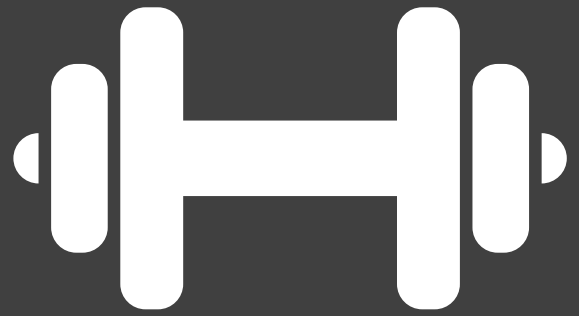
한빛미디어 HanbitMedia, Inc.

Bsidesoft co.

OOP DESIGN WITH GAME

1 2 3 4 5 6

객체지향 준비운동



객체 지향 vs ???

객체지향을 이해하는 것이 어렵다!
반대되는 개념을 살펴보자.
객체지향의 반대 개념은?

객체지향 vs 프로시저지향

객체 지향 vs ???

객체지향을 이해하는 것이 어렵다!
반대되는 개념을 살펴보자.
객체지향의 반대 개념은?

~~객체지향 vs 프로시저지향~~

객체지향 vs 값지향

객체지향

참조 값을 사용함.
- 한 번 만든 객체가 전파됨
- 데이터를 처리하는 메소드를 내장함
- 모든 메소드는 현재 객체 컨텍스트 사용

각 객체는 단일한 책임을 갖음
- 자신이 처리할 수 없는 책임은 타 객체와 협력
- 객체망에 참여하여 자신의 역할 수행함

객체지향 사고
1. 객체망으로 문제를 해결한다.
2. 단일책임을 준수한다.
3. 객체를 이용하여 문제를 해결한다.
4. 은닉과 캡슐화를 활용하여 상태를 처리한다.

값지향

값의 복사를 사용함.
- 매번 새로운 값을 만들어 냄
- 값을 처리하는 유틸리티 함수가 존재
- 함수가 값을 처리하는 경우 컨텍스트는 없음

값은 순수한 데이터로 책임의 개념이 없음
- 각 역할에 맞는 적절한 값이 존재함
- 값을 처리하는 함수의 조합으로 문제를 해결함

값 지향 사고
1. 연산을 통해 문제를 해결한다.
2. 연산에 적합한 값을 정의한다(함수포함)
3. 값을 통해 문제를 해결한다.
4. 순수함수를 활용하여 상태를 제거한다.

