Tower of Hanoi

Personal Project

김유진

INDEX


01
개요

프로젝트 개요

하노이 탑 알고리즘 구현

재귀함수를 활용해 “하노이 탑” 알고리즘을 웹사이트로 구현했습니다.

사용자가 원판의 이동 과정을 쉽게 이해할 수 있도록, 이동 경로와 과정을 시각적으로 표현했습니다.

또한, 그림을 활용해 원판 이동의 전체 흐름을 한눈에 파악할 수 있도록 설계했습니다.

개발 기간

24.10.23 - 24.10.27

개발 일정

1일차 : UI 구현 및 로직 설계

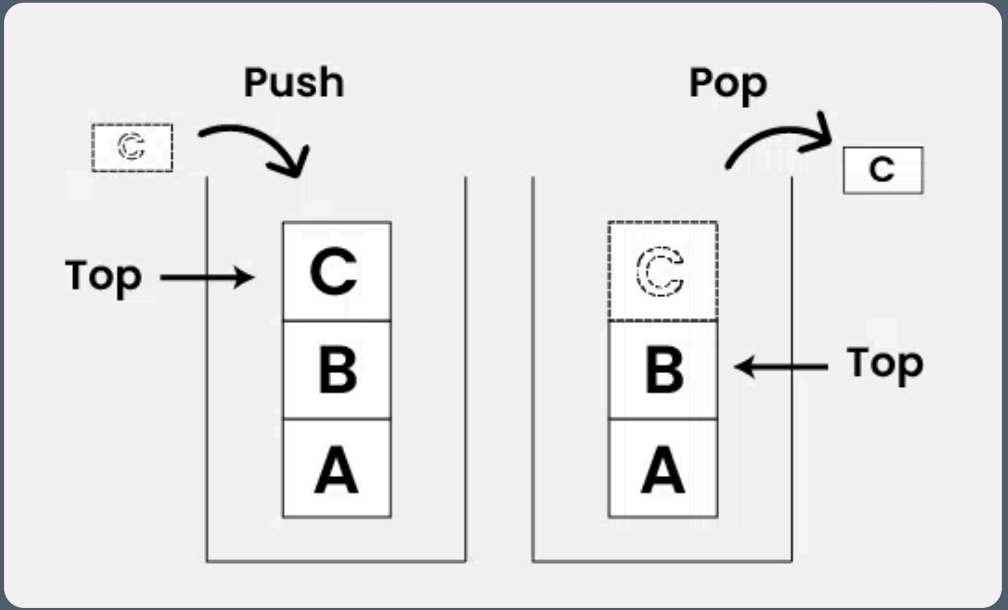
2일차 : 시각화 기능 구현

3일차 : UI 개선 및 오류 수정

4일차 : 프로젝트 후기 문서 작성

자료구조: STACK(스택)

