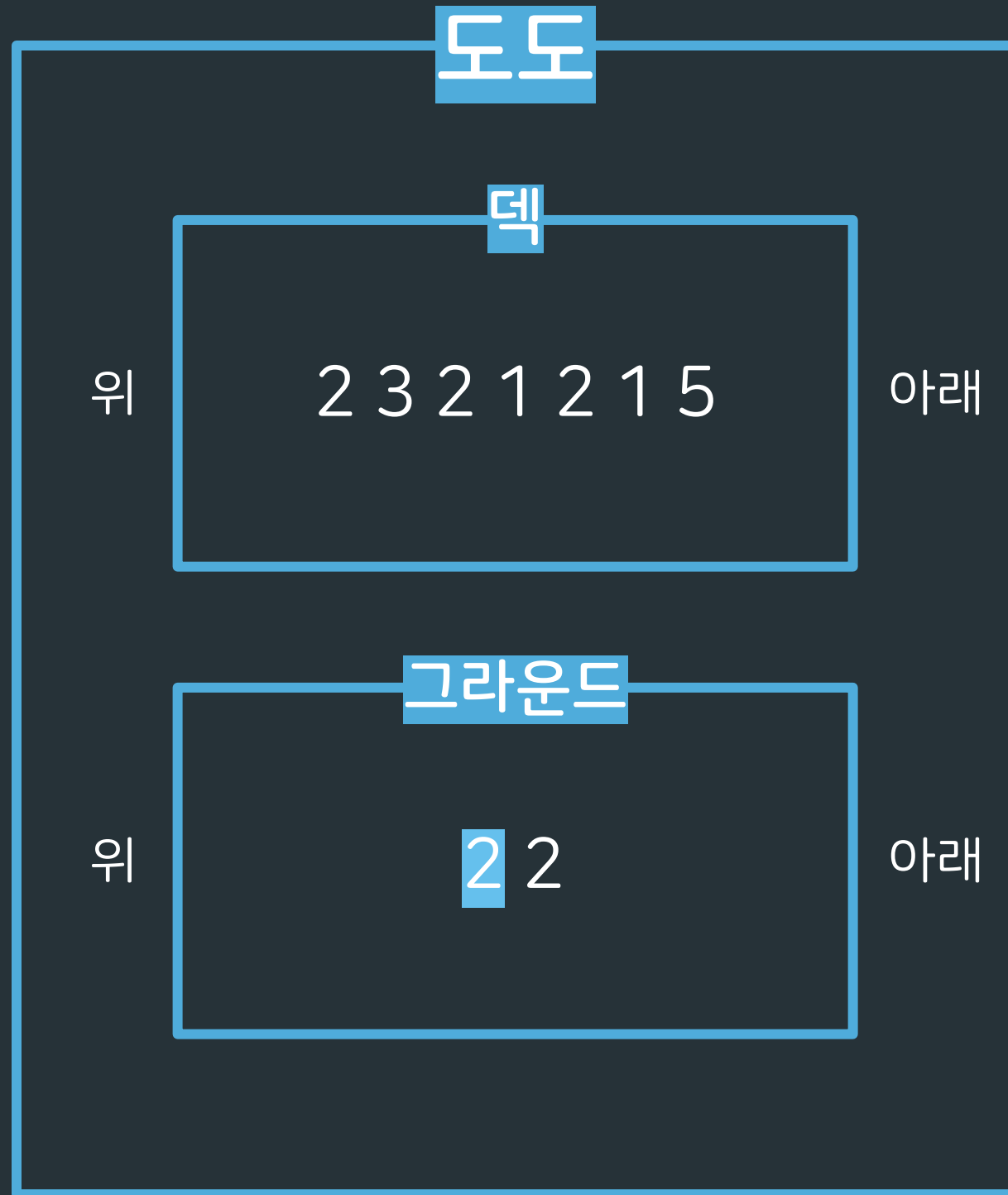
순서대로 게임을 진행해보자

M = 6



