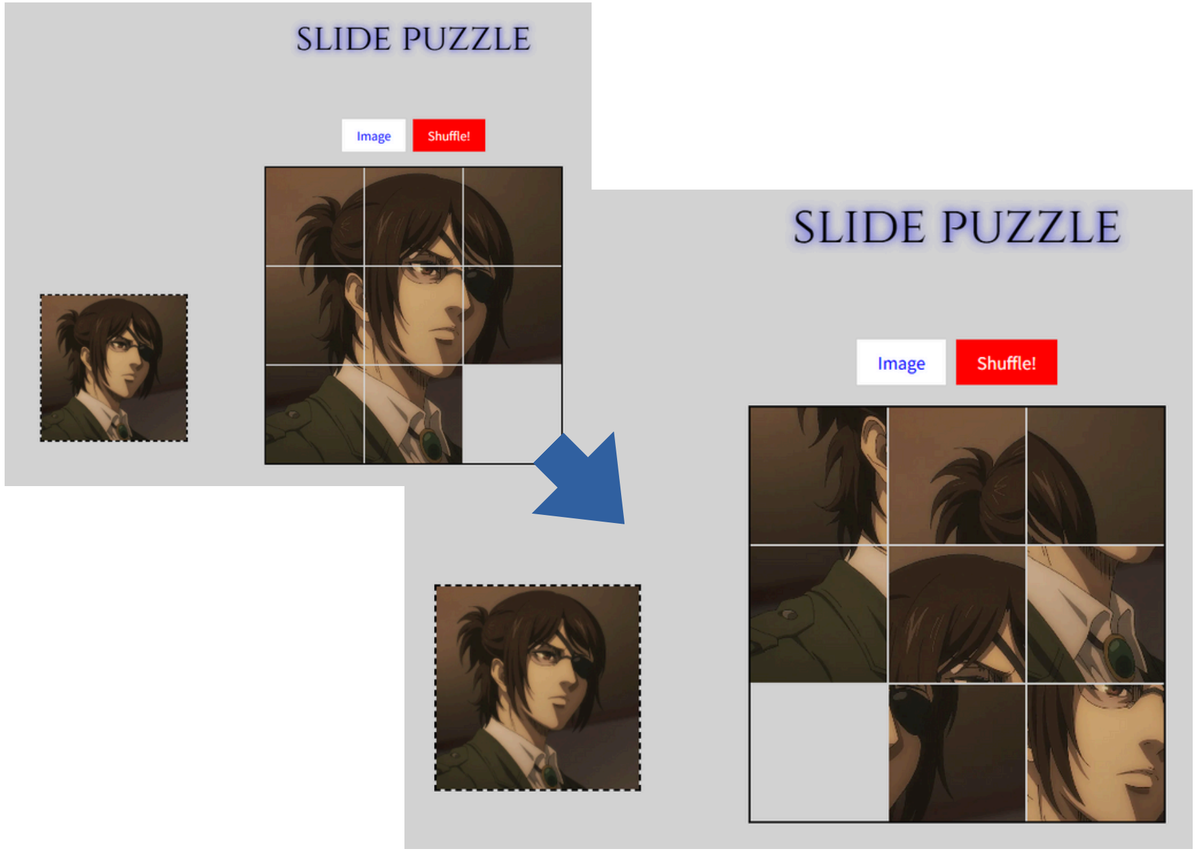
슬라이드 퍼즐 만들기

JavaScript 로 게임 구현

최준희

목차
Contents

게임 설명 gameOverview;



1. 원하는 이미지 파일 삽입
2. Shuffle 버튼으로 퍼즐 섞기
3. 퍼즐을 클릭하여 빈칸으로 이동

미리 보기 그림과
동일하게 맞추면 성공

게임 구상 gameDesign;