객체지향에서 단일 책임의 도출 기준

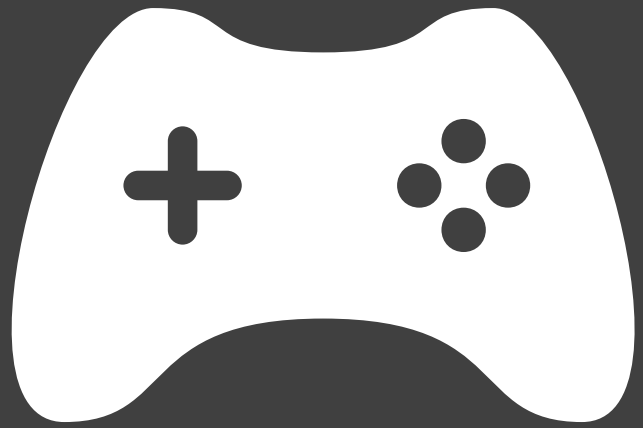
객체디자인 또는 설계

객체의 책임을 도출하고 객체망을 구성하는 것

책임(또는 역할)을 도출하는 기준으로 사용할 수 있는 후보군

1. 도메인 - 의존성
2. 네이티브영역 - 가변성
3. 변화율 - 유지보수

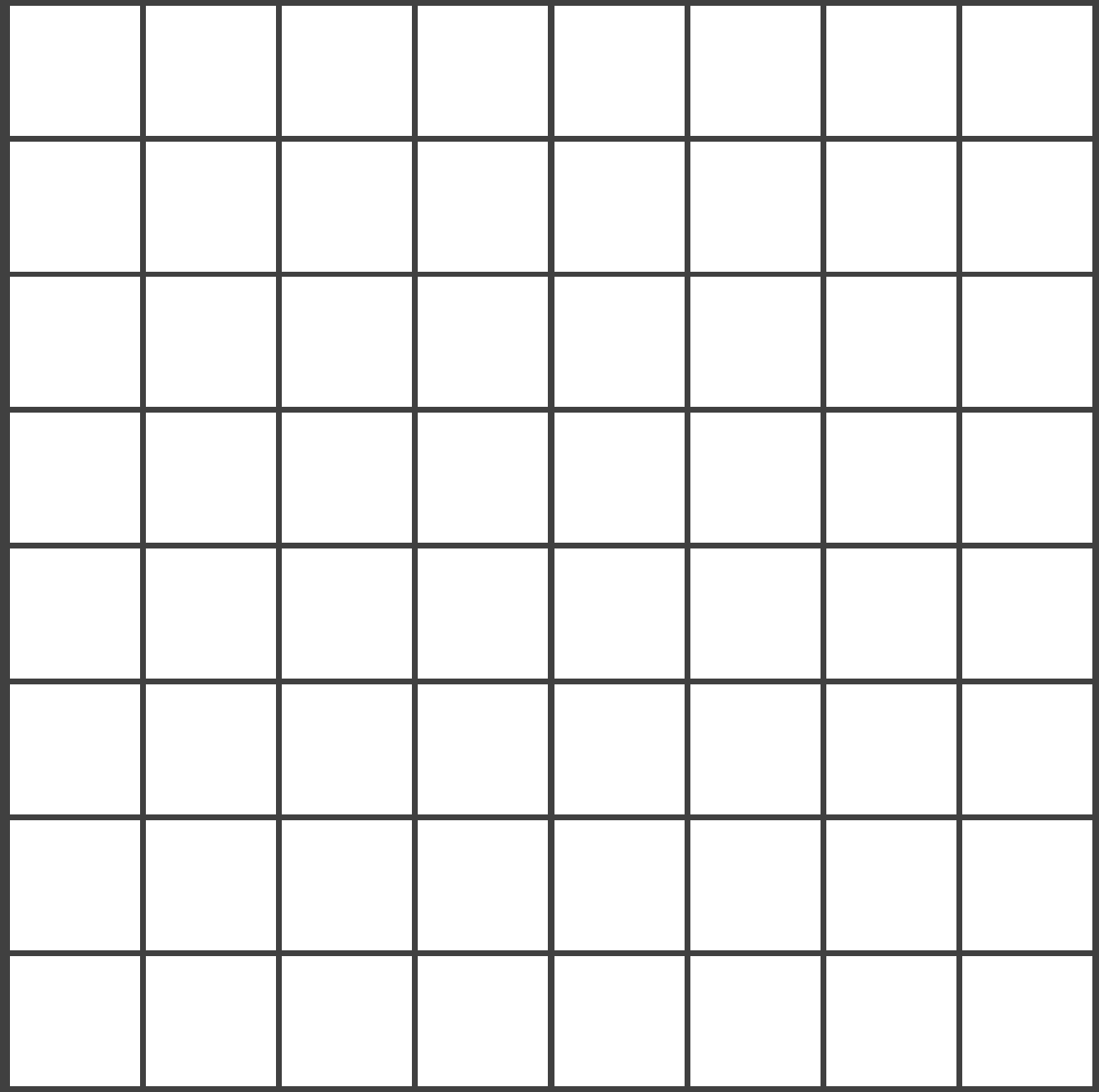
게임 개요 및 규칙

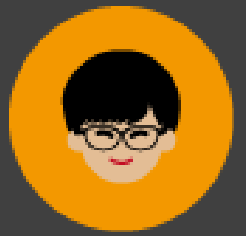


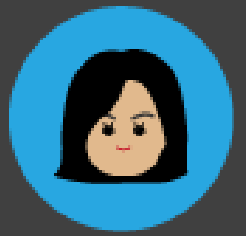
출처
http://www.dreamwiz.com/VIEW/NEWS/AWVIa13h9sStCcB3XgDp

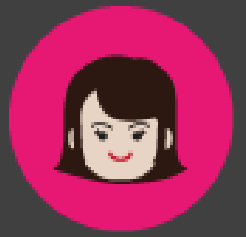
출처
http://www.dreamwiz.com/VIEW/NEWS/AWVIa13h9sStCcB3XgDp

