프로젝트

미니게임천국

프로젝트기반 프론트엔드(React, Vue) 웹&앱 SW 개발자

장보은

목차 a table of contents

1 기획의도

2 사용 언어 및 도구

3 주요 기능 및 동작

4 시연

Part 1

기획의도

Part 1

기획의도

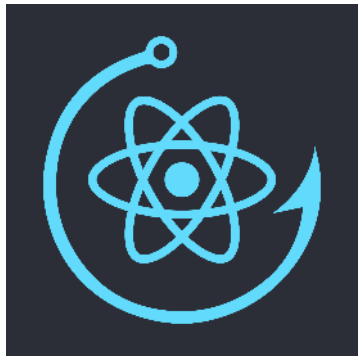
게임 만들기를 통해 다양한 React Hooks를 사용하여 부족했던 부분을 채우고 공부하며 범용성 있고 효율적으로 React를 사용하기 위해 기획하였다.

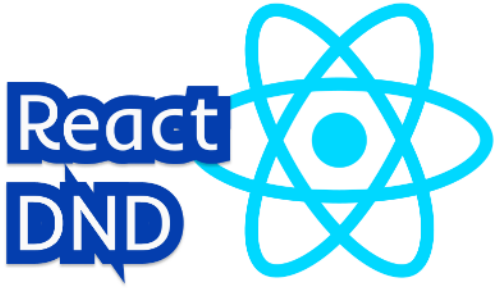
또한 게임 배너, 이벤트 등을 고려하여 다양한 게임을 만들어보았다.

