Theme：　アニマルオセロ（風）ゲーム　を　講座の内容を使って　やってみよう伏線回収

コンセプト：
1、講座の内容をどれだけ使えるか（Day18までの伏線を全力で回収）
　　→ この講座を受けたらできる。
2、とりあえず、まっさらなところからWebアプリを作る。
3、アルゴリズムは自分で構築する（どこかからパクってこない）

回収できた伏線　（講義で出てきた順）

- 自作関数（Day2)
- ナイトメア課題　flatten（Day3)
- If文（Day4）
- ２次元配列（Day7）
- メソッド.push（Day7）
- .lengthメソッド(Day7)
- オブジェクト（Day8)
- whileループ（Day11)
- forループ（Day11)
- 値としての関数（Day12）
- 自作メソッドを実装（Day13)
- DOMの要素操作 (Day15)
- CSSの操作 (Day16)
- 高階関数.forEach（Day17 ポケモンPJT)
- 高階関数.filter(Day18)
- アロー関数(Day18)

＜回収しどころの例　＝＞使いどころ＞

●自作関数（Day2)　＝＞　至る所で使っている

●ナイトメア課題　flatten（Day3)

```
//初めてのナイトメア課題　を応用したメソッドを作る
flatten: function (inarray) {
    let returnArray = [];
    let tempArray = [];
    inarray.forEach(
        (element) => {
            if (Array.isArray(element)) {
                tempArray = this.flatten(element);
                returnArray = returnArray.concat(tempArray);
            } else {
                returnArray.push(element);
            }
        }
    );
    return returnArray;
},
```

●.lengthメソッド(Day7)

●高階関数.filter(Day18)

●アロー関数(Day18)

＝＞　盤面のカウントで利用

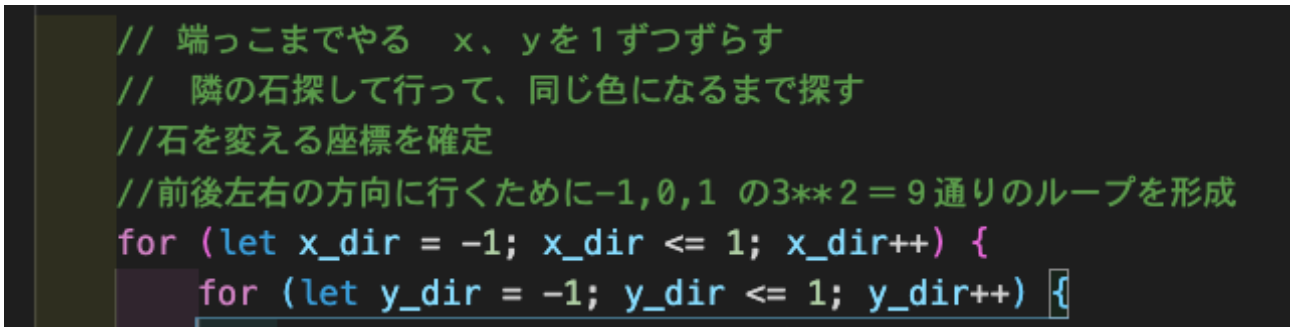
盤面データが　２次元配列に格納されているので、　それをflattenで１次元化

その配列を、.filterメソッドで抽出し .length メソッドでカウントする

```
//石の数を数えて返すメソッド
count: function () {
    return [
        this.flatten(this.data)
            .filter(element => element === 1)
            .length,
        this.flatten(this.data)
            .filter(element => element === 2)
            .length
    ];
},
```

ーーー盤面探索ロジックの話ーーー
●whileループ（Day11）
●forループ（Day11）
●If文（Day4）
●メソッド.push（Day7）
＝＞　盤面のひっくりかえす石を探索するロジックで使用

1）置かれた石を中心に　前後左右（8方向）で探索する。インデックス番号で探索方向を
　を指定するため　上下、左右の方向を　それぞれー１，０，１で表現
　(forループでうごかす)

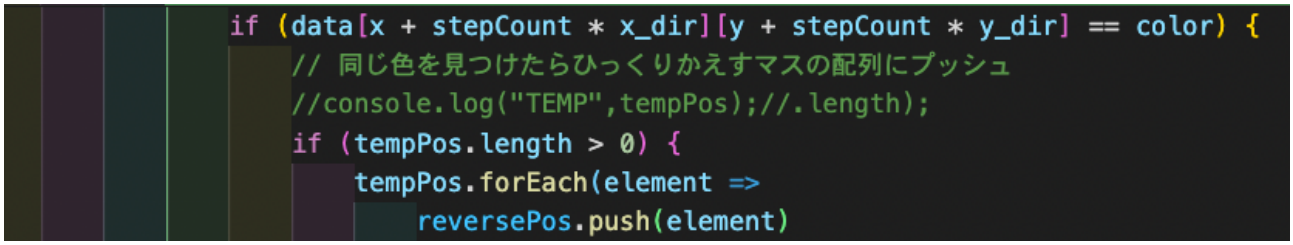


2）1）で指定した方向に向かって　端っこになるまで3〜5）を繰り返し、１マスずつ位置（配列のindexをずらして）探索する（indexが0未満or最大値を超える）