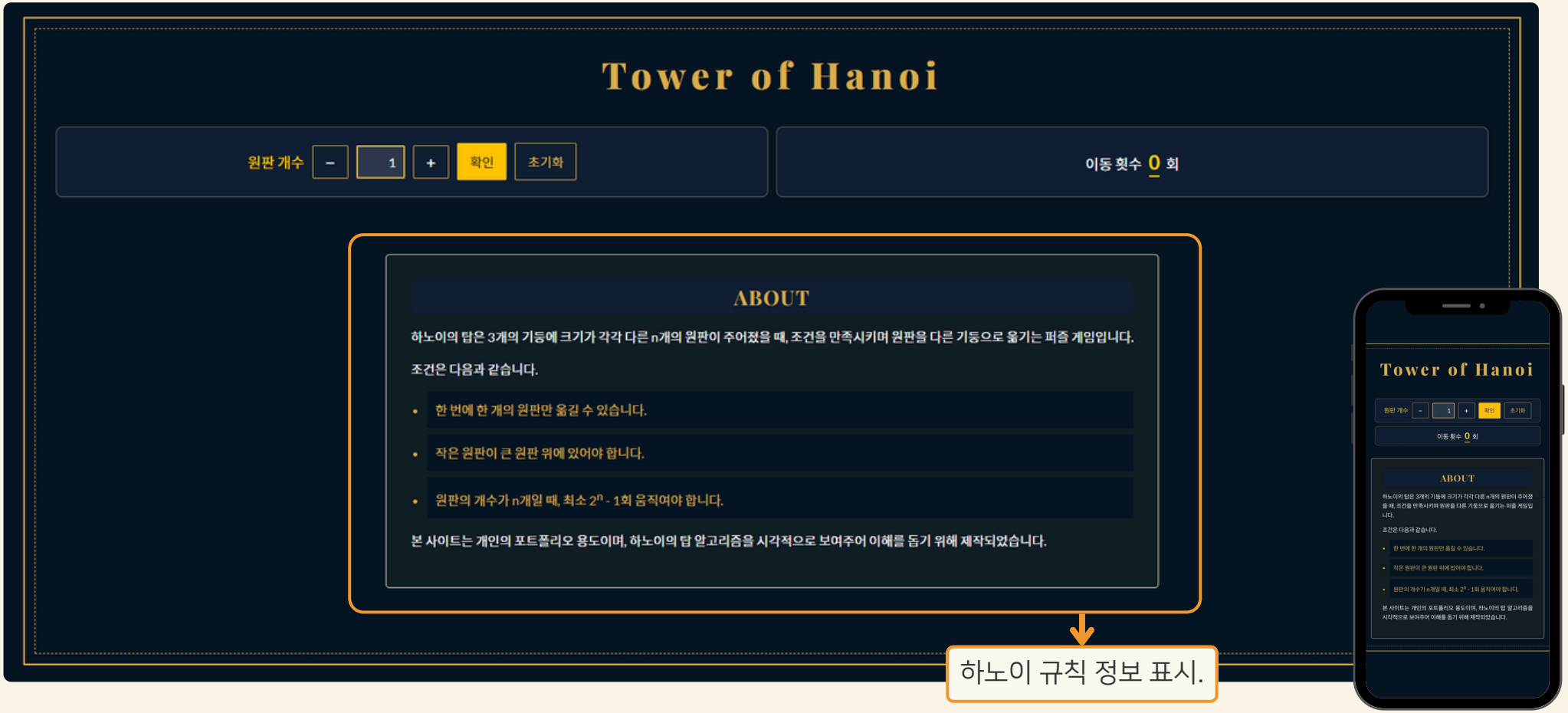
STACK의 LIFO(Last-In, First-Out) 특성을 활용하여,
원판을 저장하고 필요에 따라 순서대로 꺼내는 데 사용했습니다.



- 스택은 LIFO(Last-In, First-Out) 원칙에 따라 데이터를 관리하는 자료구조입니다.
- Push : 데이터를 스택의 최상단에 추가하는 연산입니다.
- Pop : 스택의 최상단 데이터를 제거하는 연산입니다.

02
화면설계

기본화면



- **기본 화면**에는 하노이의 탑에 대한 **설명과 게임 규칙이 포함된 정보**가 표시됩니다.
- 사용자는 입력값(원판 개수)을 설정한 후 **확인 버튼을 눌러 게임을 시작**할 수 있습니다.
- **초기화 버튼**으로 **입력값과 화면을 초기 상태**로 되돌릴 수 있도록 구성되어 있습니다.

입력값 예외 처리 및 부가 기능

원판 개수 입력

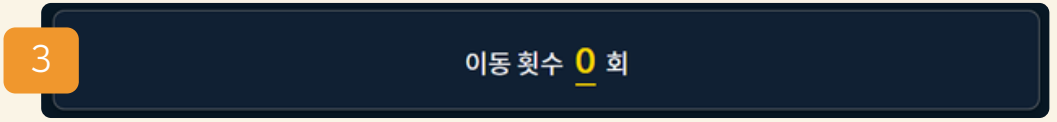
- 원판 개수를 입력할 수 있는 입력창이 있으며, 최소값은 1로 제한됩니다.
- 입력값이 1보다 작으면 **자동으로 1**로 설정되고, 7 이상의 값을 입력하면 **최대값인 7**로 조정됩니다.