type : 0, 1, 2, 3, 4
cell size : 8 x 8



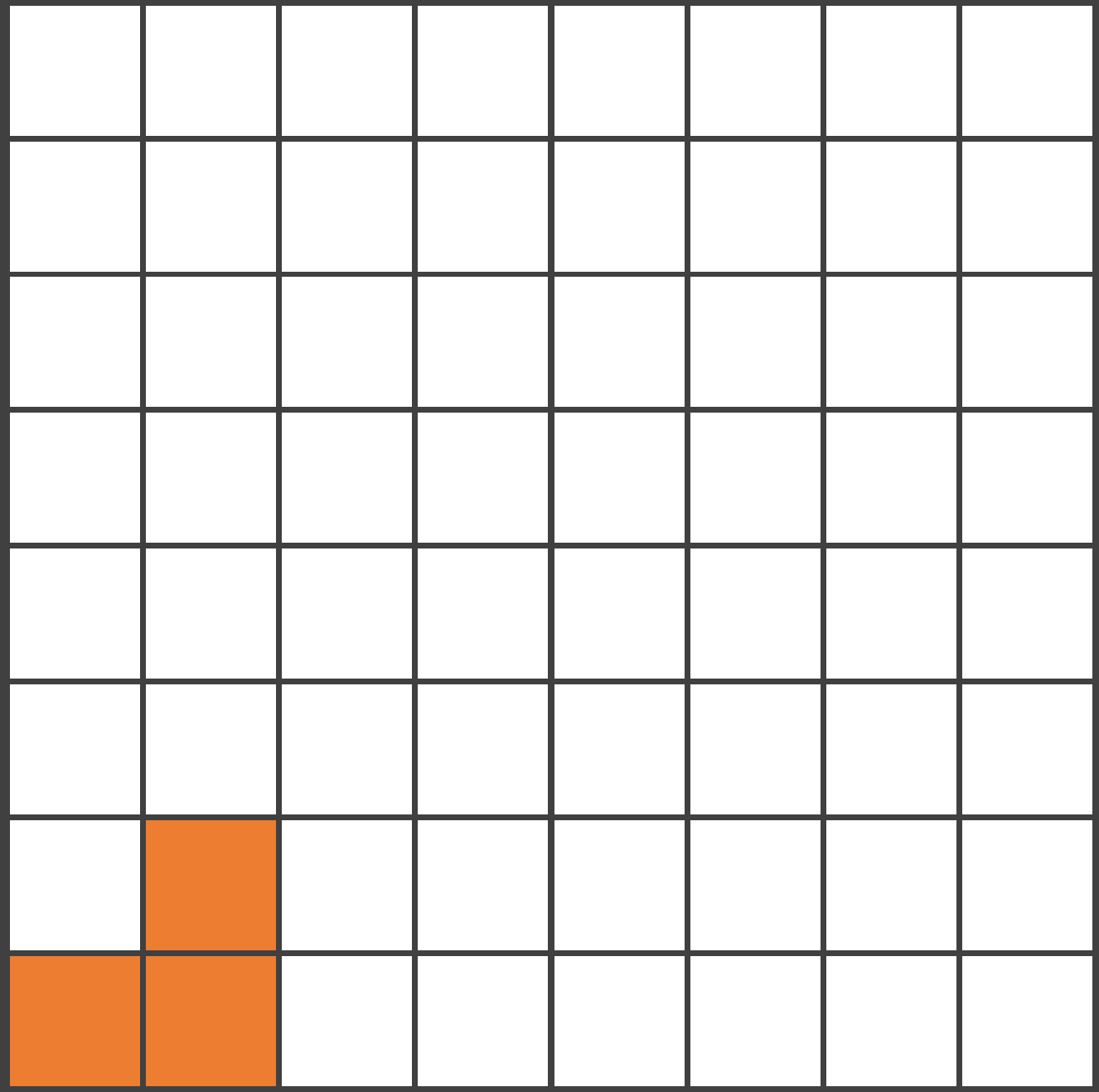


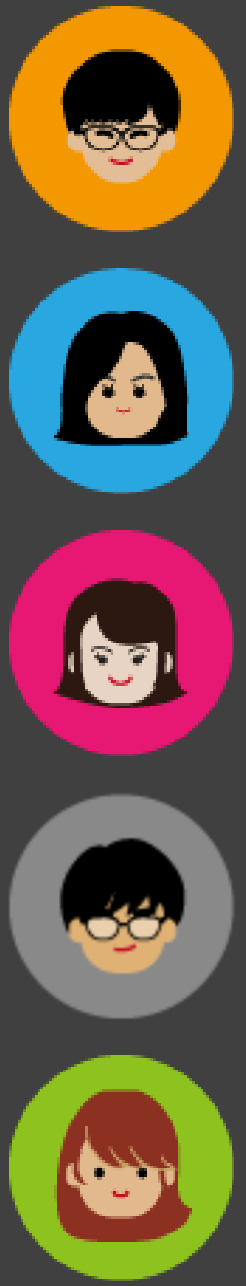




type : 0, 1, 2, 3, 4
cell size : 8 x 8

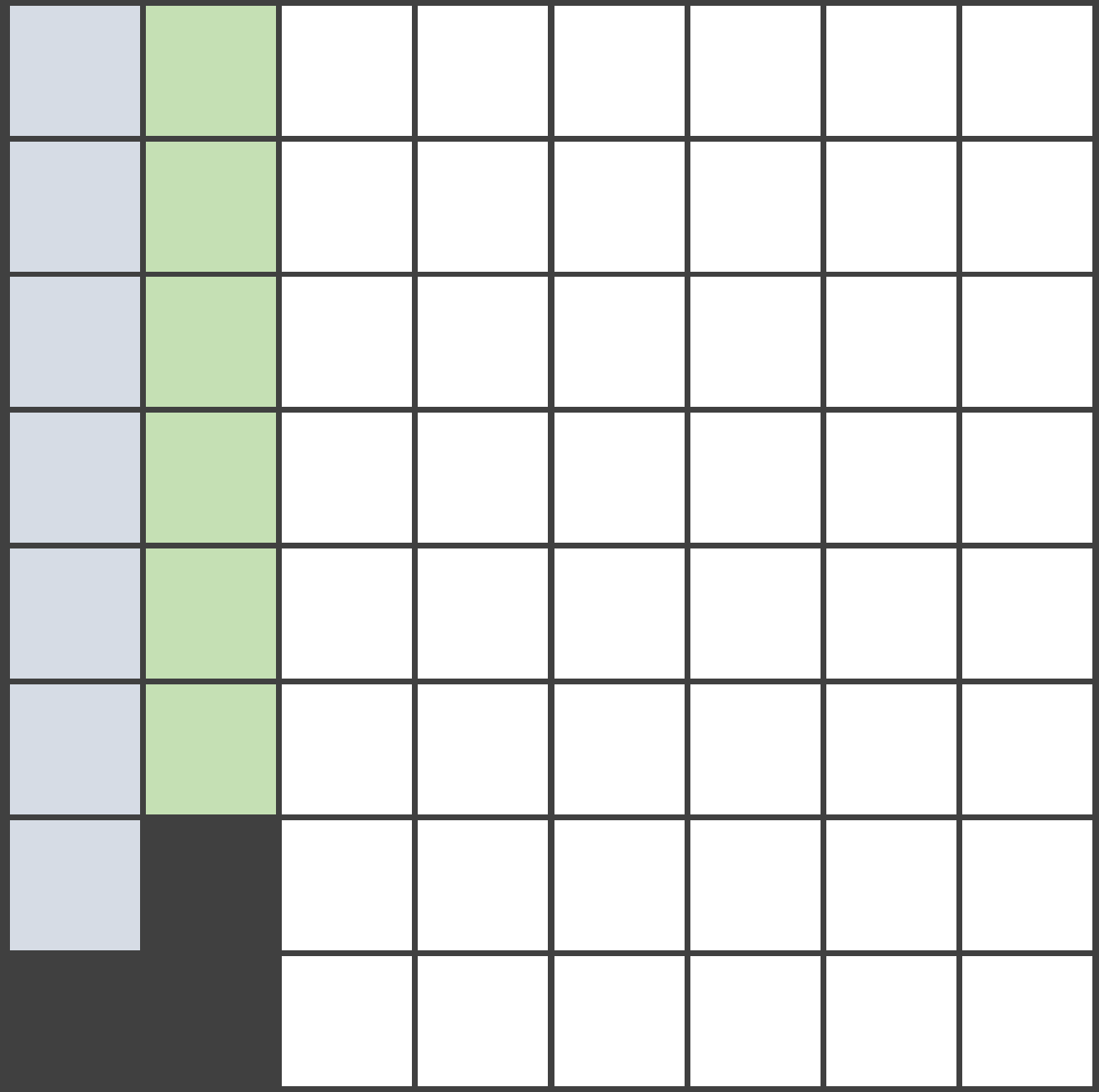