1

이미지를 삽입하면 퍼즐 생성

보드 안에 삽입된 이미지가
자동으로 3 x 3 형태의 조각으로 만들어짐

2

셔플하면 무작위로 섞임

Shuffle 버튼 클릭 시에
생성된 퍼즐 조각들이 임의의 위치로 이동

3

빈 칸의 주위 퍼즐만 이동 가능

빈 칸의 상하좌우에 위치한 조각들만
해당 빈칸으로 이동 가능 해야함

4

완성 시 게임 종료

원본 이미지와 동일하게 퍼즐을 맞췄을 시
alert 기능과 함께 게임 종료

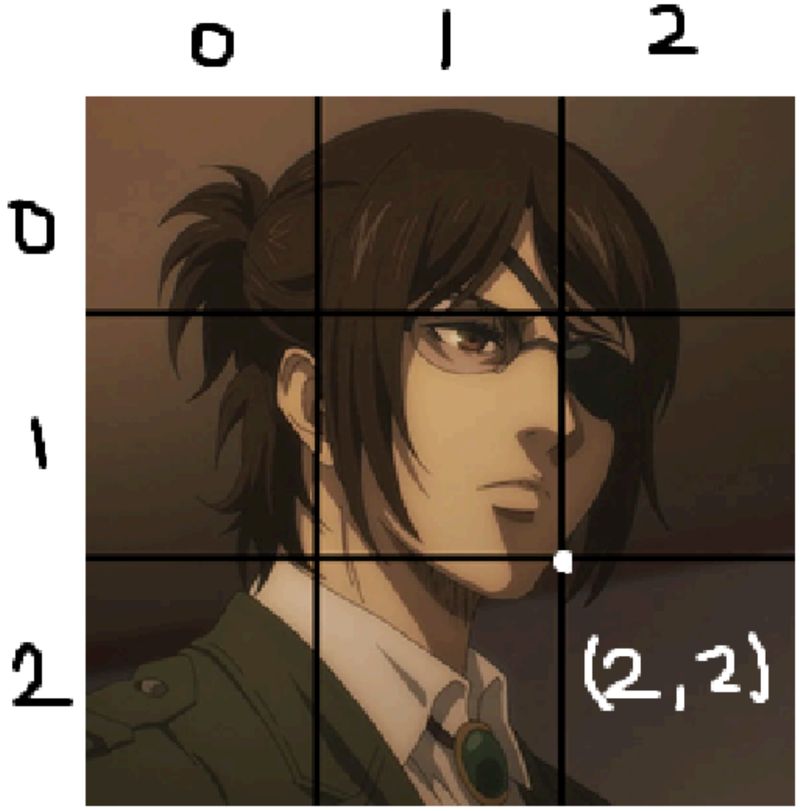
주요 기능 keyFeatures;

```
function createPuzzle() {
  puzzleContainer.innerHTML = ""; //퍼즐 컨테이너 초기화
  tiles = []; //타일 정보를 저장할 배열, 기존 타일 배열 초기화

  for (let i = 0; i < size * size; i++) {
    //3X3 즉 0~8 을 반복문
    const tile = document.createElement("div"); // 8개의 타일 div 생성
    tile.classList.add("tile"); //타일에 tile - class를 추가
    tile.dataset.index = i; //각 타일에 index 정보 추가

    if (i !== emptyIndex) {
      //3x3에서는 emptyIndex가 8
      tile.style.backgroundImage = `url(${imgSrc})`;
      const row = Math.floor(i / size); // 행 : 0,0,0,1,1,1,2,2,2
      const col = i % size; //             열 : 0,1,2,0,1,2,0,1,2
      tile.style.backgroundPosition = `${(col / (size - 1)) * 100}% ${
        (row / (size - 1)) * 100 // 해당 tile에서 보여줄 영역 설정, 추후 정답 위치
      }%`;
    }
    tile.onclick = () => moveTile(i); //누른 타일의 index값을 받아 moveTile 함수로 이동
    puzzleContainer.appendChild(tile); //부모요소.appendchild(자식요소) -DOM에 추가
    tiles.push(tile); // 모든 타일을 배열로 관리
  }
}
```

```
.tile {
  background-size: 300%;
  cursor: pointer;
}
```

1.이미지 삽입 → 나눠서 관리

반복문을 이용하여 각각의 퍼즐 조각을
담을 영역을 생성하고
추후 **타일 이동** / **정답 확인** 등의 함수를
용이하게 연계하기 위해
index 정보도 같이 입력해주며,
배열로 관리한다.
각 영역에 맞는 그림 저장은
CSS를 통해 원본 그림의 3배로 불러온 뒤
각각의 영역에 맞는 행과 열 번호를 부여하고
그 번호에 맞는 부분을 일치시켜준다.



let emptyIndex = size * size - 1;
emptyIndex 가 해당하는 영역인 (2,2)가 빈칸으로 설정된다.

```css
#puzzleContainer {
  width: 400px;
  height: 400px;
  display: grid;
  grid-template-columns: repeat(3, 1fr);
  grid-template-rows: repeat(3, 1fr);
  gap: 2px;
  border: 2px solid black;
  margin-top: 20px;
  object-fit: contain;
```

```
shuffleBtn.onclick = () => shufflePuzzle(); //셔플 버튼 클릭시 shuffle 함수 실행

function shufflePuzzle() {
  isShuffling = true;
  for (let i = 0; i < 100; i++) {
    const movable = getMovableIndexes();
    const rand = movable[Math.floor(Math.random() * movable.length)]; //movable[0~1]
    moveTile(rand);
  }
  isShuffling = false;
}
```

```
function getMovableIndexes() {
  const indexes = []; //이동 가능한 타일들의 배열
  const emptyRow = Math.floor(emptyIndex / size); // 8나누기3 몫의 내림 : 2
  const emptyCol = emptyIndex % size; // 8나누기3의 나머지 2
  if (emptyRow > 0) indexes.push(emptyIndex - size); //index : 5
  if (emptyRow < size - 1) indexes.push(emptyIndex + size);
  if (emptyCol > 0) indexes.push(emptyIndex - 1); // index : 7
  if (emptyCol < size - 1) indexes.push(emptyIndex + 1);
  return indexes;
}
```

