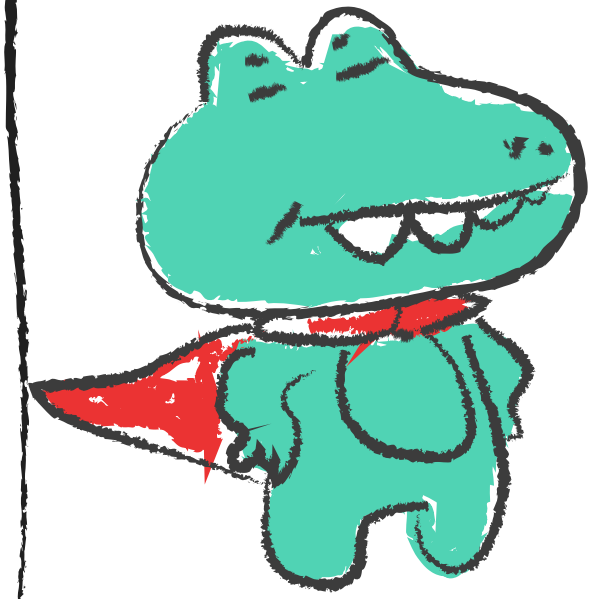
이번에 소개할 게임은



가위바위보

국민 게임이죠~

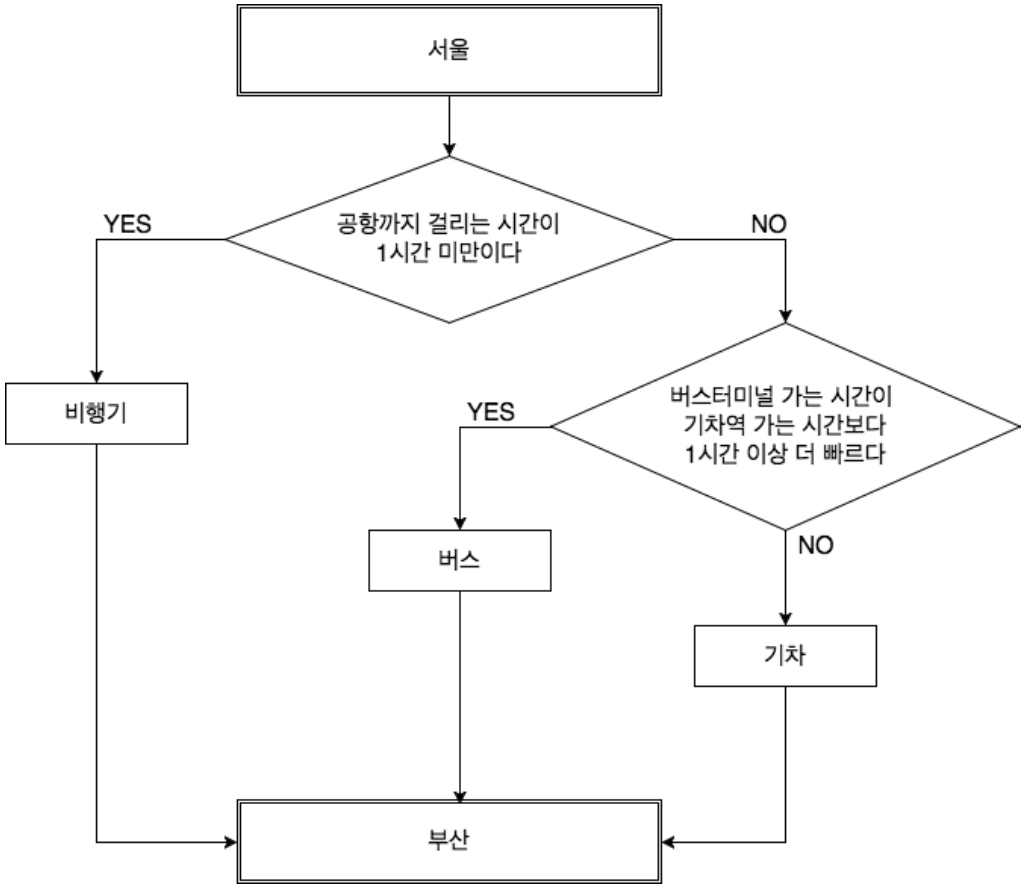
알고리즘

다른 말로 순서도 라고도 해!