한 번에 3개 이상 같은 색으로 인접한 블록이 선택되면 삭제됨

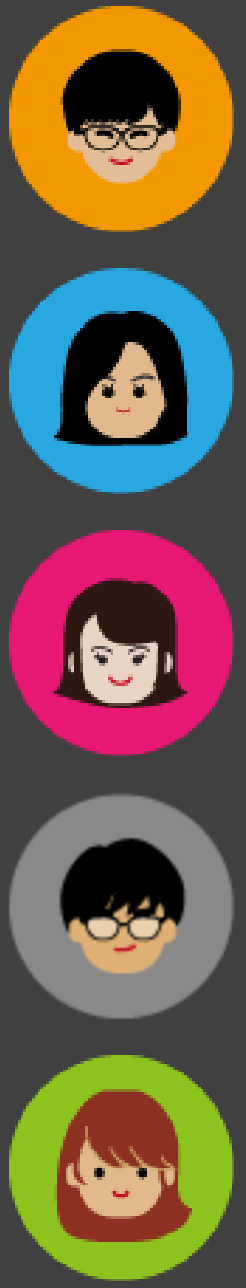




type : 0, 1, 2, 3, 4
cell size : 8 x 8

한 번에 3개 이상 같은 색으로 인접한 블록이 선택되면 삭제됨