Part 2

사용 언어 및 도구

사용 언어 및 도구





RxJS

Chess.js

Part 3

주요 기능 및 동작

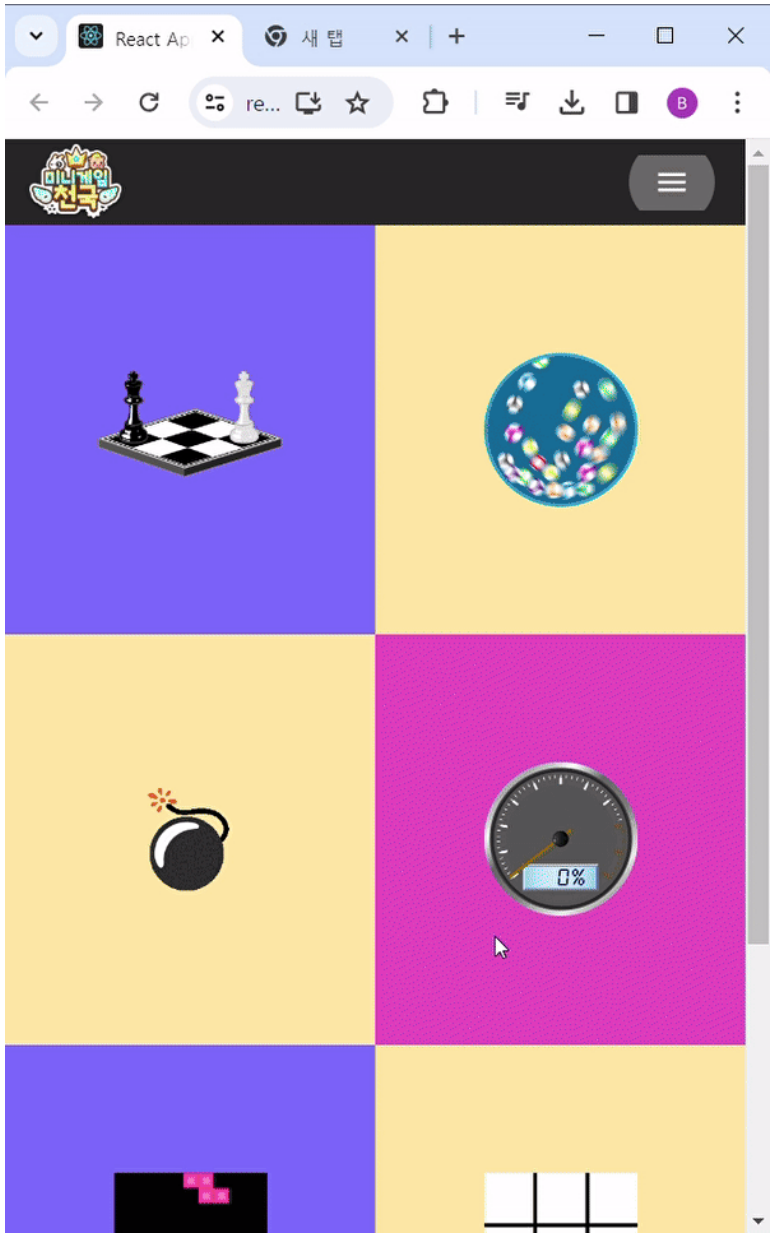
주요 기능 및 동작

반응 속도 테스트

초록색 화면이 나왔을 때 클릭하면

반응 속도 시간이 나온다.

너무 빠르게 클릭하면 다른 문구가 나온다.



주요 기능 및 동작

반응 속도 테스트

useState와 useRef를 사용하여

화면에 영향을 주는 것과

주지 않는 것을 구분하여 사용하고

useCallback을 이용하여 최적화 하였다.

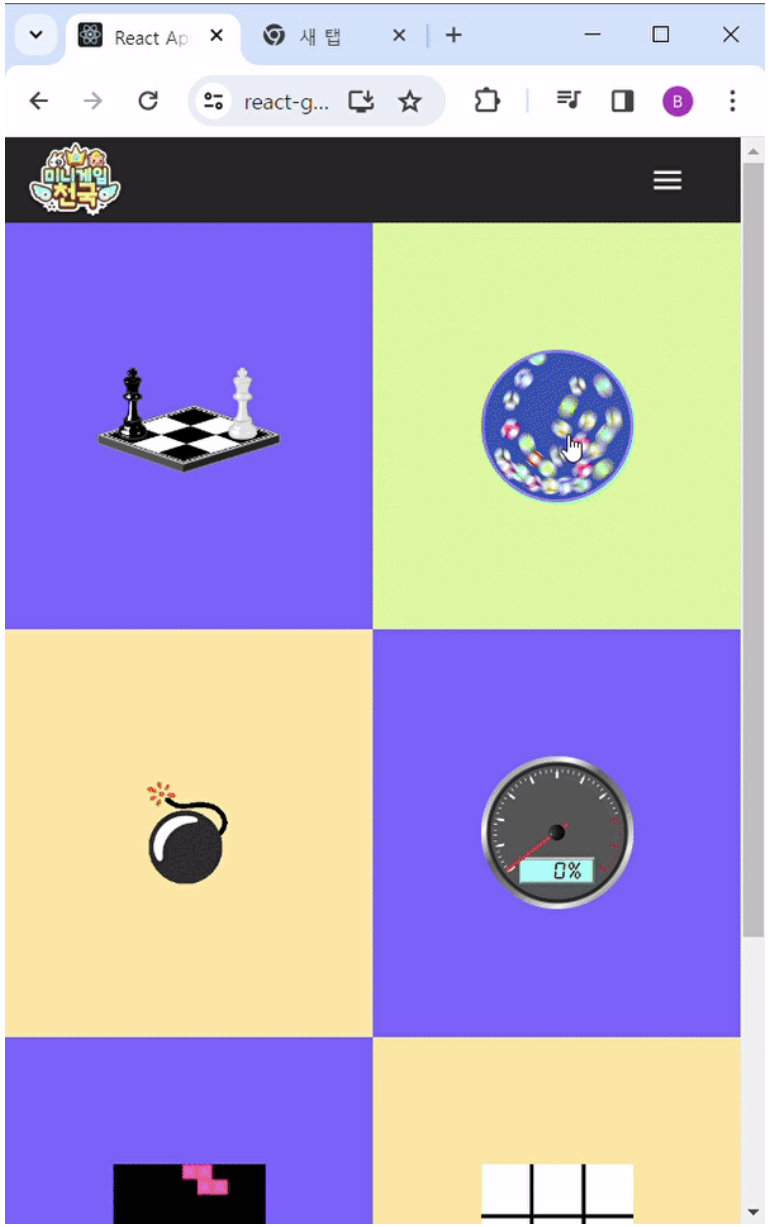
```
const ResponseCheck = () => {
  const [state, setState] = useState('waiting');
  const [message, setMessage] = useState('클릭해서 시작하세요.');
  const [result, setResult] = useState([]);
  // useState : 값이 바뀌면 return 부분이 다시 리 랜더링
  // useRef : 값이 바뀌어도 리 랜더링 x(값은 바뀌지만 화면에는 영향 x)
  // useRef는 current로 접근
  const timeout = useRef(null);
  const startTime = useRef(0);
  const endTime = useRef(0);

  const onClickScreen = useCallback(() => {
    if (state === 'waiting') {
      timeout.current = setTimeout(() => {
        setState('now');
        setMessage('지금 클릭');
        startTime.current = new Date();
      }, Math.floor(Math.random() * 1000) + 2000); // 2초~3초 랜덤
```