1.Shuffle → 초기 상태에서 직접 이동

조각들을 임의로 배치하면 풀 수 없는
조합이 나올 수 있기 때문에,
직접 퍼즐을 이동해서 섞어야 한다.
현재 emptyIndex 로 주변의 움직일 수 있는
tile들의 index를 배열로 받아온다.
움직일 수 있는 위치들의 배열 중 임의로
선택하여 moveTile 함수로 보낸다.

```
function moveTile(i) {
  const targetRow = Math.floor(i / size); // 7/3의 내림 : 2
  const targetCol = i % size; // 7/3의 나머지 : 1
  const emptyRow = Math.floor(emptyIndex / size); // 8/3의 내림 : 2
  const emptyCol = emptyIndex % size; // 8/3의 나머지 : 2
  if (
    (targetRow === emptyRow && Math.abs(targetCol - emptyCol) === 1) ||
    (targetCol === emptyCol && Math.abs(targetRow - emptyRow) === 1)
  ) {
    [
      tiles[i].style.backgroundImage,
      tiles[emptyIndex].style.backgroundImage,
    ] = [
      tiles[emptyIndex].style.backgroundImage,
      tiles[i].style.backgroundImage,
    ];
    [
      tiles[i].style.backgroundPosition,
      tiles[emptyIndex].style.backgroundPosition,
    ] = [
      tiles[emptyIndex].style.backgroundPosition,
      tiles[i].style.backgroundPosition,
    ];
    emptyIndex = i; //
    if (!isShuffling && checkSolved()) {
      setTimeout(() => alert("퍼즐 완성! 🎉"), 100);
    }
  }
```

3. 이동 가능한 퍼즐 선택

shufflePuzzle 함수의 조건문으로부터 받아온 rand 변수의 index를 통해서
현재 emptyIndex와 비교하여 빈칸과 붙어 있는 퍼즐인지를 확인해야함.
(같은 행일 때 1열 차이 / 같은 열일 때 1행 차이)
사용자가 플레이 할 때에도 같은 함수를 moveTile() 이용하기 때문에, 확인하는 작업이 필요하다.

---

destructing assignment (배열이나 객체 안의 값을 쉽게 꺼내서 변수에 바로 할당하는 문법)
처음에 emptyIndex에는 이미지가 할당되지 않았으므로 Image 또한 바꿔줘야 함.

emptyIndex = i;

```
function checkSolved() {
  for (let i = 0; i < tiles.length - 1; i++) {
    // i < 7 => 첫 emptyindex(8번) 제외하고
    const row = Math.floor(i / size); // 행 : 0,0,0,1,1,1,2,2
    const col = i % size; // 열 : 0,1,2,0,1,2,0,1
    const correctPos = `${(col / (size - 1)) * 100}% ${
      (row / (size - 1)) * 100
    }%`;
    if (tiles[i].style.backgroundPosition !== correctPos) return false;
  }
  return true;
}
```

```
emptyIndex = i; //
if (!isShuffling && checkSolved()) {
  setTimeout(() => alert("퍼즐 완성! 🎉"), 100);
}
```

4. 게임 성공 판단 기준

checkSolved 함수 생성
처음 createPuzzle() 과 동일하게
첫 시작 때 index의 backgroundPosition 값이
동일해지면 return true;

moveTile() 마지막에 checkSolved()를 넣어
이동할 때 마다 확인 할 수 있도록 설정.

==> 가끔 shuffle 후에 성공 alert 가 횟수 랜덤하게
뜨는 문제가 발생.

```
  emptyIndex = i; //
  if (!isShuffling && checkSolved()) {
    setTimeout(() => alert("퍼즐 완성! 🎉"), 100);
  }
```

```
let isShuffling = false;

shuffleBtn.onclick = () => shufflePuzzle(); //셔플 버튼 클릭시 shuffle 함

function shufflePuzzle() {
  isShuffling = true;
  for (let i = 0; i < 100; i++) {
    const movable = getMovableIndexes();
    const rand = movable[Math.floor(Math.random() * movable.length)];
    moveTile(rand);
  }
  isShuffling = false;
}
```

isShuffling : boolean 변수 생성
shufflePuzzle 중에는 섞고 있다.
=> 반복문을 통해 moveTile() 갈 때에는
alert 실행할 조건이 만족 되지 못하도록
=>(isShuffling = false; + checkSolved = true;)
만족 시 alert 활성화.

결과 및 평가 projectSelfReview;

1. CSS의 grid 레이아웃 기능과 backgroundPosition을 활용하여 이미지를 분할하는 과정이 인상적이었다.

2. JavaScript로 DOM을 조작하며 이미지 조각을 섞고 이동시키는 과정을 통해 웹의 구조적 동작 원리를 더 깊이 이해할 수 있었다.

3. 게임을 만드는 과정에서 destructuring assignment 라는 새로운 문법을 통해 객체나 배열의 값을 간결하게 꺼내거나 서로 교환할 수 있는 방법을 배웠습니다.

감사합니다