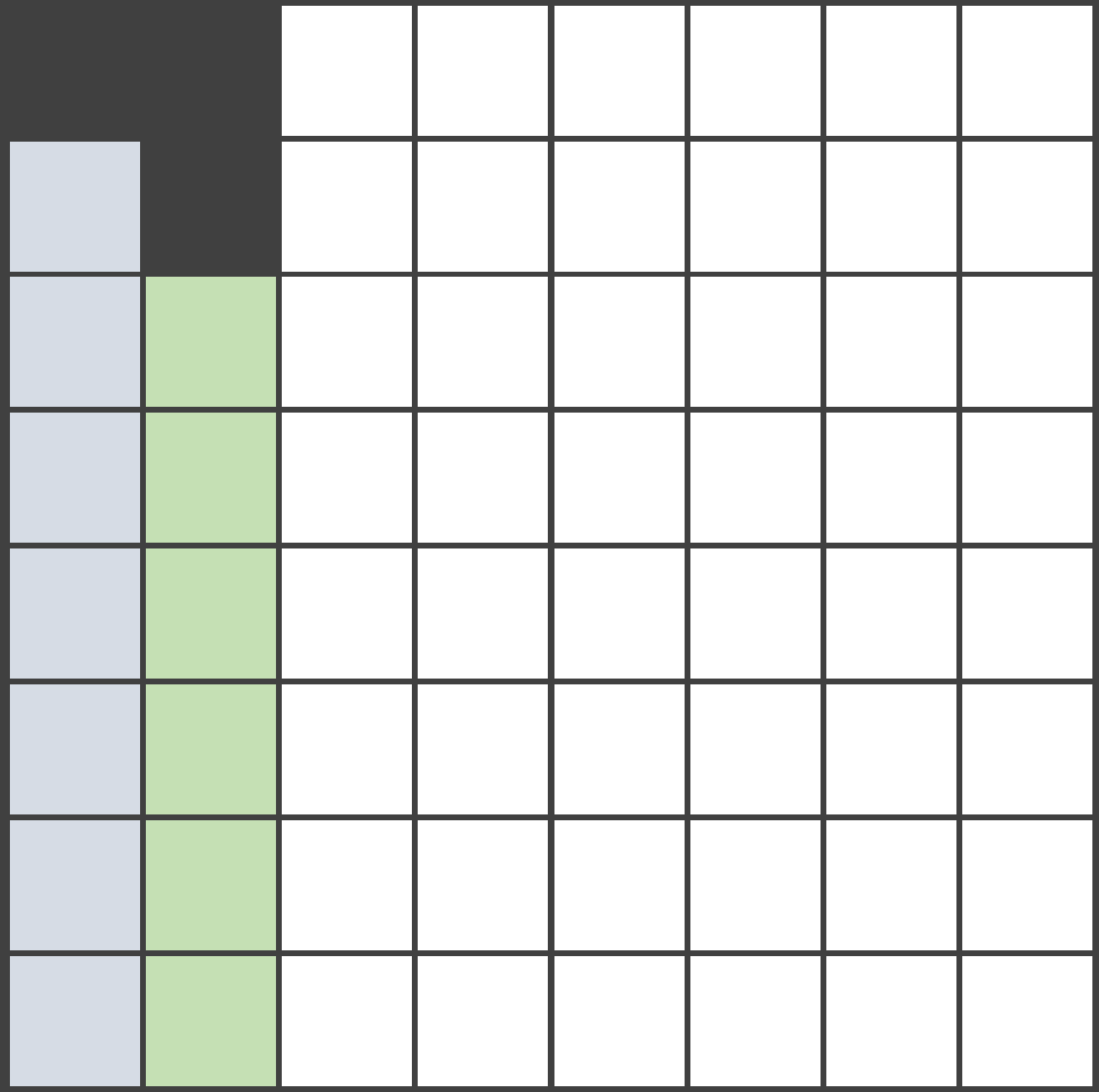


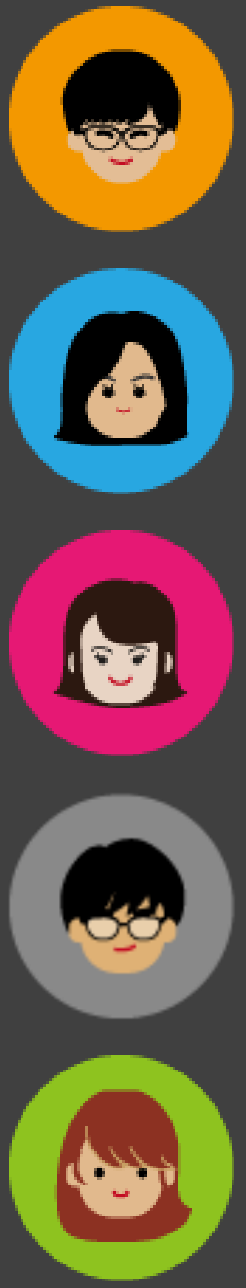


type : 0, 1, 2, 3, 4
cell size : 8 x 8

한 번에 3개 이상 같은 색으로 인접한 블록이 선택되면 삭제됨

삭제되면 위의 블록이 내려옴



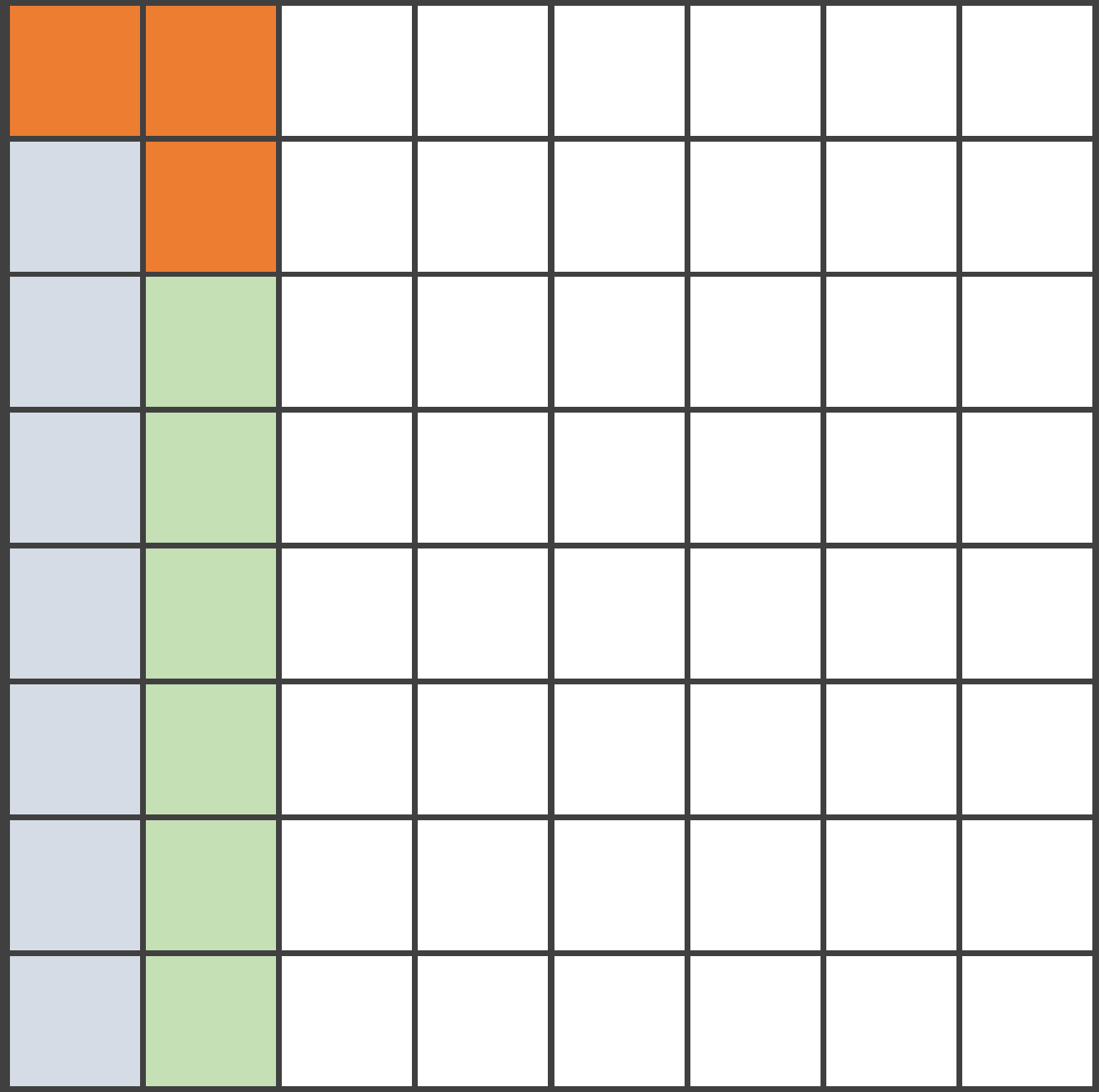