1. 상대의 가위/바위/보를 랜덤하게 생성한다.
2. 사용자에게 가위/바위/보를 입력받는다.
3. 상대와 사용자에게 입력받은 가위/바위/보를 비교하여 Win, Lose, Draw를 결정한다!

* 가위 : -1, 바위 : 0, 보 : 1 *

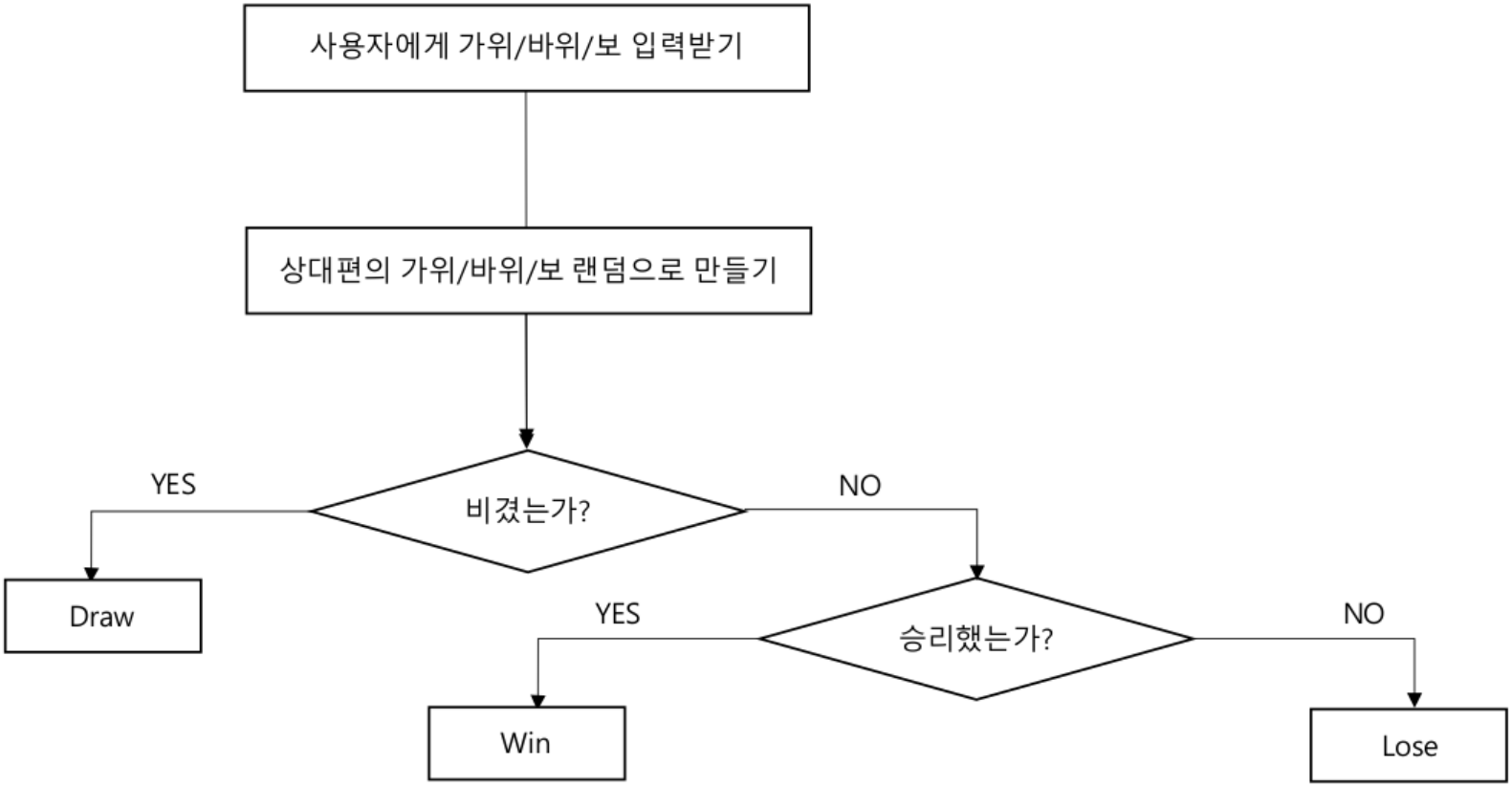
알고리즘


서울
공항까지 걸리는 시간이
1시간 미만이다
YES
NO
비행기
버스터미널 가는 시간이
기차역 가는 시간보다
1시간 이상 더 빠르다
YES
NO
버스
기차
부산

코딩할 때는 이런식
으로 그려서 표현해

알고리즘

가위바위보 흐름도!



가위바위보 코드 살펴보기

이렇게 초록색 칸이 있는 부분의 코드를 채워 넣으면 됩니다!

가위바위 코드 살펴보기

```
const img = document.querySelector("#answer");
```

id가 answer인 요소를 img에 할당

```
const select = [undefined]; // select라는 배열에는 [undefined] or [-1] or [0] or [1]만 들어가게 된다.
const scissorAnswer = -1; // -1은 가위를 의미
const rockAnswer = 0; // 0은 바위를 의미
const paperAnswer = 1; // 1은 보자기를 의미
```

필요한 변수 선언부

값을 지정하여 변수 선언

가위바위 코드 살펴보기

```
const rotate = () => { // 상대방의 카드를 뒤집는다.
    let elem = document.querySelector("#counter");
    if (elem.style.transform == "rotateY(180deg)") {
        elem.style.transform = "rotateY(0deg)";
    } else {
        elem.style.transform = "rotateY(180deg)";
    }
}
```

상대방의 카드를 뒤집는 함수

뒤집힌 상태라면 원래대로 돌려놓고 아니라면(초기상태라면) 뒤집기

```
const scissor = document.querySelector("#가위");
const rock = document.querySelector("#바위");