주요 기능 및 동작

로또 번호 뽑기

첫 화면 시 자동으로 7개의 번호가 뽑힌다.

한번 더 버튼을 클릭하면 다시 번호가 출력된다.



주요 기능 및 동작

로또 번호 뽑기

useMemo를 사용하여 복잡한 함수 결과값을 기억하고 useEffect를 사용하여 시간이 지났을 때만 이벤트가 호출되도록 하였다.

```
useEffect(() => {
    console.log('useEffect');
    // 첫번째 번호 1초후 나옴 2번째 번호 2초후 나옴... n초후 나옴
    for (let i = 0; i < winNumbers.length - 1; i++) {
      timeouts.current[i] = setTimeout(() => {
        setWinBalls((prevBalls) => [...prevBalls, winNumbers[i]]);
      }, (i + 1) * 1000);
    }
    // 보너스 번호 7번째 7초후 나옴
    timeouts.current[6] = setTimeout(() => {
      setBonus(winNumbers[6]);
      setRedo(true);
    }, 7000);
    return () => {
      timeouts.current.forEach((v) => {
        clearTimeout(v);
      });
    };
}, [timeouts.current]);
```

```
// 자식 컴포넌트에 props로 함수를 넘길때는 useCallback필수
const onClickRedo = useCallback(() =>{
  console.log('onClickRedo');
  console.log(winNumbers);
  setWinNumbers(getWinNumbers());
  setWinBalls([]);
  setBonus(null);
  setRedo(false);
  timeouts.current = [];
}, [winNumbers]);
```

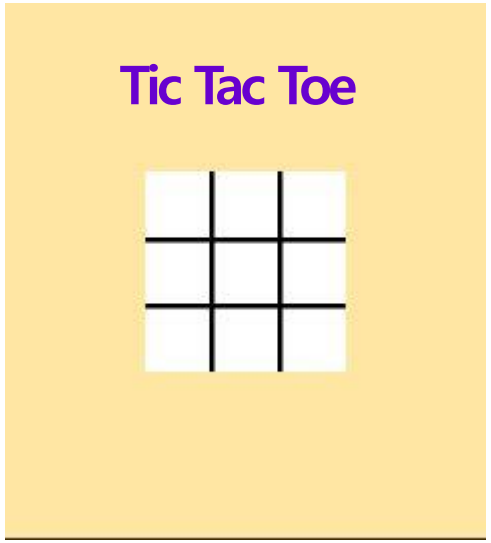
```
const Lotto = () => {
    // useMemo : 복잡한 함수 결과값을 기억
    // useRef : 일반 값을 기억
    const lottoNumbers = useMemo(() => getWinNumbers(), []);
    const [winNumbers, setWinNumbers] = useState(lottoNumbers);
    const [winBalls, setWinBalls] = useState([]);
    const [bonus, setBonus] = useState(null);
    const [redo ,setRedo] = useState(false);
    const timeouts = useRef([]);
```