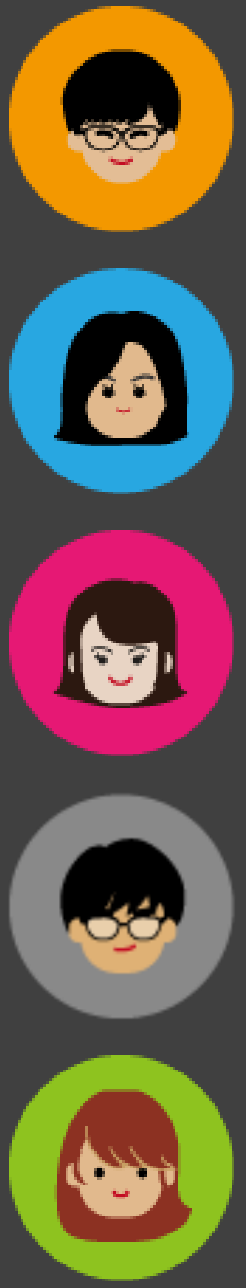
type : 0, 1, 2, 3, 4
cell size : 8 x 8

한 번에 3개 이상 같은 색으로 인접한 블록이 선택되면 삭제됨

삭제되면 위의 블록이 내려옴

내려온 뒤 빈 공간의 블록이 생성되어 채워짐





어디부터 시작할까?