const paper = document.querySelector("#보");
const countdown = document.querySelector(".countdown");
```

id와 class로 요소를 찾아서 변수에 할당

가위바위 코드 살펴보기

```
const scissorSelect = () => {
    rock.style.width = "170px";
    rock.style.height = "255px";
    scissor.style.width = "204px";
    scissor.style.height = "306px";
    paper.style.width = "170px";
    paper.style.height = "255px";
    select[0] = scissorAnswer; // 0번째 인덱스에 scissorAnswer, 즉 -1을 대입 -> select = [-1]
}

const rockSelect = ()  =>{
    rock.style.width = "204px";
    rock.style.height = "306px";
    scissor.style.width = "170px";
    scissor.style.height = "255px";
    paper.style.width = "170px";
    paper.style.height = "255px";
    select[0] = rockAnswer; // 0번째 인덱스에 rockAnswer, 즉 0을 대입 -> select = [0]
}

const paperSelect = () => {
    rock.style.width = "170px";
    rock.style.height = "255px";
    scissor.style.width = "170px";
    scissor.style.height = "255px";
    paper.style.width = "204px";
    paper.style.height = "306px";
    select[0] = paperAnswer; // 0번째 인덱스에 paperAnswer, 즉 1을 대입 -> select = [1]
}
```