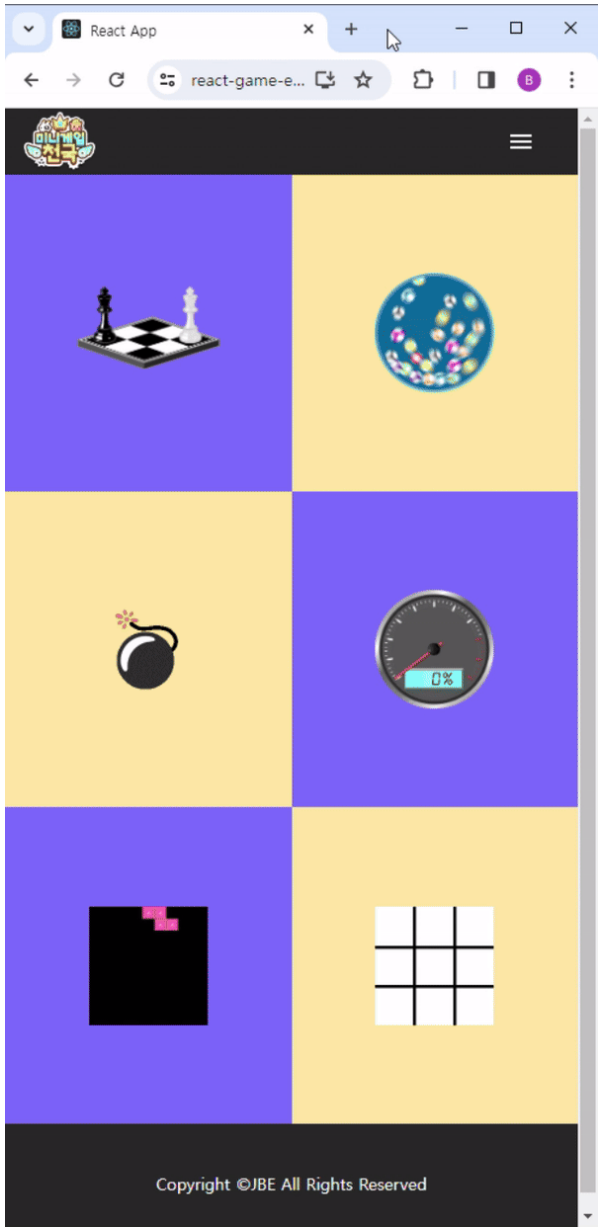
주요 기능 및 동작

Tic Tac Toe

9개의 칸이 있으며 클릭 시 O, X를 번갈아 가며

출력하고 먼저 같은 모양 3개를 빙고하면 승리한다.





주요 기능 및 동작

Tic Tac Toe

useReducer를 사용하여 action을 해석해서 state를 직접 바꿔주며 reducer에서 action을 어떻게 바꿀지 설정하여 dispatch하여 이벤트가 실행되도록 하였다.

```
// 컴포넌트에 넣는 이벤트는 useCallback
const onClickTable = useCallback(() =>{
    // dispatch 안에 들어가는 것을 action이라 부른다.
    //{type:'SET_WINNER', winner:'O'} action 객체
    // type : action의 이름
    // dispatch 하면 action을 실행한다.
    dispatch({type:SET_WINNER, winner:'O'});
}, []);
```

```
// 비동기 state따라 처리할때는 useEffect
useEffect(() => {
    const [row, cell] = recentCell;
```

```
function TicTacToe(){
    const [state, dispatch] = useReducer(reducer, initialState);
    const { tableData, turn, winner, recentCell } = state;
```

```
// action을 변수에 담아 저장
export const SET_WINNER = 'SET_WINNER';
export const CLICK_CELL = 'CLICK_CELL';
export const CHANGE_TURN = 'CHANGE_TURN';
export const RESET_GAME = 'RESET_GAME';
```

```
// action을 해석해서 state를 직접 바꿔주는 역할
// action을 dispatch할때마다 reducer 실행
// 어떻게 바꿀지는 reducer에서 써준다.
const reducer = (state, action) =>{
    switch (action.type){
        case SET_WINNER:
        // state.winner = action.winner; 이렇게 하면 안됨!
            return{
                ...state, // 데이터 불변성
                winner: action.winner,
            };
        case CLICK_CELL:{
            // 객체가 있으면 얕은 복사를 해준다.
            const tableData = [...state.tableData];
            tableData[action.row] = [...tableData[action.row]];
            tableData[action.row][action.cell] = state.turn;
            return{
                ...state,
                tableData,
                // 최근에 클릭한 셀 기억
                recentCell: [action.row, action.cell],
```