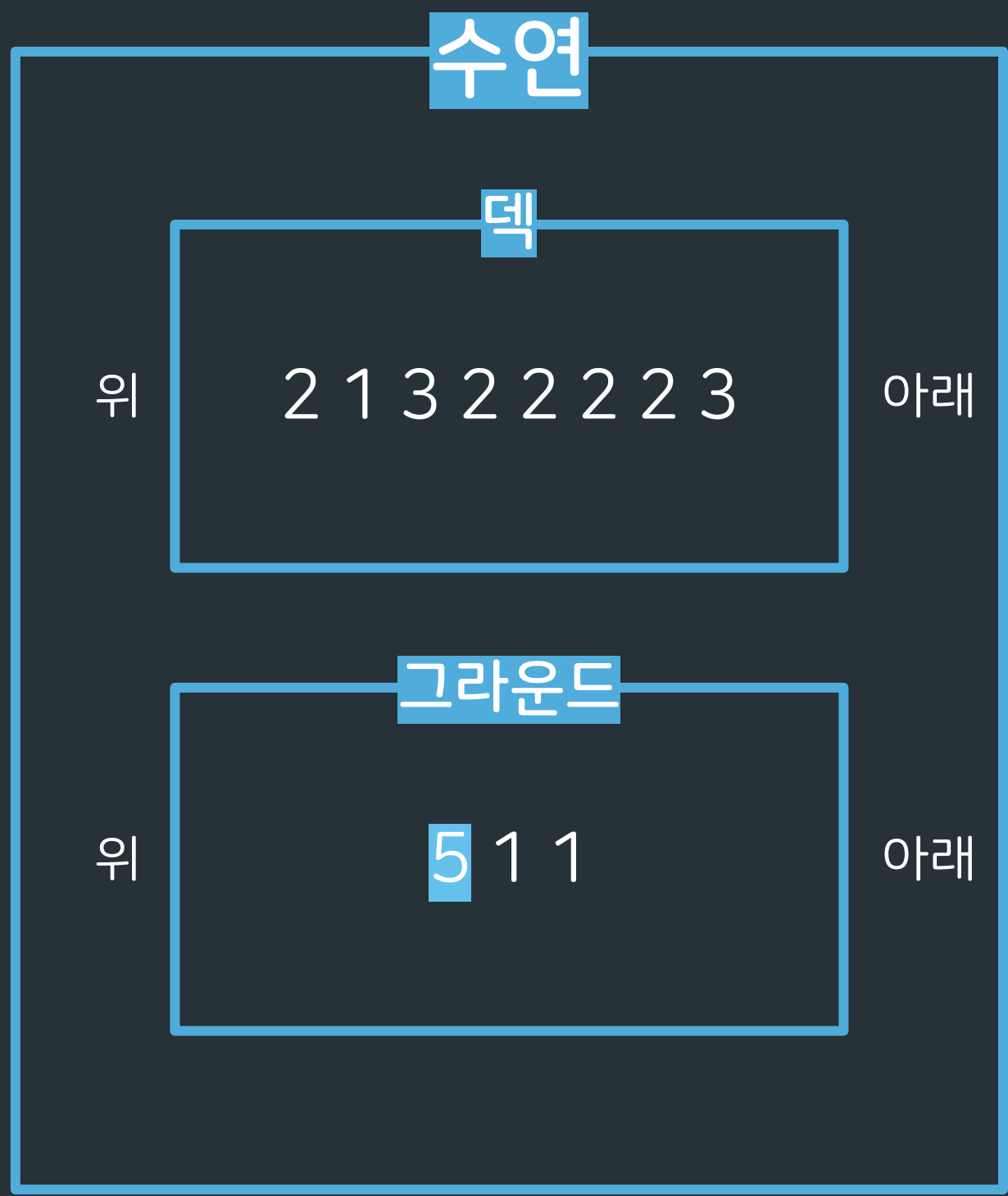
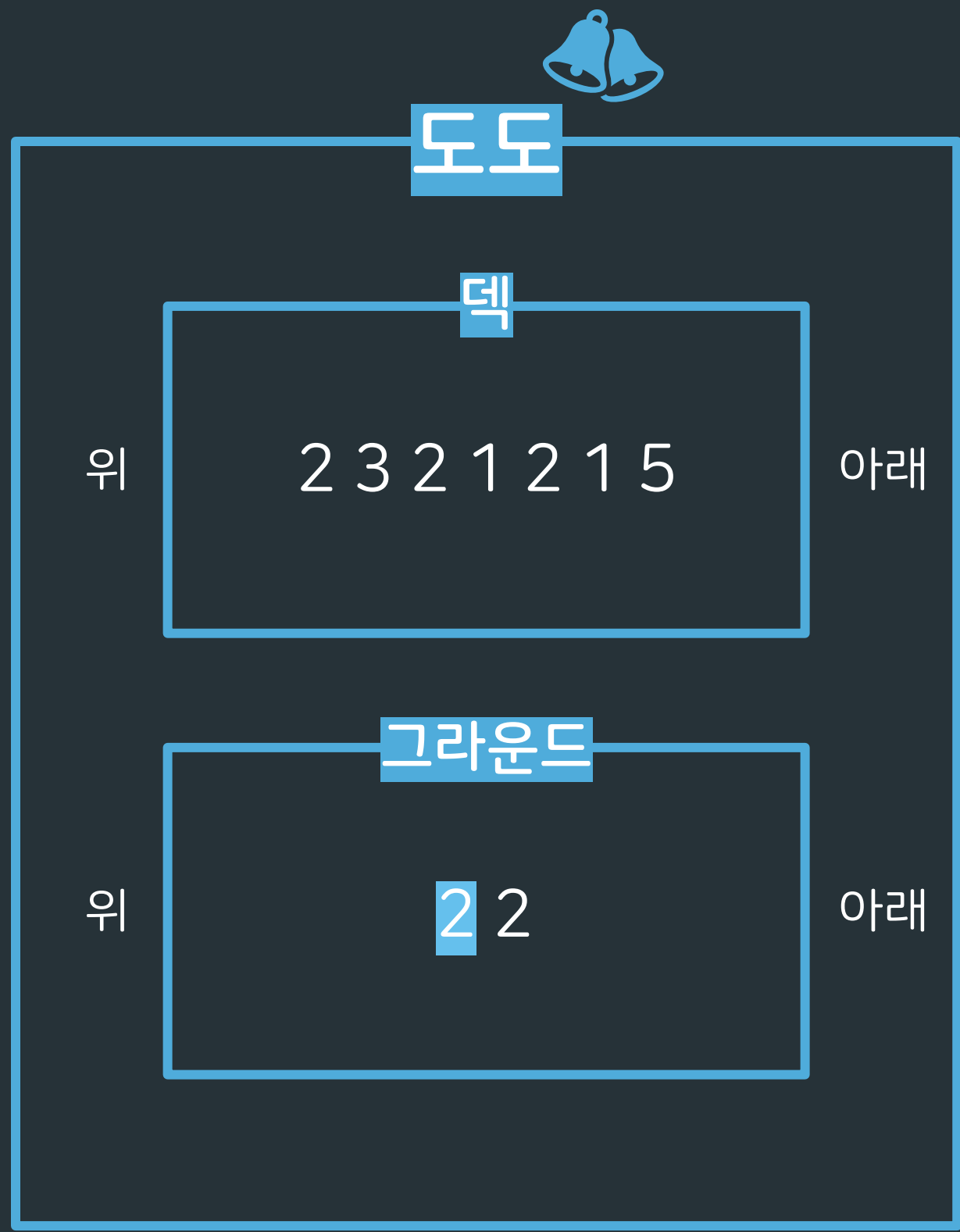
순서대로 게임을 진행해보자