카드를 선택하면 해당 카드의 스타일을 변경하는 함수

가위바위 코드 살펴보기

```
const countdownSubmit = () => {
    if (countdown.innerHTML == "Ready" && !(select[0] == undefined)) { // 만약 텍스트가 Ready로 되어있으며, select 배열이 [undenified]가
        countdown.innerHTML = "3";
        setTimeout(() => { // 1초 후 Ready 위치에 있는 text를 2로 변경
            countdown.innerHTML = "2";
        }, 1000);
        setTimeout(() => { // 2초 후 Ready 위치에 있는 text를 1로 변경
            countdown.innerHTML = "1";
        }, 2000);
        setTimeout(() => { // 3초 후 Ready 위치에 있는 text를 0으로 변경
            countdown.innerHTML = "0";
        }, 3000);
        setTimeout(() => { // 4초 후 Ready 위치에 있는 text를 Win, Draw, Lose 중 하나로 변경
            rotate(); // 상대편의 카드를 뒤집는 함수 실행
            countdown.innerHTML = start();
        }, 4000);
        setTimeout(() => { // 9초 후 원상복귀
            rotate(); // 상대편의 카드를 뒤집는 함수 실행(원상복귀)
            countdown.innerHTML = "Ready"; // 다시 Ready로 변경
        }, 9000);
    } else if(select[0] == undefined) { // 가위바위보가 선택되지 않으면 select 배열은 초기 설정대로 [undefined]일 것, 따라서 가위바위보를 선
        alert("가위바위보를 선택하세요!")
    }
}
```

Ready 텍스트를 클릭하면

카운트다운을 시작하고 원상복귀

선택되지 않으면 경고 메세지

가위바위 코드 살펴보기

```
scissor.addEventListener("click", scissorSelect); // 가위 카드 클릭시 scissorSelect 함수 실행하는 이벤트 등록
rock.addEventListener("click", rockSelect); // 바위 카드 클릭시 scissorSelect 함수 실행하는 이벤트 등록
paper.addEventListener("click", paperSelect); // 보자기 카드 클릭시 scissorSelect 함수 실행하는 이벤트 등록
countdown.addEventListener("click", countdownSubmit); // Ready text 클릭시 countdownSubmit 함수 실행하는 이벤트 등록
```

HTML 요소에 이벤트 등록하는 함수

각 요소들을 클릭하면 등록된 함수 실행

```
const numToDetail = (num) => { // -1, 0, 1을 가위, 바위, 보자기로 변환해주는 함수
    return num == -1 ? "가위"
           : num == 0 ? "바위"
           : num == 1 ? "보자기"
           : "error : " + num;
}
```

숫자(num)를 가위, 바위, 보자기로 변환해주는 함수

숫자(num)의 값에 따라 적절한 문자열 반환

가위바위 코드 살펴보기

```
const start = () => { // 메인함수로, 상대방의 패를 결정하고,
    let answer = randomNumber(-1, 1);
    changeIMG(answer);
    console.log("상대방 :", numToDetail(answer))
    let submit = select[0];
    console.log("사용자 :", numToDetail(submit))
    return judge(answer, submit);
};

const changeIMG = (answer) => { // randomNumber로부터 정해진 answer, 즉 상대편의 패에 따라 이미지를 변경한다.
    if (answer == -1) {
        img.src = "img/가위.png";
    } else if (answer == 0) {
        img.src = "img/주먹.png";
    } else {
        img.src = "img/보자기.png";
    }
};
```

게임의 메인 함수 randomNumber(-1,1)을 호출하여 -1, 0, 1 중 랜덤한 숫자를 answer에 할당

answer와 submit 값을 이용하여 judge 함수 호출하고 게임의 결과 판정 후 반환

answer값(상대편의 패)에 따라 이미지 변경하는 함수

-1, 0, 1을 각각 가위, 바위, 보자기 이미지로 표시

가위바위 코드 살펴보기

```
const randomNumber = (n, m) => { // n부터 m까지의 랜덤한 숫자를 만드는 난수 발생 함수를 만드세요! return 타입은 INT입니다!
    /
     * n에는 시작 값이 들어오고, m에는 끝 값이 들어오게 되며
     * n~m까지의 난수를 발생시키는 random 함수를 만드세요! (Tip. 2일차 JavaScript 기초 파일의 27페이지를 확인해보세요!)
     * return 타입은 정수 타입인 INT입니다!
     */
};

const judge = (answer, submit) => { // 결과를 "Draw", "Lose", "Win" 형태로 도출하여 String의 형태로 return하세요!
    /
     * 가위 = -1, 바위 = 0, 보자기 = 1 입니다.
     * answer에는 상대편 카드로는 -1, 0, 1 중 하나의 값이 들어옵니다.
     * submit에는 자신이 선택한 카드에 따라 -1, 0, 1 중 하나의 값이 들어옵니다.
     * answser와 submit의 값을 비교하여 자신이 비겼는지, 졌는지, 이겼는지 판단하는 함수를 만드세요!
     * return 타입은 문자열인 String이며, "Draw", "Lose", "Win" 중 하나로 return되게 하면 됩니다!
     */
};
```

n~m의 범위로 랜덤한 실수를 생성하여 정수로 반환하는 함수

난수 발생시킬 때 범위 조절 주의!