주요 기능 및 동작

지뢰 찾기

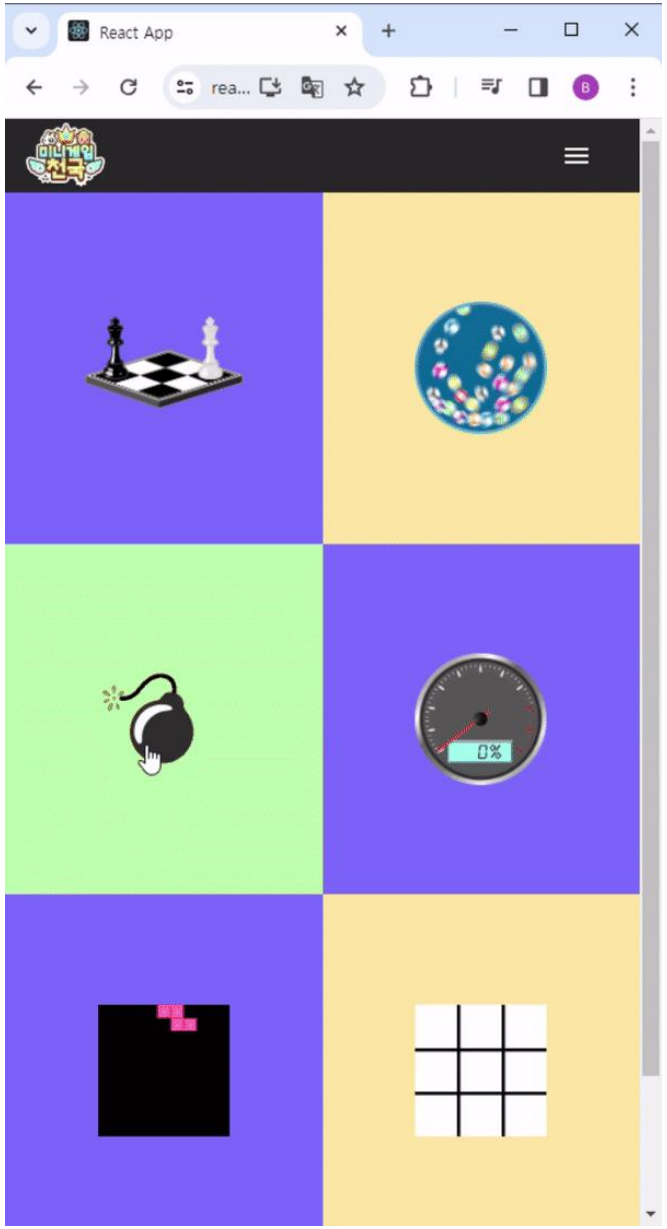
마우스 좌 클릭 : 지뢰가 없으면 칸이 채워지며

지뢰가 있으면 시간이 멈추며 그대로 게임오버가 된다.

마우스 우 클릭 : 블록에 깃발 또는 물음표가 채워진다.