증감 버튼

- +와 - 버튼으로 입력값을 **증가 또는 감소**시킬 수 있으며, 값은 **1~7 범위** 내에서만 변경 가능합니다.

초기화 버튼

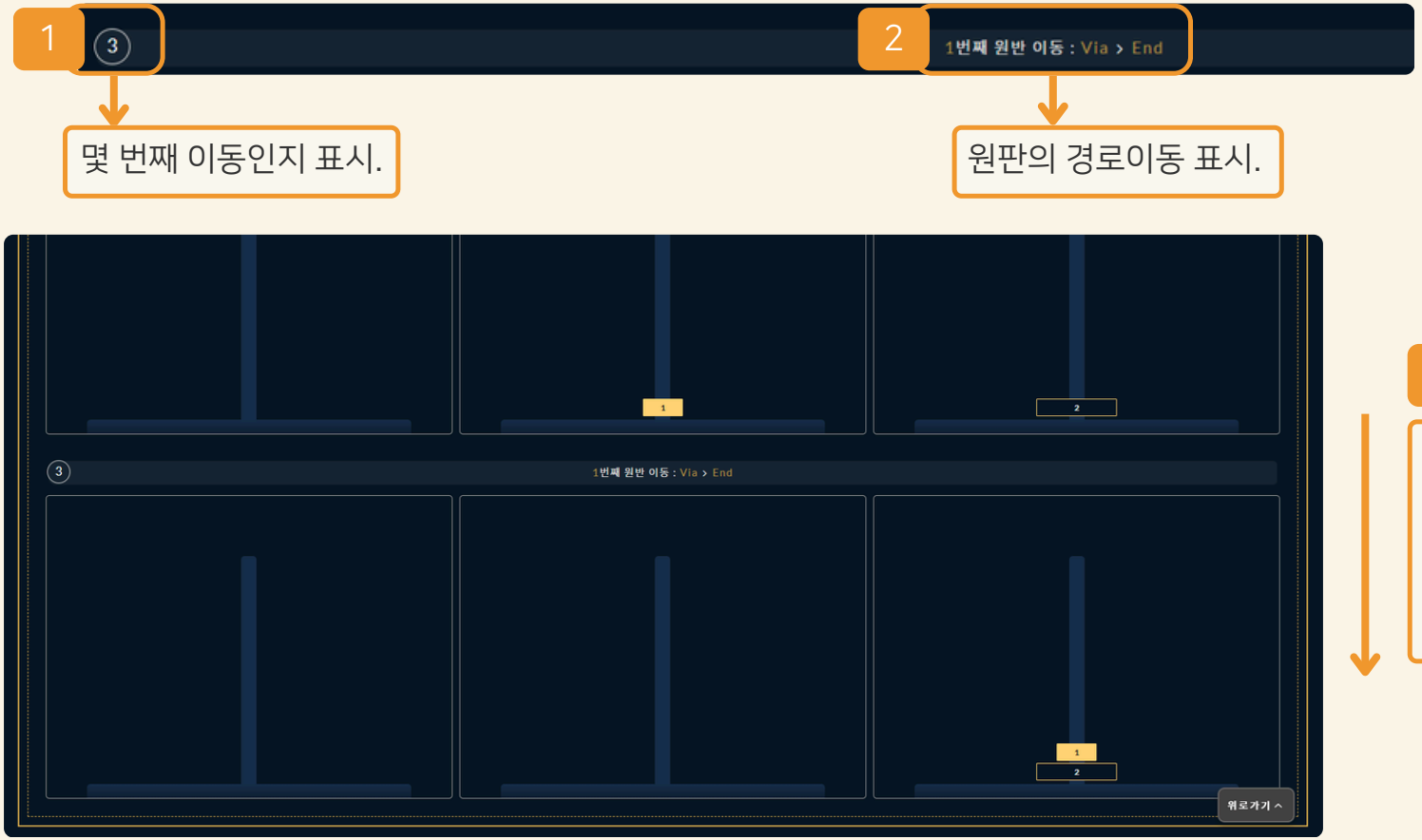
- **초기화 버튼**을 클릭 시, 화면이 **처음 상태**로 리프레시되며, **초기 안내 메시지**(ABOUT)가 **표시**됩니다.



