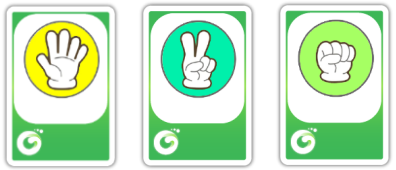
원카드 가위바위보

gd96 백시현

게임 룰

0. 양쪽 플레이어는 가위바위보 카드를 각각 3장 씩 총 9장을 가지고 시작한다.

x3 x3 x3



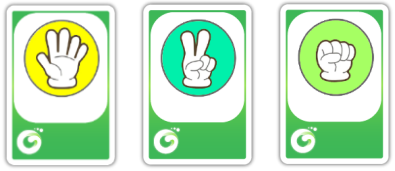
x3 x3 x3

게임 룰

2. 턴 마다 카드 한 장을 테이블에 올린다.

x3 x2 x3

vs

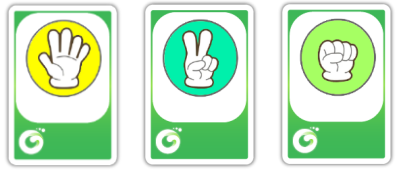


x2 x3 x3

게임 룰

3. 올린 카드의 가위바위보를 이긴 쪽이 테이블에 올라간 카드를 모두 갖는다.

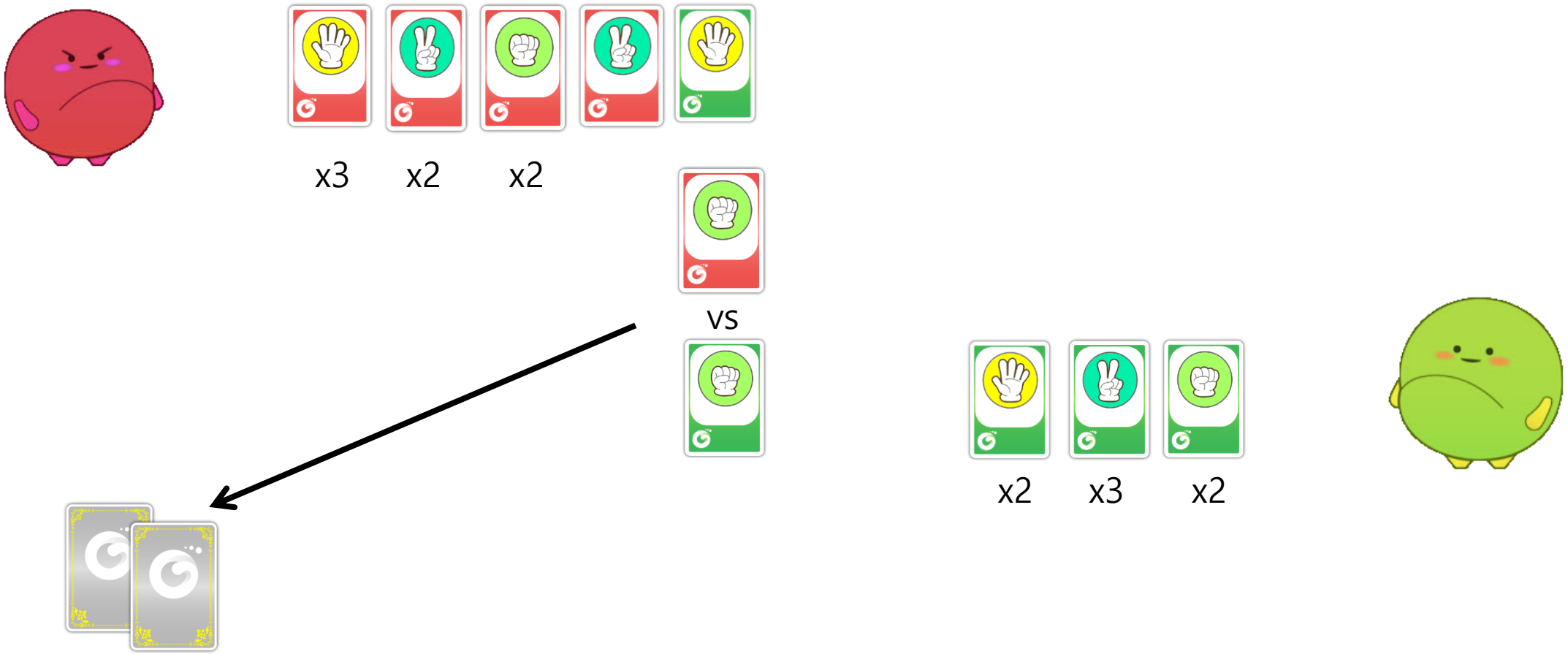
x3 x2 x3



x2 x3 x3

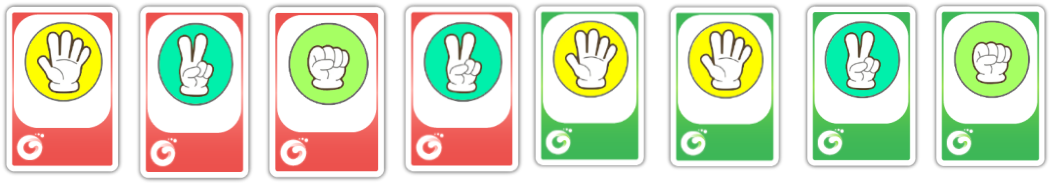
게임 룰

3-1. 비긴 경우 테이블에 올라온 두 카드는 버려진다. (전체 카드 수 감소)



게임 룰

1. 패가 0이 되면 승리. (원카드와 동일한 승리조건)



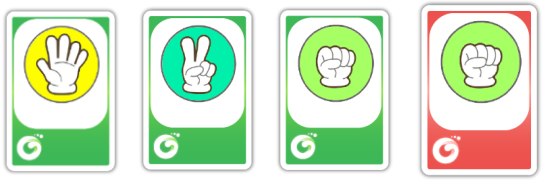
x3 x2 x2 x2 x3 x2

x0

I win!