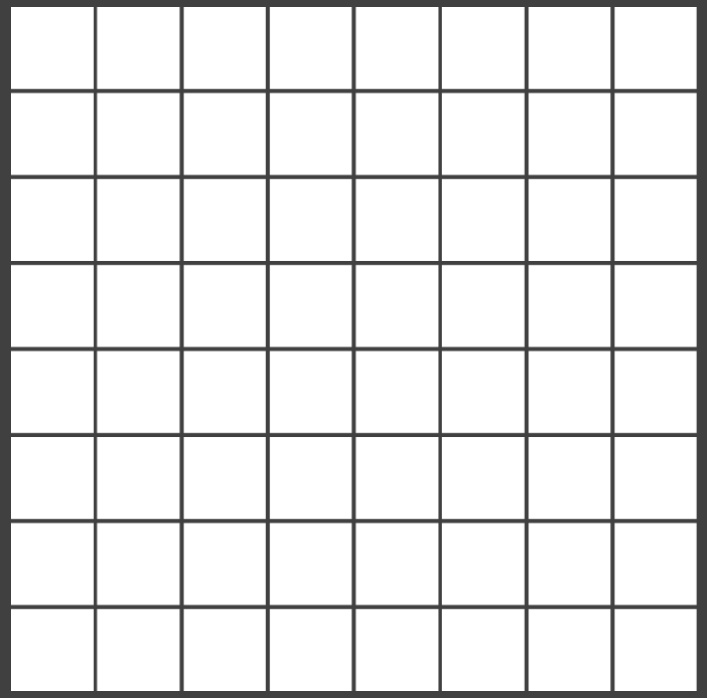
사람이 가장 먼저 파악할 수 있는 것

표면적인 것과 시각적인 것

프로그램의 핵심은 데이터

프로그램이 성립하기 위한 자료구조를 파악하자

스테이지 : 설정에 따라 자유롭게 크기를 변경할 수 있음(스칼라값)
블록타입 : 미리 정해진 블록 타입이 존재(정의값)
블록 : 생성되거나 삭제되고, 연결되거나 조건을 파악함(객체값)



```
stageWidth = 8;
stageHeight = 8;
blockTypes = [0, 1, 2, 3, 4];
Block = class{
  constructor(){}
  isNext(){}
  isValid(){}
  ..
}
```

```
const Block = class{ //블록은 여러개 생성되므로 클래스!
    //자신만의 타입을 갖는다.
    //타입에 따른 이미지 경로를 반환한다.
};
```

```
const Block = class{ //블록은 여러개 생성되므로 클래스!
   //자신만의 타입을 갖는다.
   //타입에 따른 이미지 경로를 반환한다.
};

const Block = class{
  constructor(type){
    this._type = type;
  }
  get image(){return `url('img/${this._type}.png')`;}
  get type(){return this._type;}
}
Block.GET = (type = parseInt(Math.random() * 5))=>new Block(type);
```

```
const Game = { //게임은 본체이므로 한 개만 있으면 되기 때문에 싱글톤 오브젝트로 생성
  //초기화 - 필요한 정보를 바탕으로 게임본체를 생성한다.
  //렌더 - 그림을 갱신함
  //이벤트 관련 - 각 블록에서 이벤트를 처리함
};
```

```
const Game = { //게임은 본체이므로 한 개만 있으면 되기 때문에 싱글톤 오브젝트로 생성
  //초기화 - 필요한 정보를 바탕으로 게임본체를 생성한다.
  //렌더 - 그림을 갱신함
  //이벤트 관련 - 각 블록에서 이벤트를 처리함
};
```

함수만 하나 노출하면 됨

내부에서만 사용

```
const Game = { //게임은 본체이므로 한 개만 있으면 되기 때문에 싱글톤 오브젝트로 생성
  //초기화 - 필요한 정보를 바탕으로 게임본체를 생성한다.
  //렌더 - 그림을 갱신함
  //이벤트 관련 - 각 블록에서 이벤트를 처리함
};

const Game =(_=>{
  const init = ...
  ...
  return init;
})();
```