이동 횟수 표시

- **원판 개수를 입력**하고 **확인 버튼**을 누르면 하노이 탑의 **이동 횟수가 표시**됩니다.

하노이 탑 이동 시 화면 처리

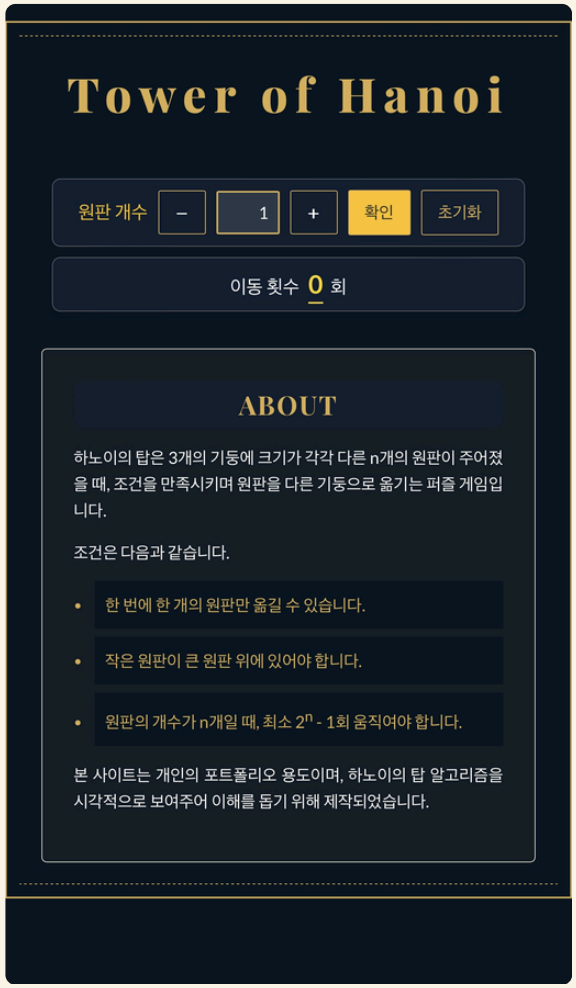


3

부드러운 화면 전환을 위해 scrollTo 메서드에 behavior: smooth 옵션이 적용되어 있음.

- 원판 개수를 입력하고 확인 버튼을 누르면, **원판의 상세 경로와 이동 횟수가 표시**됩니다.
- 결과 화면으로 자연스럽게 이동될 수 있도록 **하단으로 부드럽게 스크롤 되는 기능**이 적용되어 있습니다.
- 이 기능을 통해 하노이 탑의 알고리즘 동작을 직관적으로 이해할 수 있도록 구현하였습니다.

반응형 화면 구현



모바일 **기본 화면**

실행 화면

- **하단**으로 이동

위로 가기 버튼 클릭 시

- **상단**으로 이동


03
코드

JAVASCRIPT / Stack 자료구조를 활용한 배열 생성

```
1  // ^Stack Class 선언
2  class Stack {
3      constructor(id) {
4          this.id = id;
5          this.storage = [];
6      }
7      shiftItem() {
8          return this.storage.shift();
9      }
10     unshiftItem(item) {
11         this.storage.unshift(item);
12     }
13     clear() {
14         this.storage = [];
15     }
16     get stackId() {
17         return this.id;
18     }
19     get stackStorage() {
20         return [...this.storage];
21     }
22 }//$Stack
```

- Stack 자료구조의 특성을 활용하여 배열을 구현한 클래스입니다.
- 전역 속성에 직접 접근을 제한하기 위해 get 메서드를 사용하여 “id” 와 “storage” 값을 반환하도록 설계했습니다.