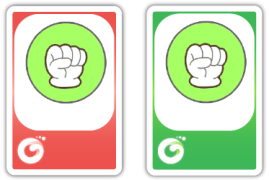
tip

- 1. 이기면 패가 늘어 다음 라운드의 선택지가 많아진다.
  그러나 승리 조건에서 멀어진다.



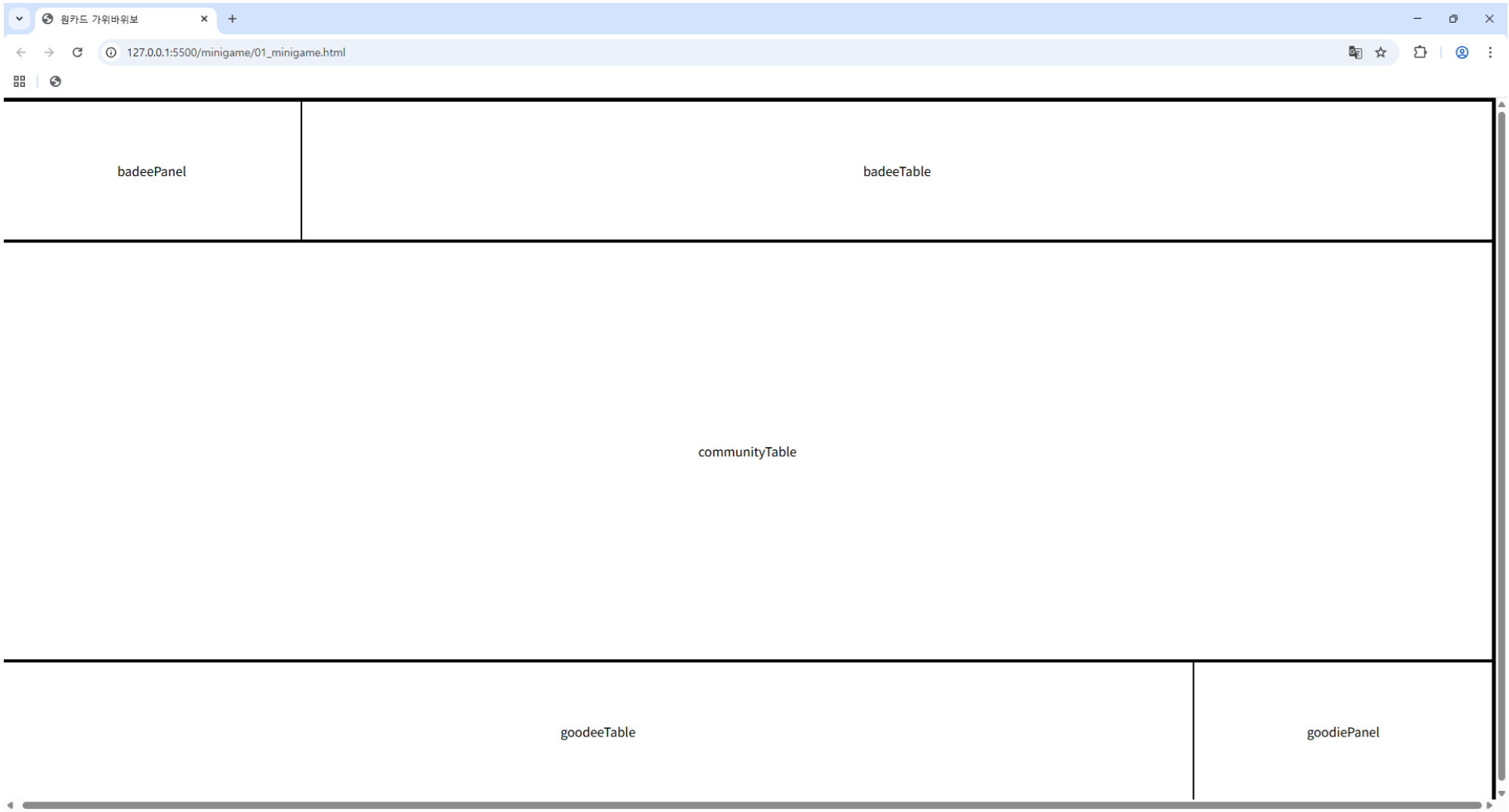
- 2. 지거나 비기면 패가 줄어들어 다음 라운드의 선택지가 줄어든다.
  그러나 승리 조건에 가까워진다.

- 3. 상대보다 패가 적을 경우 비김을 유도하여
  내 패도 줄이고 상대의 선택지도 줄이는 것이 최고의 선택.



- 4. 상대보다 패가 많을 경우 일부러 지는 것을 유도하여
  내 패는 줄이고 상대 패는 늘리는 것이 최고의 선택.