게임의 결과 판단하는 함수

각 조건에 따라 값 반환

이번에 소개할 게임은

미로

미로를 빠져나가게 도와주세요!

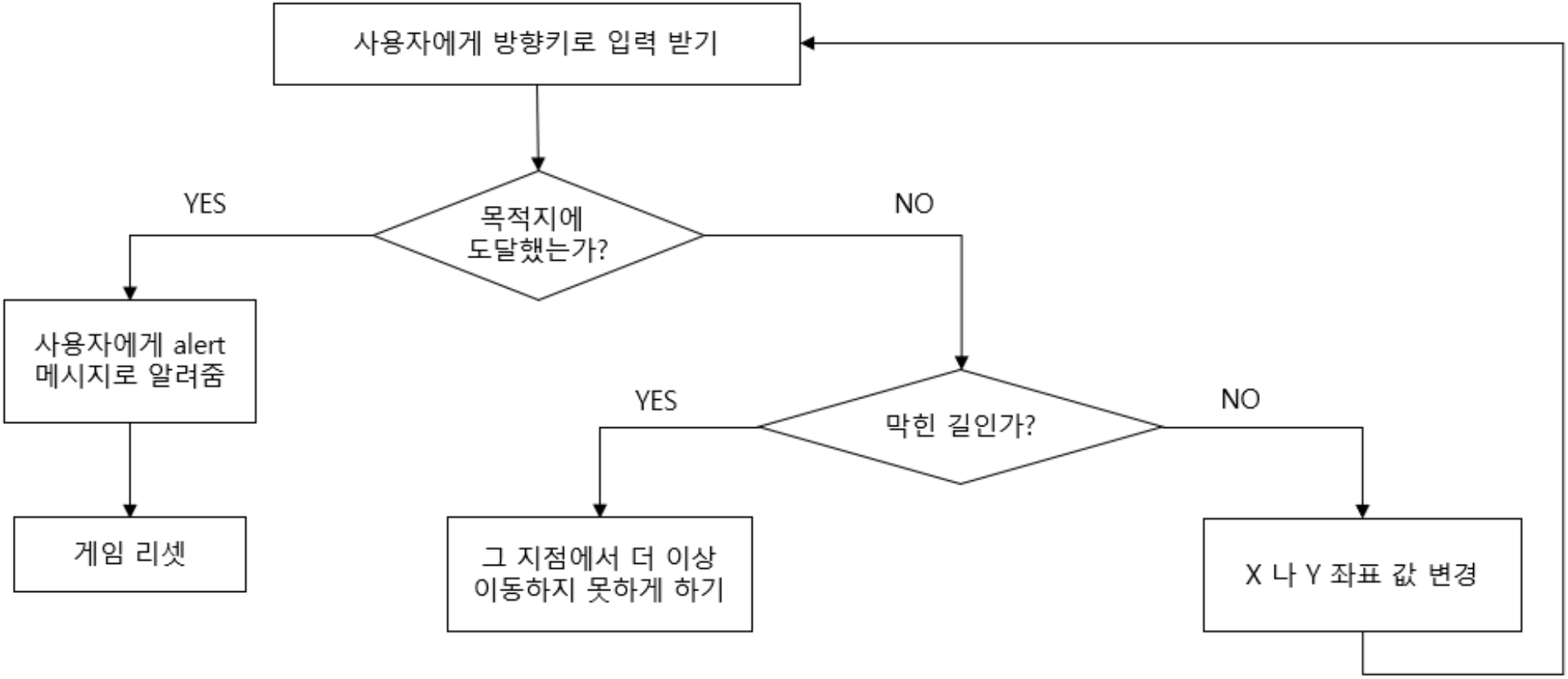
알고리즘

다른 말로 순서도
라고도 해!