M = 7



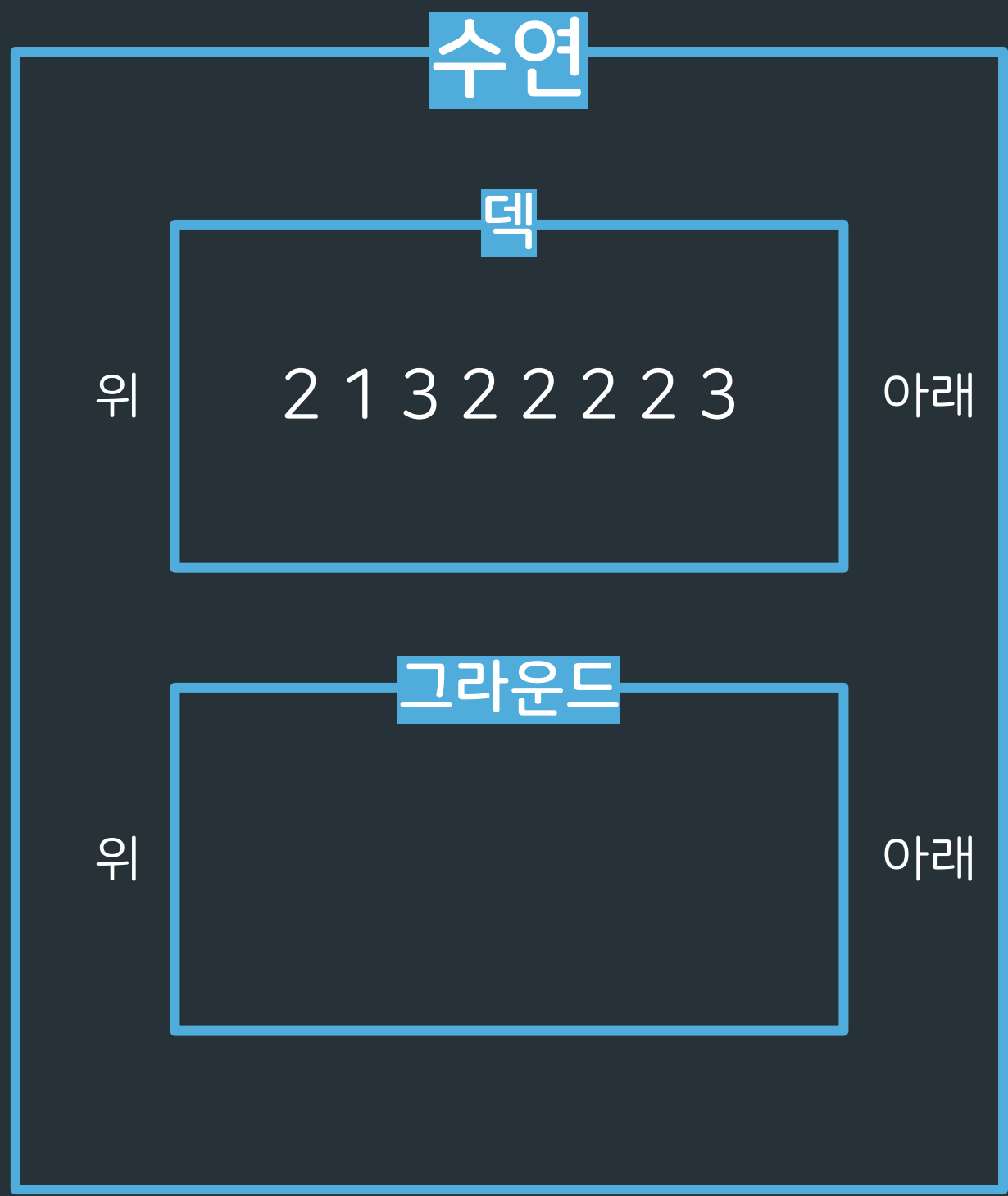
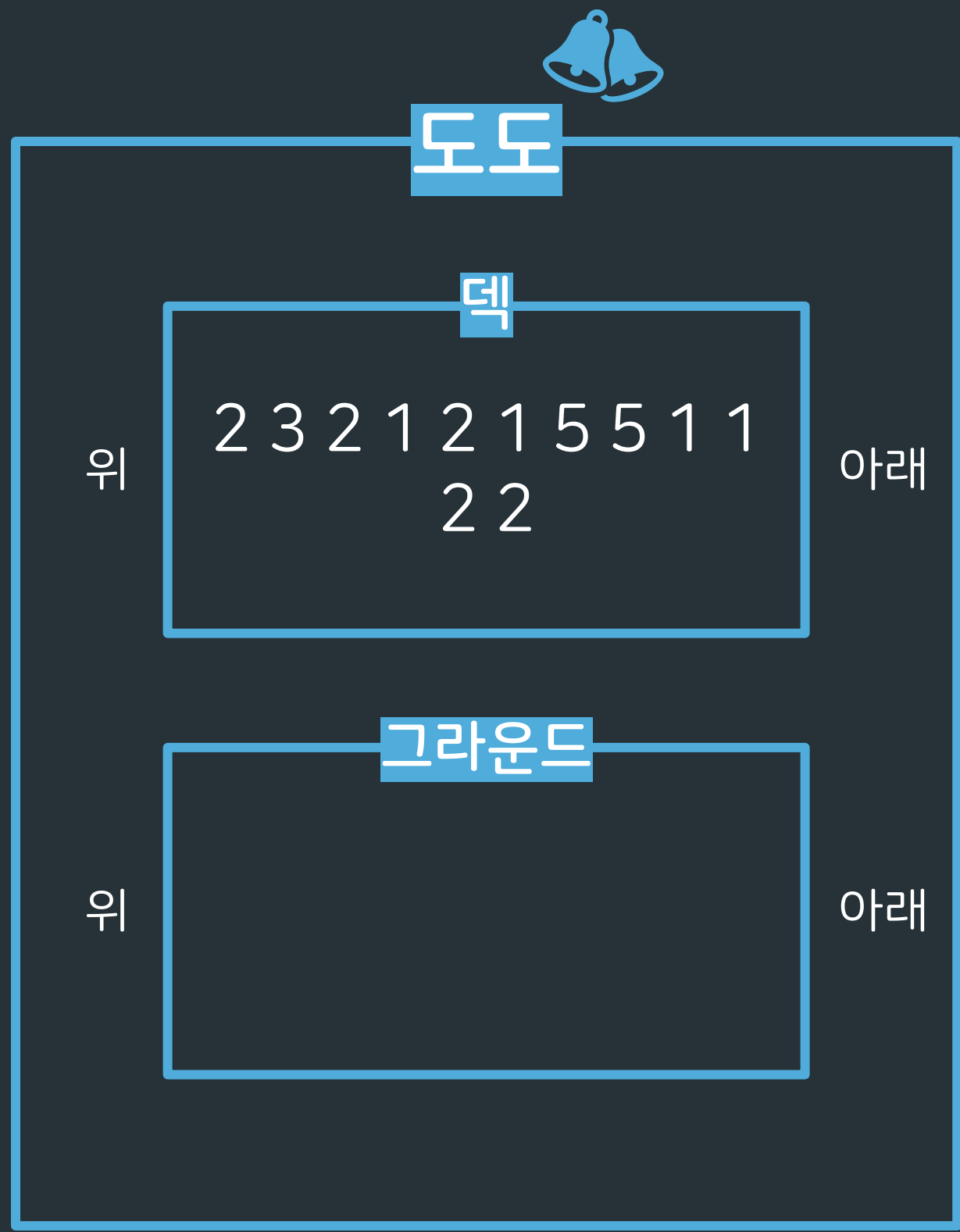
순서대로 게임을 진행해보자