영역 나누기

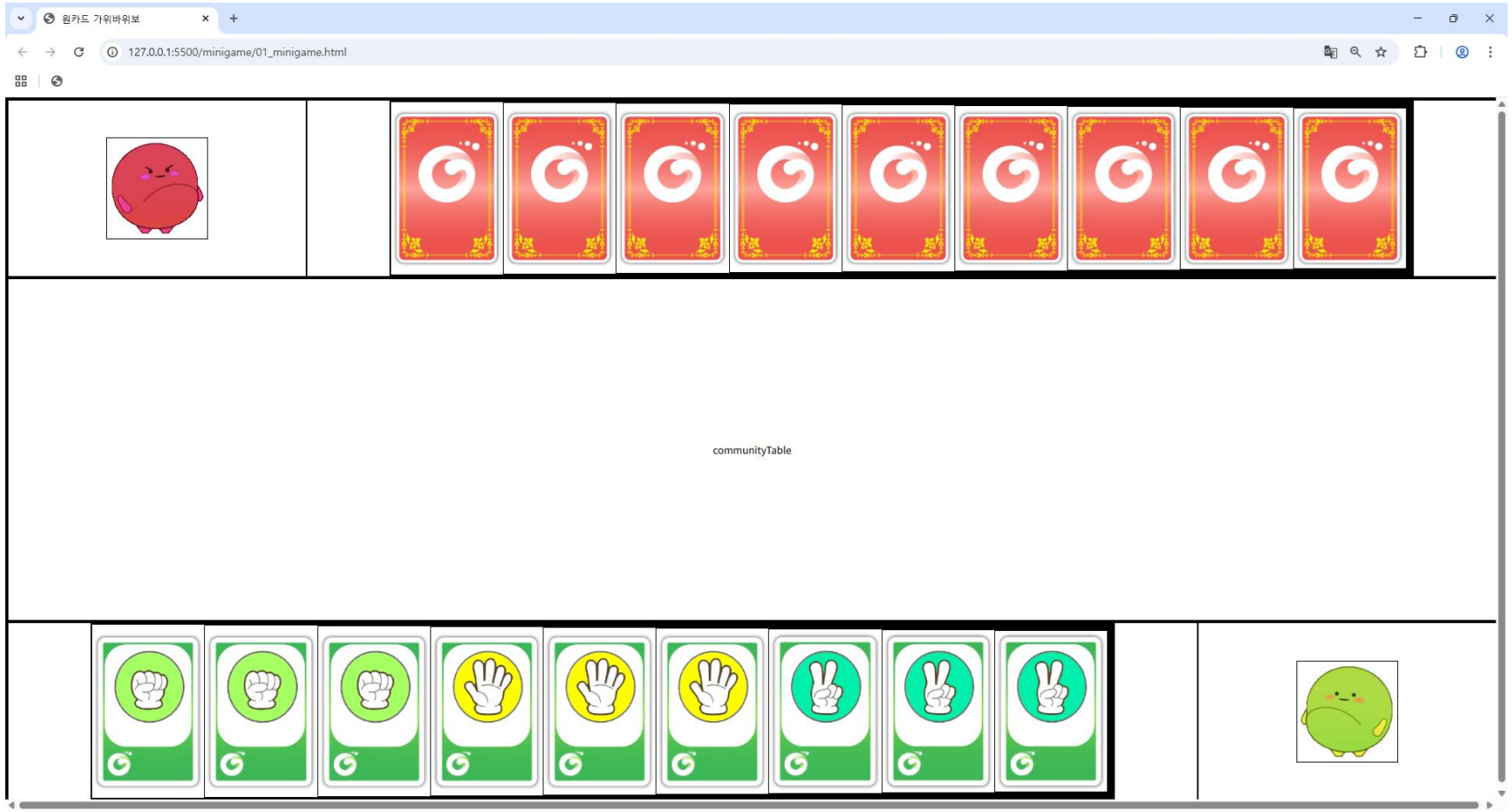

원카드 가위바위보
127.0.0.1:5500/minigame/01_minigame.html
badeePanel
badeeTable
communityTable
goodeeTable
goodiePanel

```html
<body>
    <div class="fullscreen">  <!--전체화면-->

<!--badeeTable 영역-->
        <div class="charaTable">
            <div class="charaPanel">badeePanel</div>
            <div class="deckTable badeeTable">badeeTable</div>
        </div>
<!--badeeTable 영역 끝-->

<!--communityTable 영역-->
        <div class="communityTable">
            <div class="community">communityTable</div>
        </div>
<!--communityTable 영역 끝-->

<!--goodeeTable 영역-->
        <div class="charaTable">
            <div class="deckTable goodeeTable">goodeeTable</div>
            <div class="charaPanel">goodiePanel</div>
        </div>
<!--goodeeTable 영역 끝-->

    </div>  <!--전체화면 끝-->

</body>
```

div와 css를 사용해 영역을 나누었다.

```css
div{
    text-align: center;
    display: flex;
    align-items: center;
    justify-content: center;
}
.fullscreen{
    display: flex;
    flex-direction: column; /*자식들을 세로로 배치*/
    width: 100%;
    height: 100%;
    align-items: stretch;
}
.charaTable{
    display:flex; /*자식들을 가로로 배치*/
    align-items: stretch;
    flex: 2 1 0;

}
```

이미지 추가 및 카드 생성


원카드 가위바위보
127.0.0.1:5500/minigame/01_minigame.html
communityTable

```
function cardPrepare(){      //카드 18장 생성
    card =  ["r.png","p.png","s.png","rev.png"]
    const frontPath = (owner, type)=>{
        const idx =(type === 'rock')?0:(type==="paper")?1:2;
        return `./image/cards/card_${owner}_${card[idx]}`;
    }
    let cardImage1 = ""
    for(i=0;i<9;i++){
        const type = (i <3)? 'rock' : (i<6?'paper':'scissors');
        const front = frontPath('goodee', type);
        cardImage1+=`
            <div>
                <img class='player pic ${type}'
                id='pic${i}'
                data-owner="player"
                data-type="${type}"
                data-front="${front}"
                data-origin="goodee"
                src="${front}">
            </div>`
    }
    document.querySelector("#gHand").innerHTML = cardImage1;
    let cardImage2 = ""
    for(i=9;i<18;i++){
        const type = (i<12) ? 'rock' : (i<15 ? 'paper' : 'scissors');
        const front = frontPath('badee',type);
        cardImage2 +=`
            <div>
                <img class="enemy pic ${type}"
                    id="pic${i}"
                    data-owner="enemy"
                    data-type="${type}"
                    data-front="${front}"
                    data-origin="badee"
                    src="${front}">
            </div>`;
    }
    document.querySelector("#bHand").innerHTML = cardImage2;
    document.querySelectorAll("#bHand img.enemy").forEach(e=>{
        e.setAttribute("src",'./image/cards/card_badee_rev.png');
    });
}
```

for문을 사용하여 카드 18장을 추가한다

i를 0부터 17까지 돌려
각 카드의 id값을 넣어준다.