1. 사용자에게 방향 입력받아서 이동한다.
2. 목적지에 도착했는지 검사한다.
   - 도착했다면, alert 메세지 보여주고 게임을 리셋한다.
   - 도착하지 않았다면, 막힌 길인지 검사한다.
3. 막힌 길인지 검사한다.
   - 막힌 길이라면, 좌표 변화도 없고 그자리 그대로 이동을 하지 않는다.
   - 막힌 길이 아니라면, 맞는 좌표(X 또는 Y) 값을 +1 하면서 이동시킨다.

알고리즘

미로 흐름도!


사용자에게 방향키로 입력 받기
목적지에 도달했는가?
YES
NO
사용자에게 alert 메시지로 알려줌
게임 리셋
막힌 길인가?
YES
NO
그 지점에서 더 이상 이동하지 못하게 하기
X 나 Y 좌표 값 변경

미로 코드 살펴보기

이렇게 초록색 칸이 있는 부분의 코드를 채워 넣으면 됩니다!

미로 코드 살펴보기

```
var tc = 21; // tile count (무조건 홀수)
var gs = 20; // 미로의 정사각형 한 칸 사이즈, grid size
var field; // 미로 벽에 대해 값이 0인 맵 위치 배열
var px = py = 1; // 0 < =  px,py < tc
var xv = yv = 0;
var tracker;
var stack;
var stucked; // 뜻 :  갇힌, 타입 : true, false
var cx, cy; // 상시 바뀌는 나의 현재 좌표 위치
```

필요한 변수 선언부

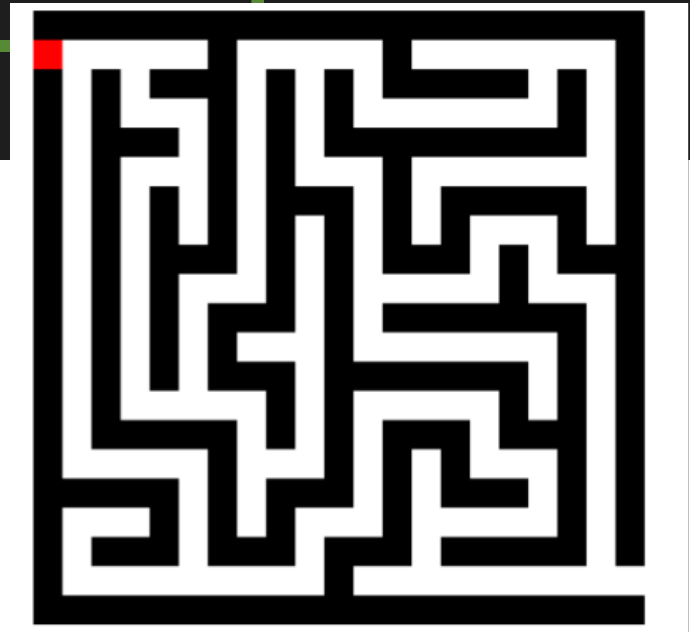
```
window.onload = function(){
    canv = document.getElementById("maze"); // 미로 캔버스
    ctx = canv.getContext("2d"); // 2d 그래픽의 컨텍스트 타입 지정
    document.addEventListener("keydown", keyPush); // 키를 누르면 발생하는 keydown 타입의 이벤트
    initialize(); // 게임 시작하기 위한 함수

}
```