M = 8



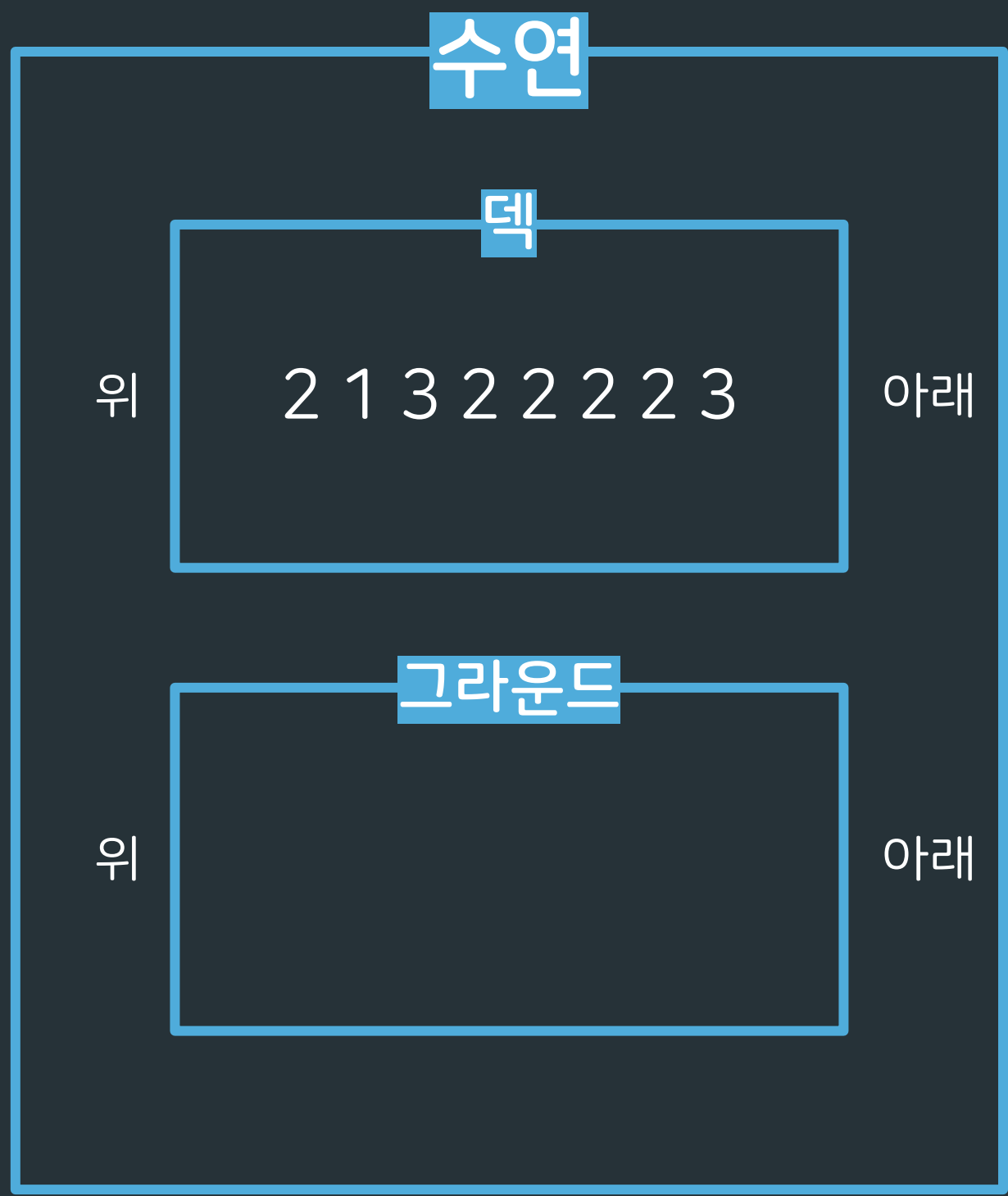