모든 카드에 type의 data를 저장했다.

Enemy의 카드는 생성되자마자
모두 img 를 rev(반전) 된 그림의 src로 넣어주었다.

type 데이터를 통해
반전된 그림을
다시 원래의 img로
바꾸는 것이
가능하다

라운드 진행 방식생성

1. 카드를 커뮤니티 영역에 드래그 앤 드롭한다.

2. Go 버튼이 생성되며 클릭 시 상대도 카드를 낸다

3.

4. 결과가 뜬다

```
function enableDragAndDrop() {
    const gHand = document.querySelector('#gHand');
    gHand.addEventListener('dragstart', (e)=>{  //드래그 발생했을 때
        const img = e.target.closest('img.pic'); //드래그 발생 요소가 img인가
        if (!img) return;                        //img아니면 무시
        img.classList.add('is-dragging');   //애니효과를 위해 클래스 추가
        e.dataTransfer.setData('cardId', img.id);  //카드id 데이터 저장
        })
    gHand.addEventListener('dragend', (e) =>{   //드래그 끝났을 때
        const img = e.target.closest('img.pic');
        if (img) img.classList.remove('is-dragging');   //드래그 클래스 제거
    });

    const comm = document.querySelector('.community');
    comm.addEventListener('dragover', (e) =>{
        e.preventDefault(); //드랍 가능 영역 위에 있을 때 드랍차단을 막기
    });
    comm.addEventListener('drop', async (e) => {
        e.preventDefault(); //기존 탭에서 드롭 처리

        const cardId = e.dataTransfer.getData('cardId');//저장된 id 불러오기
        const from = e.dataTransfer.getData('from');
        const card = document.getElementById(cardId);
        if (!card) return;
        const exists = getCommunityPlayerCard();
        if (exists && exists !== card) {
            shake(card);
            return;
        }
        e.target.appendChild(card);//community 영역에 카드 넣기
        card.classList.add('drop-anim') //드랍 애니메이션 클래스
        setTimeout(()=> card.classList.remove('drop-anim'),300);
        await ensureGoButton()
        await updateGoVisibility();
    });
```

드래그 앤 드롭 기능 구현.
플레이어가 카드를 둘 차례가 아닐 경우 false(버그 방지)

플레이어 카드가 커뮤니티에 있을 경우 함수 처리하는 버튼 생성

```
btn.addEventListener('click', onGoClick);
```


```
function enableDragAndDrop() {…
}
async function onGoClick(e) {…
}
function enemyPlay(delayMs = 500, flipDelayMs =200){…
}
async function checkWinner() {…
}
async function showResultImage(result){…
}
```

Go 버튼 클릭 시 enemyPlay 처리

```
async function onGoClick(e) {
    const btn = e.currentTarget;
    btn.disabled = true;
    const gHand = document.querySelector('#gHand');
    gHand.style.pointerEvents = 'none';
    await enemyPlay();//상대도 카드 내는 로직
    btn.classList.remove('show'); //go 버튼 숨기기
    btn.disabled = false;
}
```

```
async function showResultImage(result){
    const comm =document.querySelector('.community');
    let imgSrc ='';
    switch (result){
        case 'playerWin': imgSrc = './image/ui/win.png'; break;
        case 'enemyWin': imgSrc = './image/ui/lose.png'; break;
        case 'draw': imgSrc = './image/ui/draw.png'; break;
        default: return;
    }
    const img = document.createElement('img');
    img.src =imgSrc;
    img.className = 'result-image';
    comm.appendChild(img);

    img.classList.add('show');
    await new Promise(r => setTimeout(r,1500));
    add('hide');
    > img.remove(),800);
```