JAVASCRIPT / 하노이타워 로직 구현 클래스(1)

```
1  // ^Hanoi Class 선언
2  class Hanoi {
3      constructor(id) {
4          this.id = id;
5          this.fromStack = new Stack("Start");
6          this.viaStack = new Stack("Via");
7          this.toStack = new Stack("End");
8          this.count = 0;
9      }
10
11     getHanoi(n, from, to, via, element) {
12         ...
13     }
14     initialHTML(element) {
15         ...
16     }
17     displayHTML(element) {
18         ...
19     }
20     scrollEvent(element) {
21         ...
22     }
23     listener(currentValue, btnGroup) {
24         ...
25     }
26     run(element) {
27         ...
28     }
29 }//$Hanoi Class
```

- Hanoi 클래스는 하노이 타워 문제를 구현하기 위한 구조입니다.
- 전역 속성으로 fromStack, viaStack, toStack을 설정하여 각 기둥의 상태를 Stack으로 관리합니다.
- count 속성은 하노이 타워에서 원반이 이동된 총 횟수를 기록하는 속성입니다.

JAVASCRIPT / 하노이타워 로직 구현 클래스(2)

```
1  // ^Hanoi Class 선언
2  class Hanoi {
3      constructor(id) {
4          this.id = id;
5          this.fromStack = new Stack("Start");
6          this.viaStack = new Stack("Via");
7          this.toStack = new Stack("End");
8          this.count = 0;
9      }
10
11     getHanoi(n, from, to, via, element) {
12         ...
13     }
14     initialHTML(element) {
15         ...
16     }
17     displayHTML(element) {
18         ...
19     }
20     scrollEvent(element) {
21         ...
22     }
23     listener(currentValue, btnGroup) {
24         ...
25     }
26     run(element) {
27         ...
28     }
29 }//$Hanoi Class
```

- getHanoi: 하노이의 로직을 실행하는 메서드.
- initialHTML: 초기 DOM 화면을 구성하는 메서드.
- displayHTML: 실행된 하노이 로직을 DOM에 시각화 시키는 메서드.
- scrollEvent: 하노이를 시각화한 후 부드럽게 화면을 맨 하단으로 이동시키는 메서드.
- listener: 버튼 클릭이나 input 입력 시 해당 기능을 처리하기 위한 통합 메서드.
- run: 위의 메서드들을 한 번에 제어하여 전체 기능을 실행하는 메서드.

JAVASCRIPT / 하노이의 탑 이동 로직 구현

```
getHanoi(n, from, to, via, element) {
        if (n === 1) {
            const unshiftItem = from.shiftItem();
            // from >>> to
            to.unshiftItem(unshiftItem);
            this.count++;
            document.getElementById("towerBoxWrap").innerHTML += `
            <article class="moveLog">
                <p id="moveLogCount" class="moveLogCount">${this.count}</p>
                <h2 class="moveLogTitle">
                    <span id="diskNumber">${unshiftItem}</span>번째 원반 이동 :
                    <span id="startPoint">${from.stackId}</span>
                    <i class="xi-angle-right"></i>
                    <span id="endPoint">${to.stackId}</span>
                </h2>
            </article>`;

            this.displayHTML(document.getElementById("towerBoxWrap"));
            return;
        }
        // ### Step 1: n-1개의 디스크를 보조 기둥(via)으로 옮긴다. ###
        this.getHanoi(n - 1, from, via, to, element);
        // ### Step 2: 가장 큰 디스크를 목표 기둥으로 옮긴다. ###
        const unshiftItem = from.shiftItem();
        // from >>> to
        to.unshiftItem(unshiftItem);
        // == count 하기 ==
        this.count++;

        document.getElementById("towerBoxWrap").innerHTML += `
            // 위와 동일한 코드
        `;
        this.displayHTML(document.getElementById("towerBoxWrap"));

        // ### Step 3: 다시 보조 기둥에 있는 n-1개의 디스크를 목표 기둥으로 옮긴다. ###
        this.getHanoi(n - 1, via, to, from, element);
    }
```

- 재귀 함수 활용: 디스크를 중간 기둥(via) 경유해 목적 기둥(to)으로 순차적으로 이동합니다.
- 스택 클래스 활용: fromStack, toStack 등을 사용해 디스크 이동을 데이터로 관리합니다.
- 이동 과정 기록: 이동된 디스크 번호와 출발/도착 기둥 정보를 DOM에 실시간으로 시각화합니다.
- 이동 단계 구현: 1단계에서 경유 기둥으로 이동, 2단계에서 목표 기둥으로 최종 이동합니다.