미로의 틀을 만드는 코드

initialize 함수 - 뒤에 나옵니다! 간단하게 게임을 시작하기 위한 함수

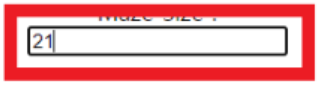
미로 코드 살펴보기

```
// enter 키를 눌렀다가 땠을 때 이벤트
function enterkey() {
    if (window.event.keyCode == 13) { // enter 키의 아스키 코드 : 13, 엔터를 누르면
    var sizeInput = document.getElementById("mazeSize").value; // 미로 사이즈, "mazeSize"라는 Id의 value값 찾기

    // 여기에 채워넣어 코드를 완성하세요!
        /*
         * 미로 사이즈가 짝수면 "Please enter an odd number." 라는 경고 메시지 생성
         * 미로 사이즈가 홀수면 미로의 사이즈를 타일의 갯수 변수에 저장, 게임 시작 함수 호출
         */

    }
}
```



enterkey 함수

미로의 사이즈 입력 후 엔터를 누른 후에 발생하는 이벤트



미로 코드 살펴보기

```
// 미로 탈출 성공 후, 게임이 리셋되거나, 처음 게임을 시작할 때 호출 하는 함수, 캔버스 스타일 & 크기도 함께 지정
function initialize(){
    document.getElementById("mazeSize").value = tc;
    make2DArray(); // tc x tc 사이즈의 배열을 생성해 field라는 변수에 저장

    ctx.fillStyle = "black"; // 캔버스의 스타일 지정 (해당 부분에 채울 색 지정)
    canv.width = canv.height = tc*gs; // 캔버스 크기 지정

    // fillRect(x, y, width, height), 시작점이 (x, y)이고 크기 width, height
    ctx.fillRect(0, 0, canv.width, canv.height); // 캔버스 시작점과 지정한 크기만큼의 사각형 생성


    makeWay(0, 1); // 입력받은 인덱스에 흰색 타일을 지정
    makeWay(tc - 1, tc - 2);

    // 미로를 생성하는 코드
    px = py = 1;
    // tracker initial position
    tracker = {x: px, y: py};
    stack = [];
    stack.push(tracker);
    stucked = false;
    randomMazeGenerator();

    cx = 0; cy = 1;
    ctx.fillStyle = "red";
    ctx.fillRect(cx*gs, cy*gs, gs, gs); // 맨 처음 빨간 점 위치

}
```

initialize 함수

미로 탈출에 성공해 게임이 리셋되거나

맨처음에 게임을 시작할 때 호출 하는 함수

캔버스 스타일, 크기 지정

미로 코드 살펴보기

```
// 미로의 벽이 아닌 지나갈 수 있는 길을 위치에 맞게 흰색 타일로 지정하는 함수
function makeWay(xx,yy){
    // 모두 0으로 채워진 field라는 배열에서 지나갈 수 있는 길을 놓을 부분에 1을 더해줌
    field[yy][xx]++;
    ctx.fillStyle="white";
    ctx.fillRect(xx*gs, yy*gs, gs, gs); // gs(grid size)만큼 흰색 타일을 그림
}
```

makeWay 함수

미로의 벽이 아닌 지나갈 수 있는 길을 흰색 타일로 지정

```
// 방향 키 이벤트
function keyPush(evt){

    switch(evt.keyCode){

        // 여기에 채워넣어 코드를 완성하세요!
        /*
         * evt.keyCode : 키보드에서 누른 키의 값을 식별하는 코드
         * 방향 키에 따른 좌표 값 변경 (왼쪽 화살표 키, 위쪽 화살표 키, 오른쪽 화살표 키, 아래쪽 화살표 키)
         * 주석으로 'here!!!' 되어있는 아래의 코드를 참고해야 함 (현재 좌표의 값이 어떻게 반영되는가)
         * ! 아스키 코드 !
         * left : 37
         * up : 38
         * right : 39
         * down : 40
        */
```

keypush 함수

왼쪽, 오른쪽, 위쪽, 아래쪽의 방향키에 대한

이벤트를 다루는 함수

```
    cx += xv; // here!!!
    cy += yv; // here!!!
    if(cx < 0 || cx > tc-1 || cy < 0 || cy > tc-1 || field[cy][cx] == 0){
        cx -= xv;
        cy -= yv;
        return;
    } else{
        ctx.fillStyle="red";
        ctx.fillRect(cx*gs, cy*gs, gs, gs);
        ctx.fillStyle="white";
        ctx.fillRect((cx-xv)*gs, (cy-yv)*gs, gs, gs);
        document.getElementById("text").innerHTML = "cx: " + cx + " cy: " + cy; // element 안의 HTML이나 XML 가져옴, 웹 상에 좌표 나타내줌

        // 여기에 채워넣어 코드를 완성하세요!
        /*
         * 현재 위치 좌표와 도착 지점의 좌표를 비교하여 도착 지점에 도달했을 때, "You Win!" 이라는 경고 메시지 생성, 게임 리셋
```