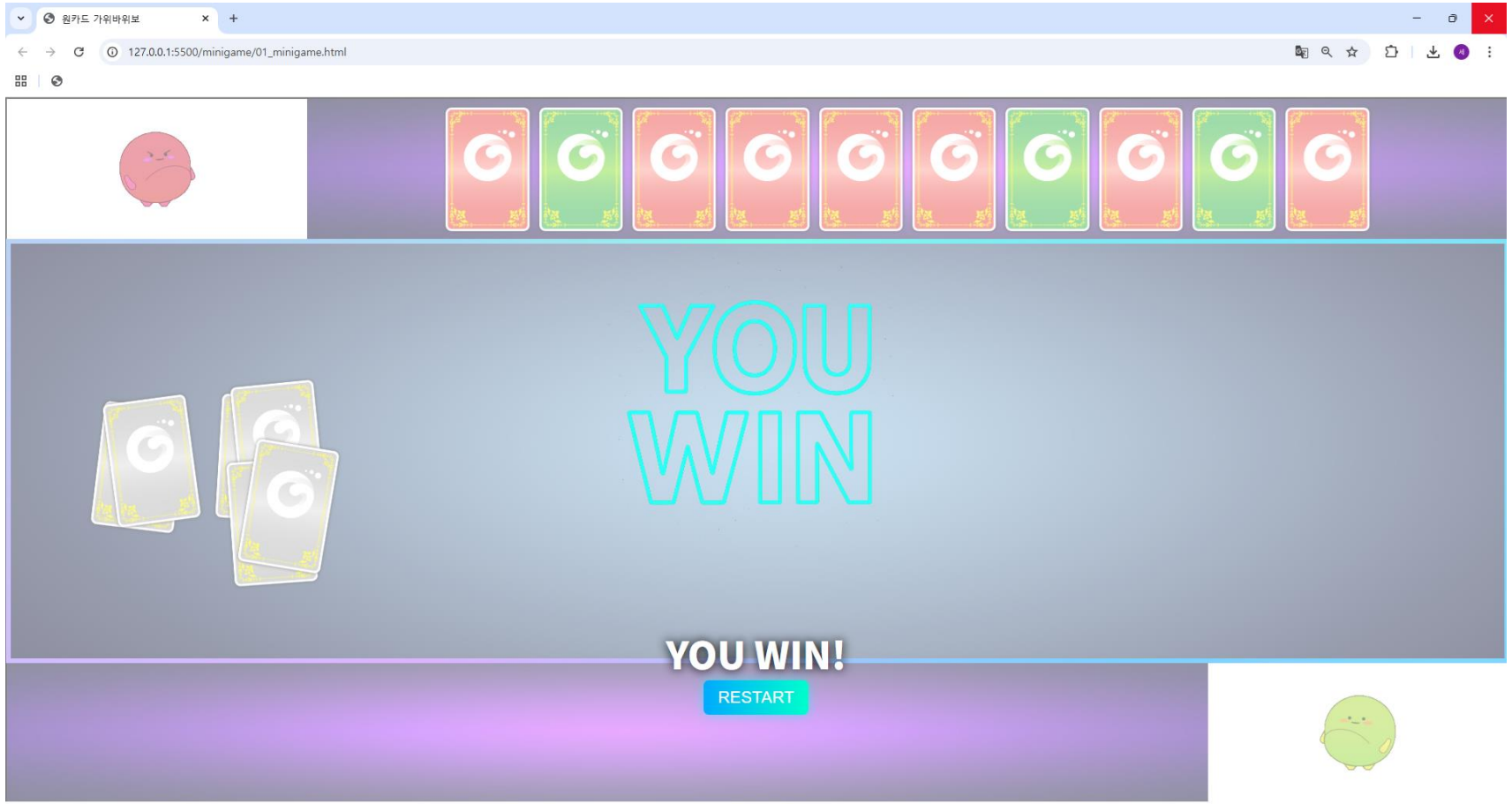
winner 확인 후 결과 이미지 show

```
async function checkWinner() {
    const comm = document.querySelector('.communi
    const playerCard = comm.querySelector('img[da
    const enemyCard = comm.querySelector('img[dat
    if (!playerCard || !enemyCard) return;

    const pType = getCardType(playerCard);
    const eType = getCardType(enemyCard);

    const result = getWinner(pType, eType);

    await showResultImage(result);
    await resolveRound(result);
}
```

```
setTimeout(()=> {checkWinner().then(resolve);
    }, 2000);
```

enemy는 특정 확률로 특정 카드를 커뮤니티에 드랍,
2초 뒤 checkWinner 처리

게임 종료 규칙 생성


원카드 가위바위보
127.0.0.1:5500/minigame/01_minigame.html
YOU
WIN
YOU WIN!
RESTART

```
async function checkWinner() {
    const comm = document.querySelector('.community');
    const playerCard = comm.querySelector('img[data-owner="player"]');
    const enemyCard = comm.querySelector('img[data-owner="enemy"]');
    if (!playerCard || !enemyCard) return;

    const pType = getCardType(playerCard);
    const eType = getCardType(enemyCard);

    const result = getWinner(pType, eType);

    await showResultImage(result);
    await resolveRound(result);
}
```

result 가 정의되면
라운드 정리

```
async function resolveRound(result) {
    const comm = document.querySelector('.community');
    const playerCard = getCommunityPlayerCard();
    const enemyCard = getCommunityEnemyCard();
        if (!playerCard || !enemyCard) return;…
    [playerCard, enemyCard].forEach(c => c.style.transition = 'all 0.5s ease');

        // 구디 승리: 그대로 자신의 손으로
        if (result === 'playerWin') {…
    }

        // 배디 승리: 두 카드 모두 rev 상태로 뒤집어 복귀
        else if (result === 'enemyWin') {…
    }

        // 무승부: 두 카드 neutral화 후 'grave'로 이동
        else if (result === 'draw') {…
    }

    await new Promise(r => setTimeout(r, 500));
    await resetRound();
}
```

정리가 끝나면 라운드 리셋

```
async function resetRound() {
    const comm = document.querySelector('.community');
    comm.innerHTML = '';
    const gHand = document.querySelector('#gHand');
    gHand.style.pointerEvents = 'auto';
    await updateGoVisibility();
    ensureGoButton();
    checkGameOver();
}
```

```
function restartGame() {    //게임재시작
    document.querySelectorAll('#gHand, #bHand, .co
    const grave = document.querySelector('.grave'
    if (grave) grave.remove();
    init(); //다시 init 발동
}
```

Restart 버튼 누를 시 모든 영역의
데이터 remove 후 다시 init 처리

```
async function init() {
    charaPrepare(); //구디배디 생성
    cardPrepare();  //카드 9장 생성
    prepareGraveArea(); //묘지 생성
    wireUI();       //애니메이션 기능
}
```