JAVASCRIPT / 하노이 탑 시각화 구현

```
displayHTML(element) {
    const createTower = (stack,DOMId) => {
        let articleElement = document.createElement("article");
        articleElement.setAttribute("class", "outputBox pictureOutputBox");

        let towerList = document.createElement("ul");
        towerList.setAttribute("id", DOMId); // 매개변수 DOMId 할당
        towerList.setAttribute("class", "towerItemGroup");

        // 매개변수로 받은 stack의 stackStorage 배열
        stack.stackStorage.forEach((number) => {
            let listItem = document.createElement("li");
            listItem.setAttribute("class", `towerStyle item${number}`);
            listItem.textContent = number;
            towerList.appendChild(listItem);
        });
        let base = document.createElement("div");
        base.setAttribute("class", "base");

        let rod = document.createElement("div");
        rod.setAttribute("class", "rod");
        articleElement.append(towerList,base,rod);

        return(articleElement);
    }
    const fromElement = createTower(this.fromStack, "AtowerArea");
    const viaElement = createTower(this.viaStack, "BtowerArea");
    const toElement = createTower(this.toStack, "CtowerArea");
    // DOM요소에 추가
    element.append(fromElement, viaElement, toElement);
}//
```

- 하노이의 탑 시각화: from, via, to 영역에 하노이탑을 시각적으로 생성하여 표시하는 메서드입니다.
- Stack 데이터 활용: stack.getStorage를 호출하여 현재 스택에 저장된 데이터를 순회하며 DOM을 생성합니다.
- DOM 추가: createTower 메서드를 활용해 생성된 각 영역을 DOM에 추가하여 화면에 시각화합니다.

JAVASCRIPT / 이벤트 통합 관리

```
listener(currentValue, btnGroup) {
    const btnGroupObj = { ...btnGroup };
    // ===버튼별 이벤트 할당 ===
    for (let btnItem in btnGroupObj) {
        if (document.getElementById(btnItem)) {
            document.getElementById(btnItem).addEventListener(btnGroupObj[btnItem], (event) => {
                // inputValue
                const inputValue = document.getElementById(currentValue);
                // Btn별 event실행문
                switch (event.target.id) {
                    case "increaseBtn":
                        // increaseBtn 기능 기재
                        break;
                    case "decreaseBtn":
                        // decreaseBtn 기능 기재
                        break;
                    case "confirmBtn":
                         // confirmBtn 기능 기재
                        break;
```

```
run(element) {
    const btnGroup = {
        increaseBtn: "click",
        decreaseBtn: "click",
        confirmBtn: "click",
        resetBtn: "click",
        numberInput: "input",
        scrollToTopBtn: "click"
    };
    this.initialHTML(element); // 초기화면
    this.listener("numberInput", btnGroup); //all_event
    this.scrollEvent("scrollToTopBtn"); //scroll_event
}
```

- 발생 가능한 요소와 이벤트 타입을 객체로 정의해 통합적으로 관리하는 메서드입니다.
- for in문을 사용해 이벤트 타입과 DOM ID를 순회하며, 해당 요소에 eventListener를 동적으로 추가합니다.
- 버튼 클릭 및 입력 이벤트 등 여러 상호작용을 한 번에 처리할 수 있도록 설계되었습니다.
- 이벤트 타입과 대상 DOM ID를 분리해 유지보수가 용이하도록 구현했습니다.