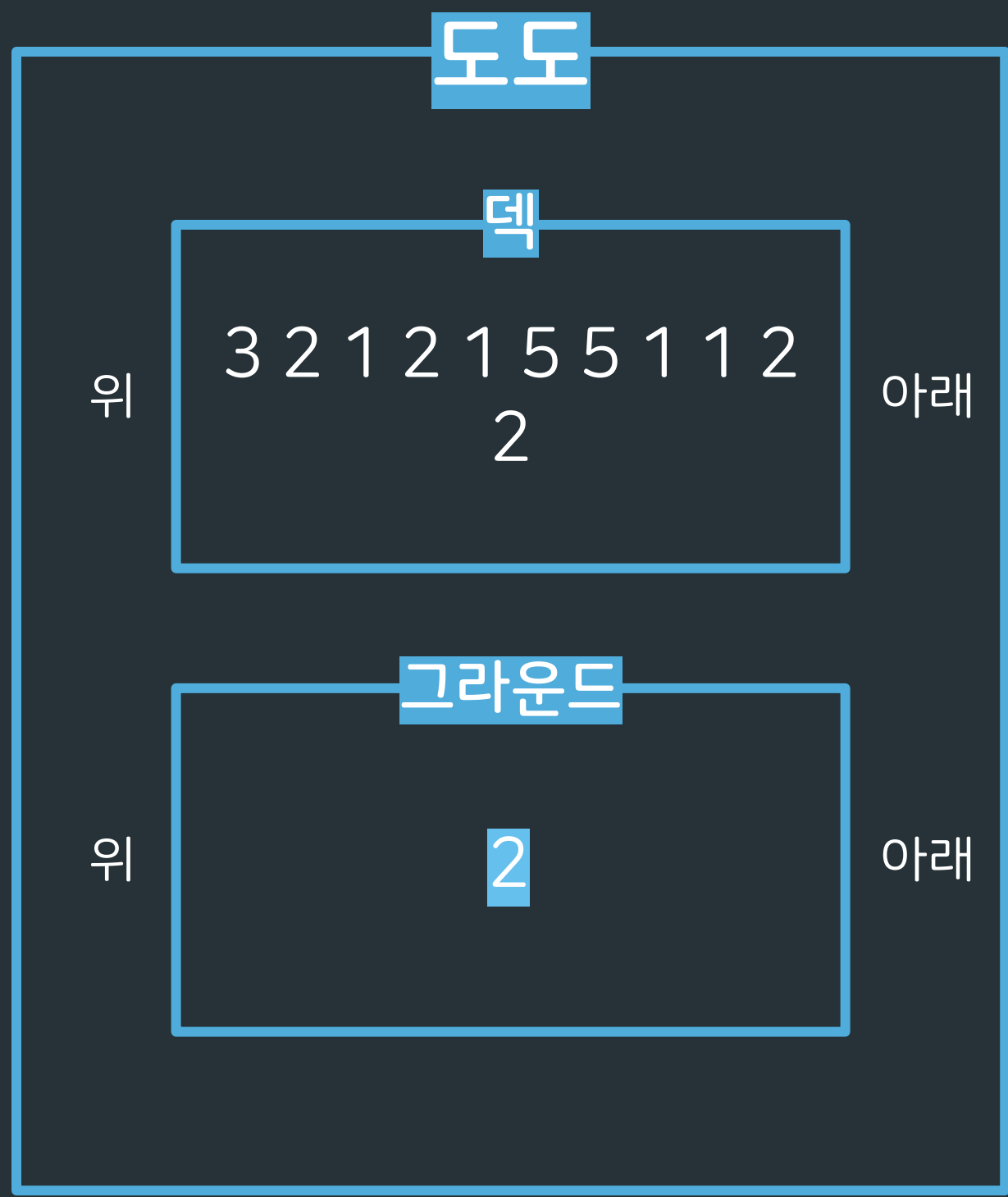
순서대로 게임을 진행해보자