지뢰를 제외한 나머지 칸을 다 채우면 승리 문구와 함께

시간 초가 나온다.



주요 기능 및 동작

지뢰 찾기

useReducer를 통해 지뢰 찾기 게임 상황을 설정하여 해당 action이 발생했을 때 dispatch하였고 Context API를 사용하여 최상위 컴포넌트에서 하위 컴포넌트로 값을 전달하였다. 마무리로 useMemo, useEffect로 최적화 하였다.

```
// <TableContext.Provider value={{tableData:state.tableData, dispatch}}>
// => 위와 같이 작성하면 MineSearch가 새롭게 리 랜더링 될 때마다 위의 객체도 새로 생긴다
// 즉 객체가 새로 생긴다는 것은 컨텍스트 api를 쓰는 자식들도 매번 새롭게 리 랜더링 된다는 뜻
// => 컨텍스트 api : 성능 최적화 하기가 힘들다
// 보완 : 캐싱

<TableContext.Provider value={value}>
<div  className='mine_container'>
    <div className='mine'>
<Form/>
<div className='timer'>{timer}</div>
<Table/>
<div className='result'>{result}</div>
    </div>
</div>
</TableContext.Provider>
```

```
function MineSearch(){
    const [state, dispatch] = useReducer(reducer, initialState);
    // 구조 분해
    const { tableData, halted, timer, result} = state;
    // useMemo 사용 : 매번 새로운 객체가 생기지 않게
    // useMemo(() => (),[바뀌는 목록])
    // cf) dispatch 함수는 바뀌지 않는다.(항상 같게 유지된다. 즉 바뀌는 목록에 추가 안해도된다.)
    const value = useMemo(() => ({tableData, halted, dispatch}), [tableData, halted]);
```

```
export const TableContext = createContext({
    tableData:[],
    halted: true,
    dispatch: () => {},
});
```

```
const reducer = (state, action) => {
    // reducer : action 발생시에 state를 어떻게 바꿀지 처리
    //action이 실행됐을때 어떤 동작을 할지 reducer를 통해 정의
    switch(action.type){
        case START_GAME:
            return{
```

주요 기능 및 동작

테트리스

방향키를 눌러 Tetraminos를 컨트롤하여

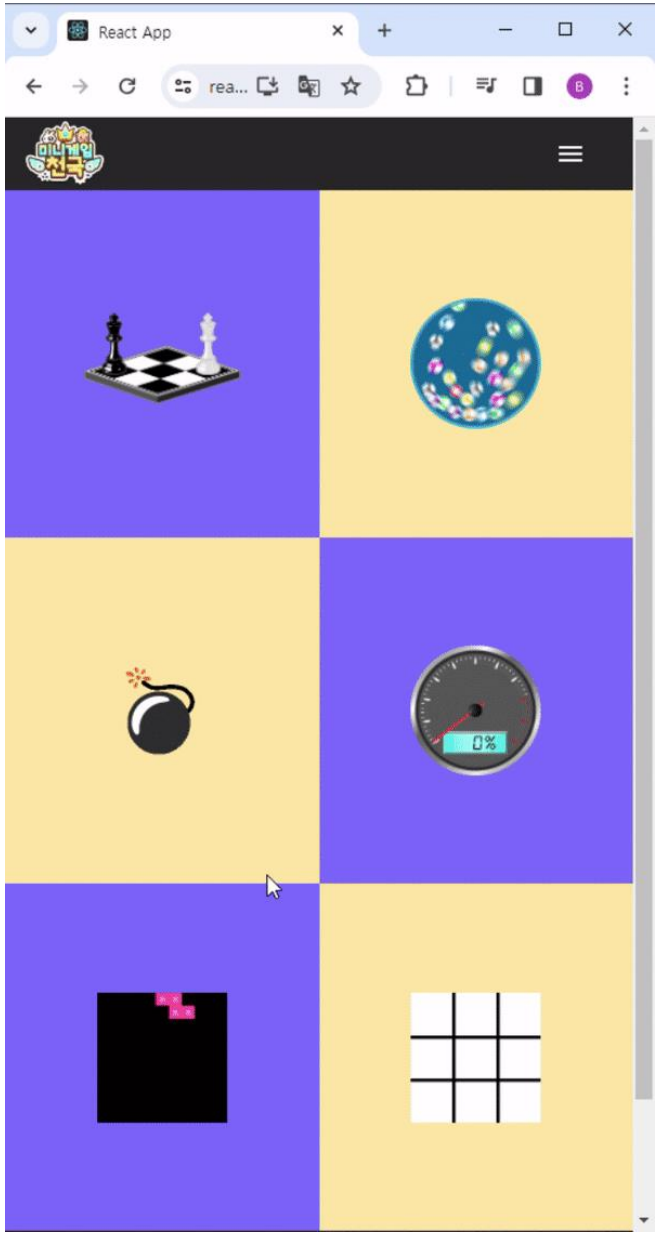
블록을 쌓고 블록이 한 줄 채워지면 사라진다.