JAVASCRIPT / 이벤트: input창 예외처리 기능


```
case "numberInput":
    // ==input창 예외처리==
    (
        (document.getElementById(btnItem).value > 7)
        && (document.getElementById(btnItem).value = document.getElementById(btnItem).value.slice(-1)) > 7
        && (document.getElementById(btnItem).value = 7)
    );

    (
        (document.getElementById(btnItem).value < 1)
        && (document.getElementById(btnItem).value = 1)
    );
```

- if문을 사용하지 않고 논리 연산자 &&(and)의 특성을 활용해 예외처리를 구현했습니다.
- 입력값이 7보다 크거나 1보다 작은 경우, &&(and) 조건을 통해 값이 자동으로 조정됩니다.
- 불필요한 조건문을 줄여 코드의 가독성과 효율성을 높였습니다.

JAVASCRIPT / 이벤트: 확인버튼 로직 흐름도


```
case "confirmBtn":
    const inputNumber = Number(document.getElementById("numberInput").value);
    // ==모든 배열 클리어==
    this.fromStack.clear();
    this.viaStack.clear();
    this.toStack.clear();
    this.count = 0;
    // ==시작 스택 쌓기==
    for (let i = inputNumber; i > 0; i--) {
        this.fromStack.unshiftItem(i);
    }
    // ==출력영역 초기화 ==
    document.getElementById("towerBoxWrap").innerHTML = "";
    // ==getHanoi 실행==
    this.getHanoi(inputNumber, this.fromStack, this.toStack, this.viaStack, document.getElementById("towerBoxWrap"));
    document.querySelector('.moveCount').textContent = this.count;
    // == 출력 후 맨 아래로 이동 == )
    window.scrollTo({ left: 0, top: (document.body.scrollHeight), behavior: 'smooth' });
    break;
```

- 입력된 원판 개수에 따라 초기 스택을 설정하고 하노이 타워 로직을 실행합니다.
- 모든 스택과 DOM 영역을 초기화한 후, getHanoi 메서드를 통해 하노이 타워 알고리즘을 구현합니다.
- 완료 후 window.scrollTo를 사용해 부드럽게 화면 하단으로 이동하도록 구현했습니다.

04
후기

“재귀를 이해하며 성장한 하노이의 타워 프로젝트”

하노이의 탑 규칙을 하나씩 노트에 적으며 **규칙성**을 찾아냈지만, 이를 **코드로 구현하는 과정**은 생각보다 쉽지 않았습니다. 비슷한 순열 문제를 해결할 때도 비슷한 어려움을 느꼈고, 결국 자료의 힌트를 참고하여 코드를 작성하게 되었습니다. 하지만 이 과정을 통해 **재귀 함수가 어떻게 동작**하는지 점차 이해할 수 있었습니다.

처음에는 **하노이의 탑 규칙을 파악**했으니 알고리즘을 금방 구현할 수 있을 것이라고 생각했지만, 생각을 코드로 옮기는 것은 다른 차원의 문제라는 점을 실감했습니다. 이를 통해 제가 아직 사고를 코드로 명확히 변환하는 데 부족함이 있다는 것을 깨달았고, 알고리즘을 제대로 구현하려면 **규칙을 철저히 이해**하는 것이 중요하다는 점을 알게 되었습니다. 특히, 알고리즘을 **시각화**하고 **각 과정의 흐름을 명확히 설계**하는 능력이 필수적임을 배웠습니다.

개발 과정에서는 여러 번의 실수를 반복했고, 그중 재귀 함수가 잘못 동작하며 스택 오버플로우와 같은 문제를 경험하기도 했습니다. 이러한 문제를 해결하는 과정에서 **재귀의 원리**를 더욱 깊이 이해하게 되었고, 문제를 **체계적으로 접근**하는 방법의 중요성 또한 다시 한번 느꼈습니다.

이번 프로젝트를 통해 제가 **부족한 점**을 명확히 알게 되었고, 앞으로 **어떤 부분을 더 보완**해야 하는지도 깨달았습니다. 앞으로도 다양한 알고리즘과 문제를 접하며 꾸준히 연습하고 배운 점들을 체득해 나가야겠다는 다짐을 하게 된 프로젝트였습니다.

감사합니다.

김유진