방향키로 움직이는 빨간점의 위치 상시 바뀜

도착 지점 도달 -> 게임 리셋

미로 코드 살펴보기

```
// 랜덤 미로 생성 함수
function randomMazeGenerator(){

    while(stack.length > 0) {

        // 여기에 채워넣어 코드를 완성하세요!
        /*
         * 길이 막혔을 때, 안 막혔을 때 각각 함수 호출 (어떠한 문법, 어떠한 변수, 어떠한 함수를 써야할지 잘 생각해보세요!)
         */
    }
}
```

randomMazeGenerator 함수

길이 막힌 경우, 길이 안 막힌 경우 함수 호출

미로 코드 살펴보기

```javascript
// 길이 막히지 않았을 때의 함수, tracking : 객체 탐지, 추적
function tracking(){

    // 여기에 채워넣어 코드를 완성하세요!
    /*
     * 0부터 3까지의 정수를 랜덤으로 생성하는 key 변수 생성
     * (Math.random 이용하기!, 한 줄 코드.)
     */

    switch(key){
    case 0: // left move
        xv =- 2; yv = 0;
        break;
    case 1: // up move
        xv = 0; yv =- 2;
        break;
    case 2: // right move
        xv = 2; yv = 0;
        break;
    case 3: // down move
        xv = 0; yv = 2;
        break;
    }

    px += xv;
    py += yv;
    if(px < 0 || px > tc-1 || py < 0 || py > tc-1){
        px -= xv;
        py -= yv;
        return;
    }
    if(field[py][px] < 1){
        makeWay(px - xv / 2, py - yv /2);
        makeWay(px, py);
        tracker = {x: px, y: py};
        stack.push(tracker);
        blockCheck(); // 사위가 막혔는지 확인하고 stucked 변수 갱신
    }else{
        px -= xv;
        py -= yv;
        return;
    }

}
```

tracking 함수

길이 안 막히지 않은 경우의 함수

미로 코드 살펴보기

```
function blockCheck(){
    var blockCount = 0;
    if(py+2 > tc-1 || field[py+2][px] != 0)
        blockCount++;
    if(py-2 < 0 || field[py-2][px] != 0)
        blockCount++;
    if(px+2 > tc-1 || field[py][px+2] != 0)
        blockCount++;
    if(px-2 < 0 || field[py][px-2] != 0)
        blockCount++;
    if(blockCount >= 4)
        stucked = true;
    else
        stucked = false;
}
```

tracker를 기준으로 상하좌우에 1이 저장되어 있거나 맵을 벗어나는지 확인.

벗어난다면 blockCount에 1을 더함

blockCount=4이면(모든 방향에서 위 조건이 성립) stucked변수를 true로 저장

미로 코드 살펴보기

```
// 길이 막혔을 때의 함수, backtracking : 해를 찾는 도중 해가 아니어서 막히면, 되돌아가서 다시 해를 찾아가는 기법
function backtracking(){
    var backtracker = stack.pop(); //stack 배열의 마지막 요소를 제거한 후, 제거한 요소를 반환
    px = backtracker.x;
    py = backtracker.y;
    blockCheck();
}
```

backtracking 함수

막다른 골목에 도달했을 경우 막다르지 않은 지점까지 백트랙킹,

지금껏 지나온 경로를 스택에 저장했기 때문에 스택의 마지막 요소를 제거하면서

되돌아오는 함수

1,2,3일차에 배운 것들을 응용해서
게임을 직접 완성해봅시다!