새게임 재생성

```
await new Promise(r => setTimeout(r, 100));
overlay.classList.add('show');


const restartBtn = overlay.querySelector('.restart-btn');
restartBtn.addEventListener('click',()=>{
    overlay.classList.add('fadeout');
    setTimeout(()=>{
        overlay.remove();
        restartGame();
    }, 1500);
});
```

```
function checkGameOver() {  //남은 패의 갯수 확인
    const playerCards = document.querySelectorAll('#gHand img').length;
    const enemyCards = document.querySelectorAll('#bHand img').length;

    if (playerCards === 0 && enemyCards === 0) showFinalResult('noWin');
    else if (playerCards === 0) showFinalResult('goodeeWin');
    else if (enemyCards === 0) showFinalResult('badeeWin');
}
```

라운드를 리셋 했는데
만약 패.length가 0이라면
finalResult 이미지와
Restart 버튼을 show

치트키 추가

showFinalResult 코드와
restartGame 코드를 위해

개발 편의성 추가

Ctrl+1 = 강제 승리
Ctrl+2 = 강제 무승부
Ctrl+3 = 강제 패배

```
//치트코드
document.addEventListener('keydown', (e)=>{
    if (!e.ctrlKey) return;

    const comm = document.querySelector('.communit
    const playerCard = getCommunityPlayerCard();
    const enemyCard = getCommunityEnemyCard();

    if(!comm) return;

    switch (e.key.toLowerCase()){
        case '1' :
            console.log('Cheat:Player Win')
            checkWinnerOverride('goodeeWin');
            break;
        case '2' :
            console.log('Cheat:No Win');
            checkWinnerOverride('noWin');
            break;
        case '3' :
            console.log('Cheat:Enemy Win');
            checkWinnerOverride('badeeWin');
            break;
        default:
            return;
    }
    e.preventDefault();
    e.stopPropagation();
});
async function checkWinnerOverride(forcedResult){
    await showFinalResult(forcedResult);
}
```

배디(enemy)의 카드 선택 방식

R. P. S. 세 가지 타입의 카드 중
각 카드의 승률을 계산하여 승률에 비례한 확률로 Math.random 구현

Ex) 가위를 내서 이길 확률 : 70% (반올림)
바위를 내서 이길 확률 : 10% (반올림)

1. Math.random 으로 0~9까지 총 10개의 정수 중 무작위 숫자 출력

2. 0~6 이 출력되면 가위. 7이 출력되면 바위. 나머지가 출력되면 보

카드의 승률 계산 방식

양측이 동시에 패를 고르는 **제로섬** 한 턴 게임이므로,
현 상태의 최적 가치는 **폰 노이만의 최소최대정리**로:

$$V(P,O) \;=\; \max_{\pi\in\Delta(A)} \min_{\sigma\in\Delta(B)} \sum_{a\in A}\sum_{b\in B} \pi(a)\,\sigma(b)\,M_{a,b}.$$

- π: 플레이어의 혼합전략(각 패를 낼 확률 분포)
- σ: 상대의 혼합전략

이는 다음 **선형계획법(LP)**으로 바로 풀 수 있습니다.

$$\boxed{\begin{aligned} V(P,O) \;&=\; \begin{cases} 1, & |P|=0 \\ 0, & |O|=0 \\ \text{Solve-LP}\big(M(P,O)\big), & \text{otherwise} \end{cases} \\ M_{a,b}(P,O) \;&=\; V\big(T(P,O\,|\,a,b)\big). \end{aligned}}$$

- 종료상태는 즉시 반환.
- 그 외에는 가능한 액션쌍에 대해 T로 다음 상태를 만들고, **메모이제이션**으로 V를 재귀 계산.
- 계산된 M으로 **LP**를 풀어 $V(P,O)$와 최적 혼합전략 $\pi^{\backslash *}$를 얻음.
- 실제 플레이에선 $\pi^{\backslash *}$에 따라 확률적으로 카드를 내면 "알파고식" **최적 정책**이 됩니다.
  (남은 카드가 1~2장으로 줄면 가용 액션이 축소되므로 자동으로 **결정적(거의 순수전략)**에 수렴합니다.)

> 구현 팁: 상태공간은 $(p_S, p_R, p_P, o_S, o_R, o_P)$로 최대 4^6 미만(각 타입 0~3장)이라 **완전탐색 DP가 충분히 가능**합니다. 각 상태에서 푸는 LP는 $\le 3\times 3$ 소형이라 연산량도 매우 작습니다.

(플레이어 입장 LP)

$$\begin{aligned} &\text{maximize } v \\ &\text{subject to } \sum_{a\in A}\pi(a)M_{a,b} \;\ge\; v \quad (\forall b\in B) \\ &\qquad\qquad \sum_{a\in A}\pi(a) \;=\; 1, \quad \pi(a)\ge 0 \end{aligned}$$

(상대 입장 LP, 쌍대형)

$$\begin{aligned} &\text{minimize } w \\ &\text{subject to } \sum_{b\in B}\sigma(b)M_{a,b} \;\le\; w \quad (\forall a\in A) \\ &\qquad\qquad \sum_{b\in B}\sigma(b) \;=\; 1, \quad \sigma(b)\ge 0 \end{aligned}$$

최적해에서 $v = w = V(P,O)$가 되고, $\pi^{\backslash *}, \sigma^{\backslash *}$가 **현 턴의 최적 혼합전략**입니다.
이렇게 얻은 $V(P,O)$는 **"한 턴 앞"만 본 값이 아니라**, $M_{a,b}$에 이미 재귀적으로 **최종 승리확률**이 들어 있으므로 **전게임의 최적가치**가 됩니다.