3）探索対象のマスが、置かれた石の色違う場合は、ひっくりかえす石の位置を一時的に記憶しておく配列に　座標をpush　さらに隣のマスを探索する。

4）探索対象のマスが空いている場合は挟んでいることにならないので探索終了(break)

5）探索対象のマスが、同じ色になった場合、ひっくりかえすことが確定するので、その時点で、探索終了。ひっくりかえす石の位置を　ひっくりかえすことが確定した配列にpushしてbreak→6）へ



```
if (data[x + stepCount * x_dir][y + stepCount * y_dir] == color) {
    // 同じ色を見つけたらひっくりかえすマスの配列にプッシュ
    //console.log("TEMP",tempPos);//.length);
    if (tempPos.length > 0) {
        tempPos.forEach(element =>
            reversePos.push(element)
```

6）1）に戻り、次の探索方向に向かう（forループのため　上下、左右方向で同じように処理が行われる。）

●高階関数.forEach（Day17 ポケモンPJT)

＝＞　２次元配列の盤面データをDOM操作で画面に描画する際に使用

```
//盤面を描画する関数
function boardUpdate() {
    //盤面の更新
    othelloData.data.forEach(
        (elementX, index_X) => {
            elementX.forEach(
                (erementY, index_Y) => {
                    document.getElementById(`x${index_X}y${index_Y}`).textContent
                        = othelloData.charAnimal[othelloData.data[index_X][index_Y]];
                }
            );
        }
    );
```

●オブジェクト（Day8)
●２次元配列（Day7）　＝＞盤面データを格納するために使用
●自作メソッドを実装（Day13)
　＝＞盤面データと、盤面の状況を返すメソッドをオブジェクト化。
　よくよく考えたらオブジェクト入れなくてもよかったかな　と思うけど。まいっか。

```
/*
 * オセロオブジェクト
 */
const othelloData = {
    charAnimal: ["", "", ""],
    // 初期盤面データ
    data: [
        [0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 1, 2, 0, 0, 0],
        [0, 0, 0, 2, 1, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0]
```

●値としての関数（Day12）
　＝＞　イベントリスナのコールバック関数

```
    singleButton.addEventListener(
        "click",
        function () {
            putAnimal(this.id);//ボタン押した時のコールバック関数
        }, false);
    singleButton.addEventListener(
        "mousemove",

        function () {
            const item = document.getElementById("item")
            item.style.left = event.clientX + "px";
            item.style.top = event.clientY + "px";
        }, false);
```

●DOMの要素操作 (Day15)

```
//ドキュメントにボタンを６４個配置する
function makeBoard() {

    // ボタンの生成
    const objBody = document.getElementsByTagName("body")[0];
    for (let x = 0; x < 8; x++) {
        const tempArray = [];
        const objDiV = document.createElement("div");
        for (let y = 0; y < 8; y++) {

            const singleButton = document.createElement("button");
            const tempDiv = document.createElement("div");
            const colormap = ["silver", "orchid"];
            const radiusMap = ["10px 20px 10px 20px", "20px 10px 20px 10px"]

            //ボタンの文字を作成
            singleButton.textContent = othelloData.charAnimal[othelloData.data[x][y]];

            //ボタンスタイルを設定
            singleButton.id = `x${x}y${y}`;
            singleButton.style.left = `${y * 60}px`;
            singleButton.className = `type${(x + y) % 2}`;
            // singleButton.style.backgroundColor = colormap[(x + y) % 2];
            // singleButton.style.borderRadius = radiusMap[(x + y) % 2];
            singleButton.type = 'button';
```

＝＞盤面の描画（ボタンの生成）をJSで実施
＝＞各種情報アップデートもJSから

●CSSの操作 (Day16)
＝＞共通のスタイルはCSSに集約

```
div#currentTurn {
    color: red;
    top: 5px;
    left: 5px;
    width: 400px;
    padding: 0px;
    border: 2px dotted black;
    border-radius: 35px;
    text-align: center;
    background-color: lightblue;
}

div#turnCount {
    text-align: center;
    vertical-align: bottom;
    left: 20px;
    width: 65px;
}
```