M = 8



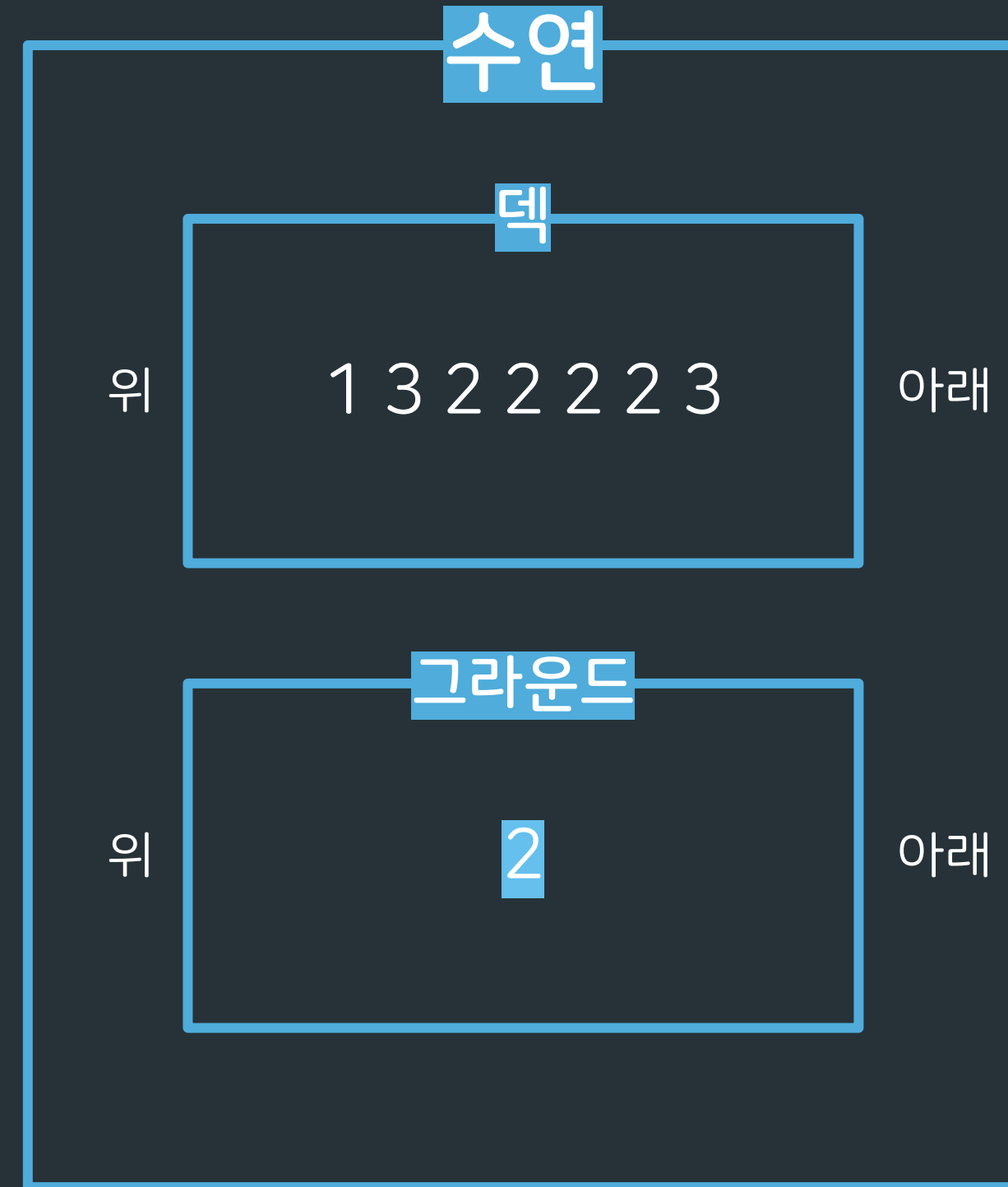
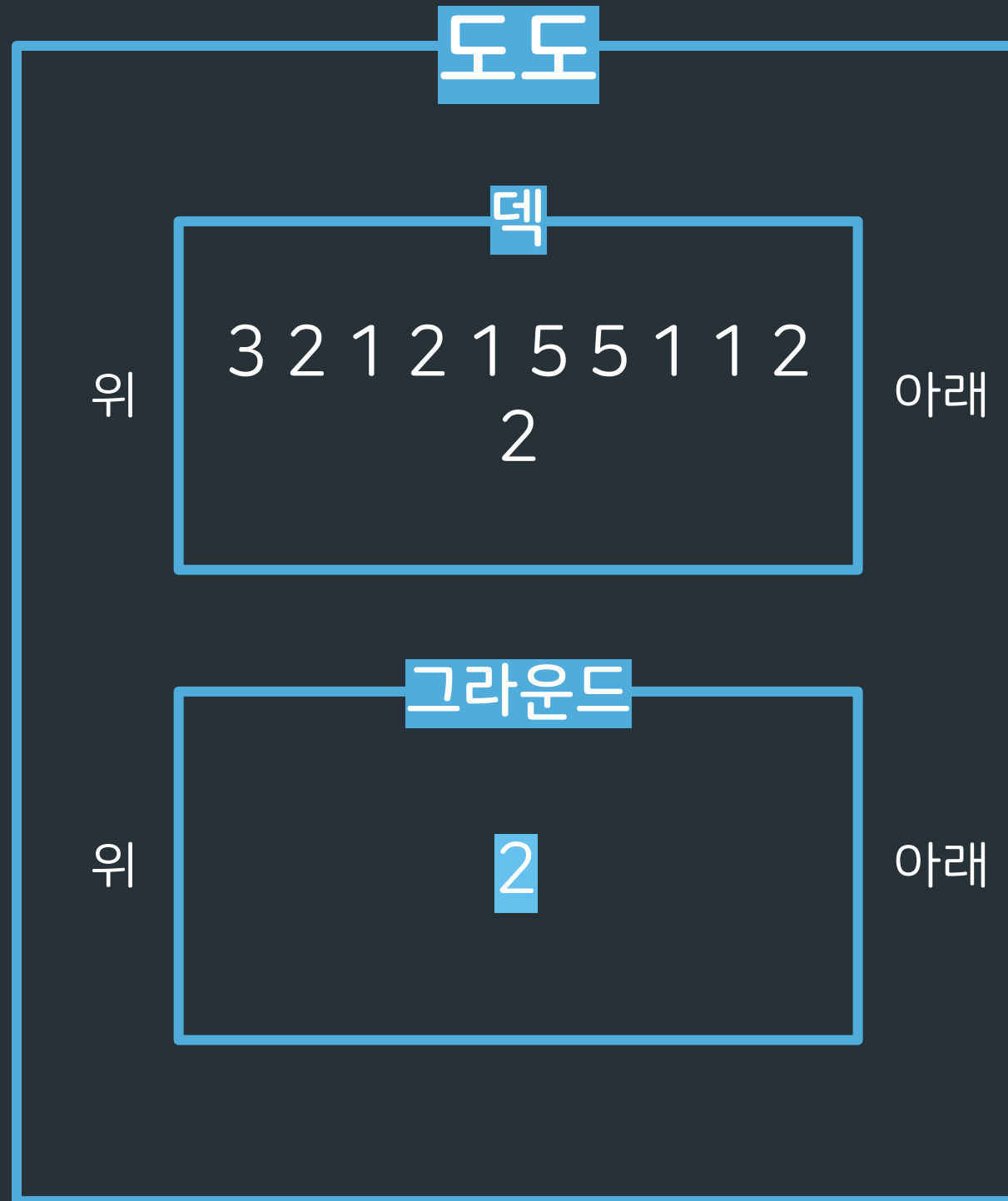
순서대로 게임을 진행해보자