함수만 하나 노출하면 됨

내부에서만 사용

```
const Game =(_=>{
  const column = 8, row = 8, blockSize = 80;
  const data = [];
  let table;
  const init = tid=>{
    table = document.querySelector(tid);
    for(let i = 0; i < row; i++){
      const r = [];
      data.push(r);
      for(let j = 0; j < column; j++) r[j] = Block.GET();
    }
    render();
  };

  const render =_=>{..}

})();
```

```
const column = 8, row = 8, blockSize = 80;
const data = [];
let table;

const el = tag=>document.createElement(tag)
const render =_=>{
  table.innerHTML = '';
  data.forEach(row=>table.appendChild(row.reduce((tr, block)=>{
    tr.appendChild(el('td')).style.cssText = `
      ${block ? `background:${block.image};` : ''}
      width:${blockSize}px;
      height:${blockSize}px;
      cursor:pointer`;
    return tr;
  }, el('tr'))));
};
```

```
const Block = class{
  constructor(type){
    this._type = type;
  }
  get image(){return `url('img/${this._type}.png')`;}
  get type(){return this._type;}
}
Block.GET = (type = parseInt(Math.random() * 5))=>new Block(type);
```

```
const Game =(_=>{
  const column = 8, row = 8, blockSize = 80;
  const data = [];
  let table;
  const init = tid=>{
    table = document.querySelector(tid);
    for(let i = 0; i < row; i++){
      const r = [];
      data.push(r);
      for(let j = 0; j < column; j++) r[j] = Block.GET();
    }
    //테이블에 이벤트를 건다
    render();
  };
})();
```

```
const Game =(_=>{
  const column = 8, row = 8, blockSize = 80;
  const data = [];
  let table;
  const init = tid=>{
    table = document.querySelector(tid);
    for(let i = 0; i < row; i++){
      const r = [];
      data.push(r);
      for(let j = 0; j < column; j++) r[j] = Block.GET();
    }
    //테이블에 이벤트를 건다
    table.addEventListener('mousedown', down);
    table.addEventListener('mouseup', up);
    table.addEventListener('mouseleave', up);
    table.addEventListener('mousemove', move);
    render();
  };
})();
```

```
const down = e=>{
  //down된 상태를 활성
  //x, y로 부터 block데이터를 얻음
  //위에서 얻은 블록을 시작블록, 현재블록으로 설정하고 선택목록에 포함시킴
}
```

```
const down = e=>{
  //down된 상태를 활성
  //x, y로 부터 block데이터를 얻음
  //위에서 얻은 블록을 시작블록, 현재블록으로 설정하고 선택목록에 포함시킴
}

const column = 8, row = 8, blockSize = 80;
const data = [];
let table;
let startBlock, currBlock, isDown;
const selected = [], getBlock = (x, y)=>{...};

const down = ({pageX:x, pageY:y})=>{
  if(isDown) return;
  const curr = getBlock(x, y);
  if(!curr) return;
  isDown = true;
  selected.length = 0;
  selected[0] = startBlock = currBlock = curr;
  render();
};
```

```
const down = e=>{
  //down된 상태를 활성
  //x, y로 부터 block데이터를 얻음
  //위에서 얻은 블록을 시작블록, 현재블록으로 설정하고 선택목록에 포함시킴
}

const column = 8, row = 8, blockSize = 80;
const data = [];
let table;
const startBlock, currBlock, selected = [], isDown;
const getBlock = (x, y)=>{
  const {top:T, left:L} = table.getBoundingClientRect();
  if(x < L || x > (L + blockSize * row) || y < T || y > (T + blockSize * column) return null;
  return data[parseInt((y - T) / blockSize)][parseInt((x - L) / blockSize)];
};
const down = ({pageX:x, pageY:y})=>{
  if(isDown) return;
  const curr = getBlock(x, y);
  if(!curr) return;
  isDown = true;
  selected.length = 0;
  selected[0] = startBlock = currBlock = curr;
  render();
};
```

```
const move = e=>{
  //down이 아니라면 이탈
  //x, y 위치의 블록을 얻음
  //블록의 타입이 같고 인접되어있는지 검사
  //위에서 얻은 블록이 선택목록에 없으면 추가
  //있다면 전전 블록일 경우 하나 삭제
}
```

```
const move = e=>{
  //down이 아니라면 이탈
  //x, y 위치의 블록을 얻음
  //블록의 타입이 같고 인접되어있는지 검사
  //위에서 얻은 블록이 선택목록에 없으면 추가
  //있다면 전전 블록일 경우 하나 삭제
}

const move =({pageX:x, pageY:y})=>{
  if(!isDown) return;
  const curr = getBlock(x, y);
  if(!curr || curr.type != startBlock.type || !isNext(curr)) return;
  if(selected.indexOf(curr) == -1) selected.push(curr);
  else if(selected[selected.length - 2] == curr) selected.pop();
  currBlock = curr;
  render();
};
```

```
const isNext = curr=>{
  let r0, c0, r1, c1, cnt = 0;
  data.some((row, i)=>{
    let j;
    if((j = row.indexOf(currBlock)) != -1) r0 = i, c0 = j, cnt++;
    if((j = row.indexOf(curr)) != -1) r1 = i, c1 = j, cnt++;
    return cnt == 2;
  });
  return curr != currBlock && Math.abs(r0 - r1) == 1 || Math.abs(c0 - c1) == 1;
};
```

```
const up = e=>{
  //down을 해제
  //선택목록이 3이상이면 삭제 실시
  //2이하면 리셋
}
```

```
const up = e=>{
  //down을 해제
  //선택목록이 3이상이면 삭제 실시
  //2이하면 리셋
}

const up =_=>selected.length > 2 ? remove() : reset();
```

```
const up = e=>{
  //down을 해제
  //선택목록이 3이상이면 삭제 실시
  //2이하면 리셋
}

const up =_=>selected.length > 2 ? remove() : reset();

const reset =_=>{
  startBlock = currBlock = null;
  selected.length = 0;
  isDown = false;
  render();
};
```

```
const remove =_=>{
  data.forEach(r=>{ //데이터삭제
    selected.forEach(v=>{
      let i;
      if((i = r.indexOf(v)) != -1) r[i] = null;
    });
  });
  render();
  setTimeout(drop, 300);
};
```

```
const drop =_=>{
  let isNext = false;
  for(let j = 0; j < column; j++){
    for(let i = row - 1; i > -1; i--){
      if(!data[i][j] && i){
        let k = i, isEmpty = true;
        while(k--) if(data[k][j]){
          isEmpty = false;
          break;
        }
        if(isEmpty) break;
        isNext = true;
        while(i--){
          data[i + 1][j] = data[i][j];
          data[i][j] = null;
        }
        break;
      }
    }
  }
  render();
  isNext ? setTimeout(drop, 300) : readyToFill();
};
```

```
const fills = [];
let fillCnt = 0;
const readyToFill =_=>{
  fills.length = 0;
  data.some(row=>{
    if(row.indexOf(null) == -1) return true;
    const r = [...row].fill(null);
    fills.push(r);
    row.forEach((v, i)=>!v && (r[i] = Block.GET()));
  });
  fillCnt = 0;
  setTimeout(fill, 300);
};
```

```
const fill =_=>{
  if(fillCnt > fills.length){
    isDown = false;
    return;
  }
  for(let i = 0; i < fillCnt; i++){
    fills[fills.length - i - 1].forEach((v, j)=>{
      if(v) data[fillCnt - i - 1][j] = v;
    });
  }
  fillCnt++;
  render();
  setTimeout(fill, 300);
};
```

PRACTICE #1

rende시점에 매번 tr, td를 생성하지 않고
style만 업데이트하도록 수정하시오.

PRACTICE #2

선택한 블록은 배경에 노란색이 들어오도록 수정하시오.