상대를 분포 $q(b)$ (예: **남은 장수 비례** $q(b)=o_b/|O|$)로 모델하면,
현 턴에서 **기댓값 최대화**로 단순화됩니다:

$$a^{\backslash *} \;=\; \arg\max_{a\in A} \sum_{b\in B} q(b)\; V\big(T(P,O|a,b)\big).$$

(이 경우 LP 없이 **기댓값**만 비교하면 됩니다.)

- 전이 T:

$$(P',O') = \begin{cases} (P-\mathbf{e}_a,\; O-\mathbf{e}_b), & a=b \\ (P+\mathbf{e}_b,\; O-\mathbf{e}_b), & a\succ b \\ (P-\mathbf{e}_a,\; O+\mathbf{e}_a), & b\succ a \end{cases}$$

- 가치재귀:

$$V(P,O) = \begin{cases} 1, & |P|=0 \\ 0, & |O|=0 \\ \max_\pi \min_\sigma \sum_{a,b}\pi(a)\sigma(b)\,V\big(T(P,O|a,b)\big), & \text{else} \end{cases}$$

단기 전략과 장기 전략

- 단기 전략:

  이번 턴에서 덱을 줄이는 것만이 목표.
  -> 패가 더 많을 때는 무조건 LOSE를 의도
  -> 패가 같으면 최대한 LOSE를 의도, 차선으로는 DRAW 를 의도.
  -> 패가 더 적을 때는 최대한 DRAW 를 의도, 차선으로 LOSE를 의도

- 장기 전략:

  내가 낼 카드와 상대가 낼 카드

  두 경우의 수를 따져서 '다음 턴 덱 분포를 계산'
  -> 끝까지 갔을 때를 계산해서 가장 승률이 높은 카드를 뽑음

단기 전략 알고리즘

$$\text{mode} = \begin{cases} \text{LOSE} & \text{if } tot_P > tot_E \\ \text{LOSE} \rightarrow \text{DRAW} & \text{if } tot_P = tot_E \\ \text{DRAW} \rightarrow \text{LOSE} & \text{if } tot_P < tot_E \end{cases}$$

If (myDeck.length > enemyDeck .length) LOSE

Else if (myDeck .length < enemyDeck .length) DRAW

Else { if canLose = LOSE
else DRAW }

여기서 lose나 draw의 방식은, 상대가 가장 많이 가지고 있는 TYPE 를 노리고 내 카드를 내는 방식

Ex) 

상대는 가위를 가장 많이 가지고 있으니
LOSE를 의도하면 바위
DRAW를 의도하면 가위
를 선택하게 끔 합니다.

장기 전략 알고리즘

함수 정의

P[ ] = 배디가 가진 카드 수
O[ ] = 구디가 가진 카드 수
a = 낼 수 있는 카드
b = 상대가 낼 수 있는 카드
V = 최종 승률
E = 이번턴 기대 승률

플레이어 (구디) 는 무조건 "단기 전략"을 사용한다고 계산함.

컴퓨터(배디) 가 낼 카드 = a

플레이어(구디) 가 낼 카드 = b

$$V(e,p) = \begin{cases} 1 & \text{if } \sum e = 0 \quad \text{(배디가 승리)} \\ 0 & \text{if } \sum p = 0 \quad \text{(배디가 패배)} \\ \max_{a \in A} \sum_b q(b) \cdot V(\text{next}(e,p,a,b)) & \text{otherwise} \end{cases}$$

이번 턴이 아니라 최종적으로 이기기 위해 구디와 배디가 가진 카드 타입의 수를 보고
최대 승률의 수를 계산하는 알고리즘

함수 정의

P[ ] = 배디가 가진 카드 수
O[ ] = 구디가 가진 카드 수
a = 낼 수 있는 카드
b = 상대가 낼 수 있는 카드
V = 최종 승률
E = 이번턴 기대 승률

컴퓨터의 기대 승률 계산

$E[a] = \Sigma_b \; q(b) * V(nextState(a,b))$

E[a] -> a를 냈을 때 기대되는 승률

Σ_b -> b의 모든 경우의 값들을 더해서

q(b) -> b를 낼 확률 ($0 \le q(b) \le 1$) 0%~100% 사이의 값

q(S)=n/6,q(R)=n/6,q(P)=n/6

V(nextstate(a,b)) ->최종승률 * a카드와 b카드가 나왔을 경우 다음 카드 상태

카드 a를 냈을 때 상대는 q(b)의 확률로 b 카드를 낸다.

각 b 마다 다음 턴의 최종 승률은 V(nextstate(a,b)) 가 된다.

즉, a 를 냈을 때 기대되는 "다음 턴의 승리확률(E[a])"은
"지금 승리확률(E) x 다음 턴의 최종승률V(nextstate(a,b))"의 값을
모든 b에 대해 전부 더한 값이다.

플레이어 (구디) 는 무조건
"단기 전략"을 사용한다고 계산함.

단기 전략의 장점 = 다음 판의 승률이 올라갈 확률이 높다.
단기 전략의 단점 = 패턴화가 되며 예측이 쉬워진다.

-> 초반에 좋고 후반에 안 좋다.

장기 전략의 장점 = 최종적으로 우승할 확률이 높다
장기 전략의 단점 = 카드가 많을 때는 계산되는 승률이 매우 낮다.

-> 후반에 좋고 초반에 안 좋다.