M = 9



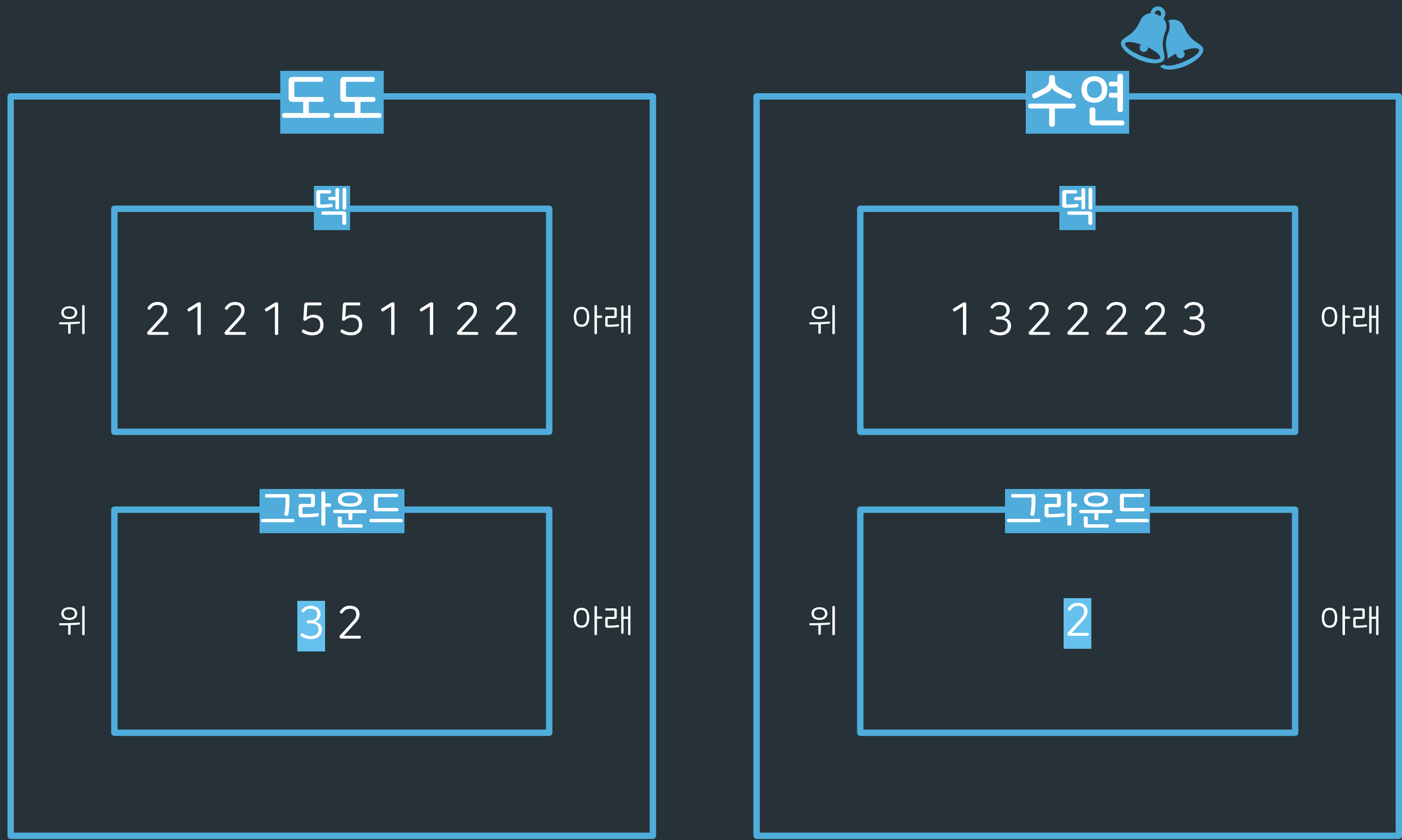
순서대로 게임을 진행해보자

M = 10



순서대로 게임을 진행해보자

M = 11



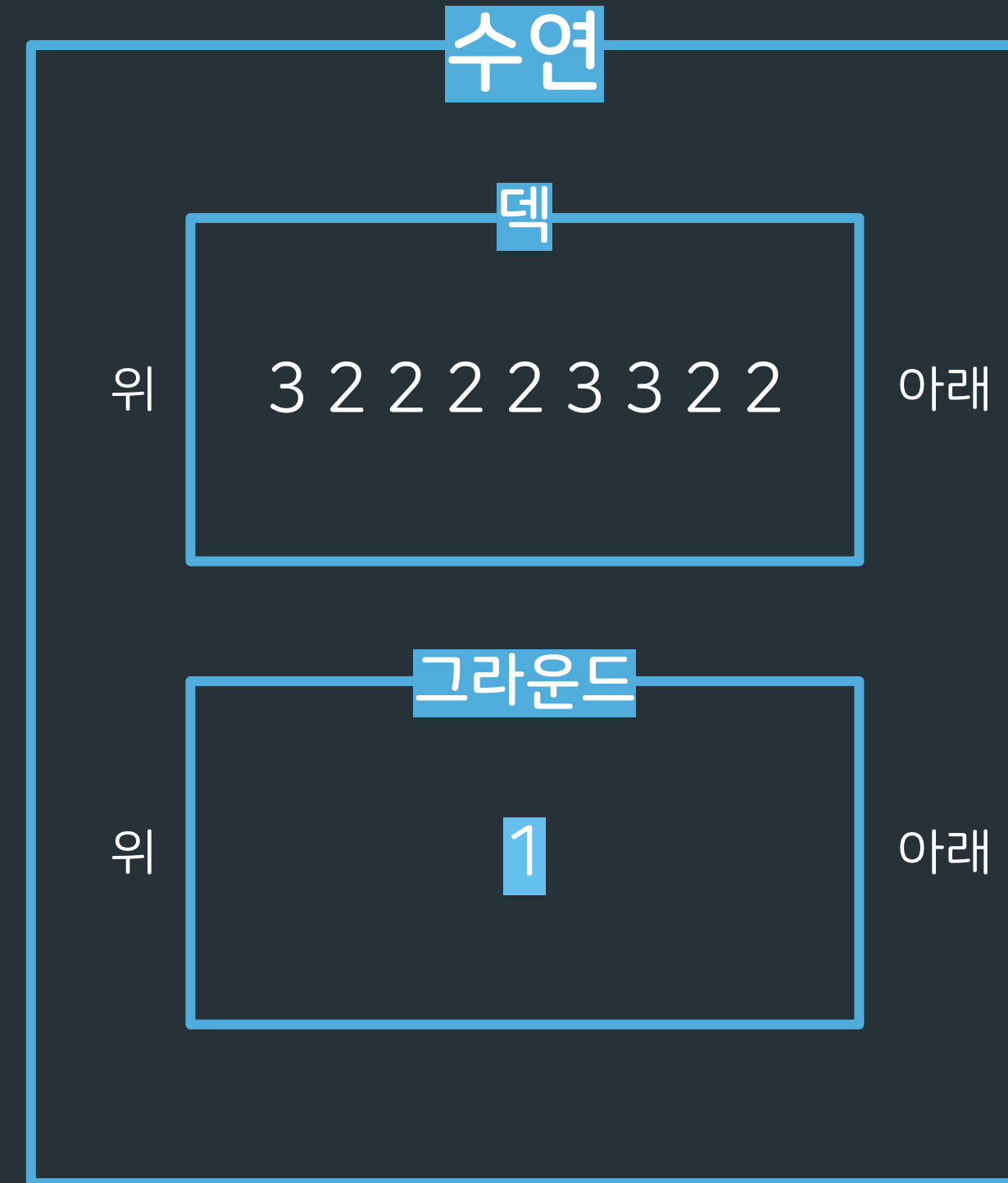
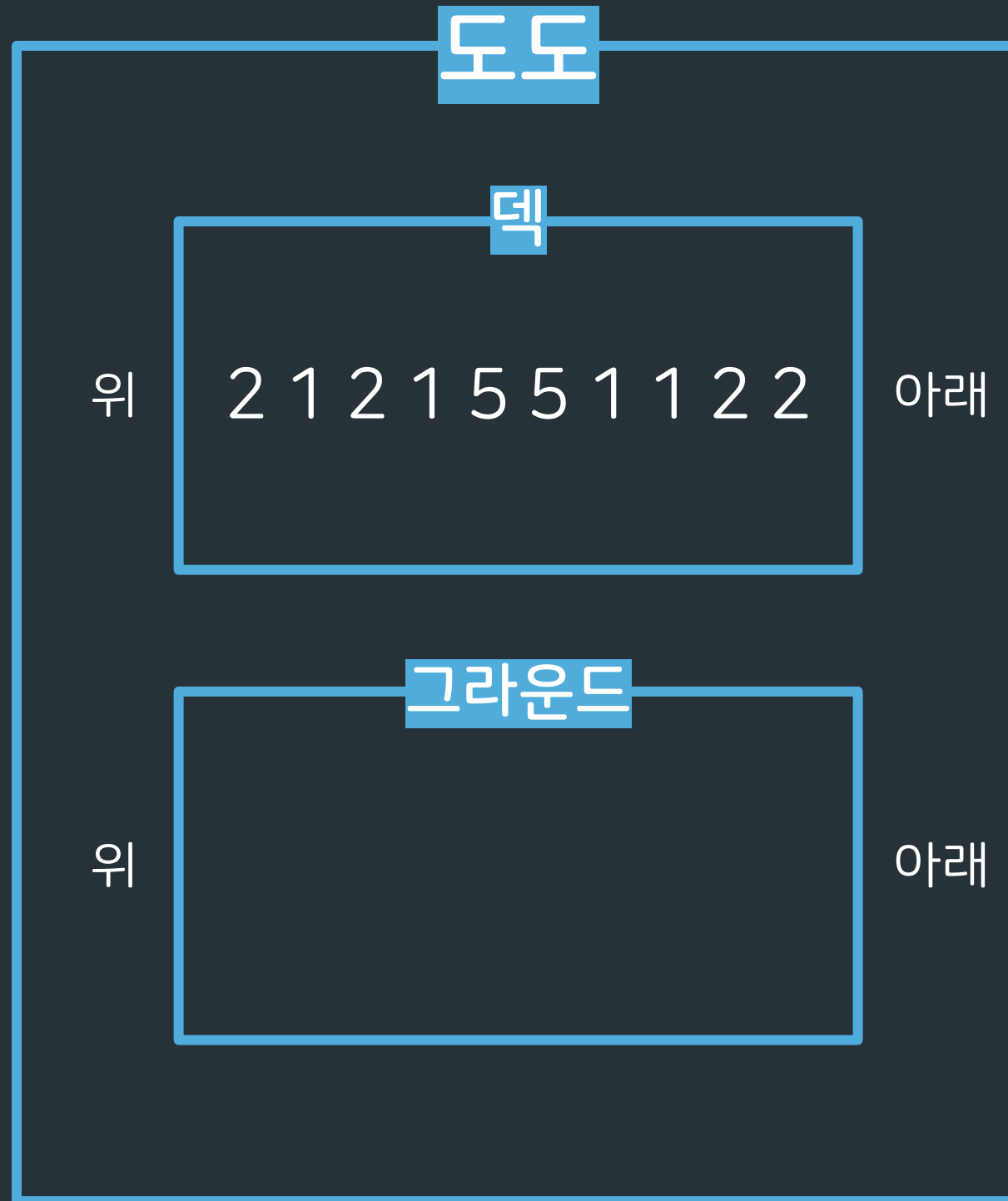
순서대로 게임을 진행해보자

M = 11



순서대로 게임을 진행해보자

M = 12



순서대로 게임을 진행해보자

도도

덱

2 1 2 1 5 5 1 1 2 2

10장

수연

덱

3 2 2 2 2 3 3 2 2

9장

>

카드를 더 많이 가지고 있는 도도의 승리!

도도, 수연은 각각 덱과 그라운드를 가짐(deque)

1. 카드를 덱에서 한 장씩 내려 놓음 (도도 먼저)
2. 어떤 플레이어가 종을 칠 수 있는지 판단 (도도/수연/없음)
 - a. 종을 친 경우 그라운드의 카드를 덱으로 이동
3. 종료 조건 만족 시 승리한 사람 판단

추가로 풀어보면 좋은 문제!

/<> 9655번 : 돌 게임 – Silver 5

/<> 9095번 : 1, 2, 3 더하기 – Silver 3

/<> 2156번 : 포도주 시식 – Silver 1

/<> 9251번 : LCS – Gold 5