가중치 전략 => 람다식

$$\lambda = \begin{cases} 0.55 & \text{if 총합} \leq 5 \quad (\text{막판, 장기}\uparrow) \\ 0.30 & \text{if 총합} \leq 10 \quad (\text{중반}) \\ 0.15 & \text{if 총합} > 10 \quad (\text{초반, 단기}\uparrow) \end{cases}$$

함수 정의

P[ ] = 배디가 가진 카드 수
O[ ] = 구디가 가진 카드 수
a = 낼 수 있는 카드
b = 상대가 낼 수 있는 카드
V = 최종 승률
E = 이번턴 기대 승률

E 를 정수 1부터 10까지 점수화

score[a] = round(E[a]*10)

0.x 를 10 곱하고 반올림

x = 점수가 가장 높은 카드 (정배, 가장 합리적)
y = 점수가 가장 낮은 카드 (역배, 역심리 용도)

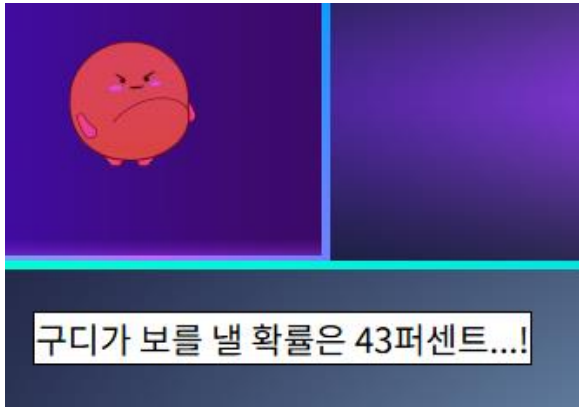
Math.random 사용해 1부터 10까지 무작위 정수 산출

if (random ≤ score[x]) → 정배(x)
else if (random ≤ score[x] + score[y]) → 역배(y)
else → 중도

Ex) 가위 승률 70퍼센트 , 바위 승률 10퍼센트
-> Math.random 을 통해 1~7이 나오면 가위
8이 나오면 바위
9~10이 나오면 보

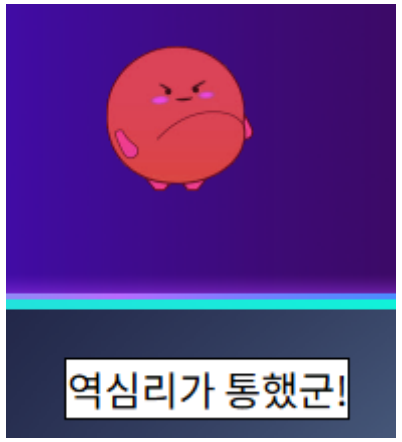
실제로 작동하는 것인가?

디버그용 배디의 대사 추가

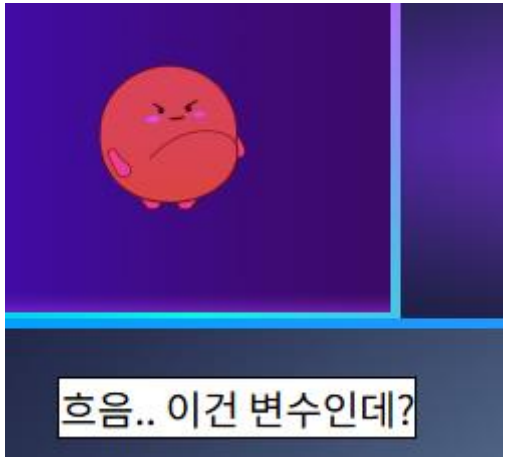


플레이어가 단기 전략으로 뽑을만한
가장 높은 확률의 카드 TYPE 와 계산상 그 확률

컴퓨터가 의도한 결과가
정배를 통해 나왔을 때



컴퓨터가 의도한 결과가
역배를 통해 나왔을 때



컴퓨터가 의도한 결과가
안 나왔을 때

```
배디가 "혼합" 전략을 택했습니다.
배디는 구디가 "보"를 42.6퍼센트 확률로 낼 것이라 생각하여 "draw"를 의도하고 내밀 카드를 고민 중입니다.
최후의 순간 배디는 "역배" 를 골랐습니다. (정배 57.1퍼센트/역배 42.9퍼센트)
[BADDY QUIP]  ▶ {resultStr: 'enemyWin', outcome: 'WIN', intended: 'draw', bucket: 'y', line: '흐음.. 이건 변수인데?'}
배디가 "혼합" 전략을 택했습니다.
배디는 구디가 "보"를 49.5퍼센트 확률로 낼 것이라 생각하여 "lose"를 의도하고 내밀 카드를 고민 중입니다.
최후의 순간 배디는 "역배" 를 골랐습니다. (정배 62.5퍼센트/역배 37.5퍼센트)
[BADDY QUIP]  ▶ {resultStr: 'playerWin', outcome: 'LOSE', intended: 'lose', bucket: 'y', line: '역심리가 통했군!'}
```

console.log 도 추가.

가끔 장기 or 혼합 전략이 되면
배디가 안 좋은 결과인 WIN 이 나왔어도 좋아하는 모습을 볼 수 있음.

감상

1. 게임에 대한 이해도가 컴퓨터보다 낮아지기 시작하면 버그를 찾기 어려워진다.

2. GPT 의존도가 너무 높으면 버그를 고치기 무척이나 어려워진다 (GPT는 전체 구조를 알지 못함)

3. 개발하기 위해서는 수 많은 치트키를 만들어야 한다.

4. 게임의 의도와는 별개로 "LOSE"를 계속 시도해야 된다는 개념이 반감이 크다.
   ->이기면 카드를 더 가져간다는 것이 자연스럽다고 생각했는데,
   그것과 상관없이 통상적인 개념으로 WIN의 결과가 무조건 승리조건에 가까워지도록 바꾸는 것이 옳다.

추가 기능의 방향성?

정배 역배 혹은 장기전략 삭제 등 확률 조율로 난이도 조절 가능 (상, 중, 하)