P 버튼 클릭 : Pause된다.

Q 버튼 클릭 : 테트리스 첫 화면으로 이동한다.

Space : 맨 아래로 떨어진다.

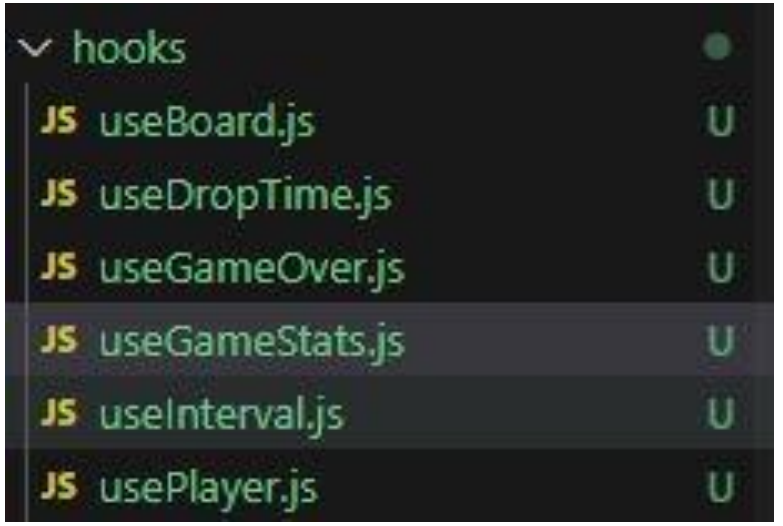
블록이 테트리스 보드를 넘어가면 게임오버가 된다.



주요 기능 및 동작

테트리스

사용자 정의 hooks를 사용하여 테트리스 게임 상황을 설정하였고 useEffect, useCallback등을 사용하여 최적화 하였다.




```
import {useGameOver} from '../hooks/useGameOver'
import Tetris from "./Tetris";


function Game({rows, columns}){
    const [gameOver, setGameOver, resetGameOver] = useGameOver();

    const start = () => {
        resetGameOver();
        console.log(`start gameOver is ${gameOver}`);
    }
```

```
import React, { useCallback, useState } from "react"

export const useGameOver = () => {
    const [gameOver, setGameOver] = useState(true);

    const resetGameOver = useCallback(() => {
        setGameOver(false);
    }, [])
    return [gameOver, setGameOver, resetGameOver];
}
```

주요 기능 및 동작

Chess

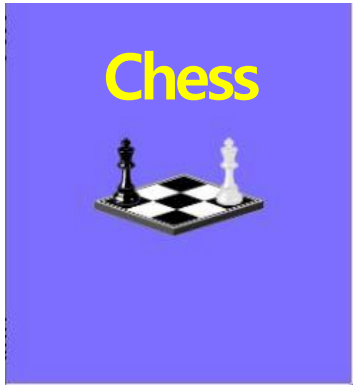
마우스를 드래그 중에 클릭한 이미지

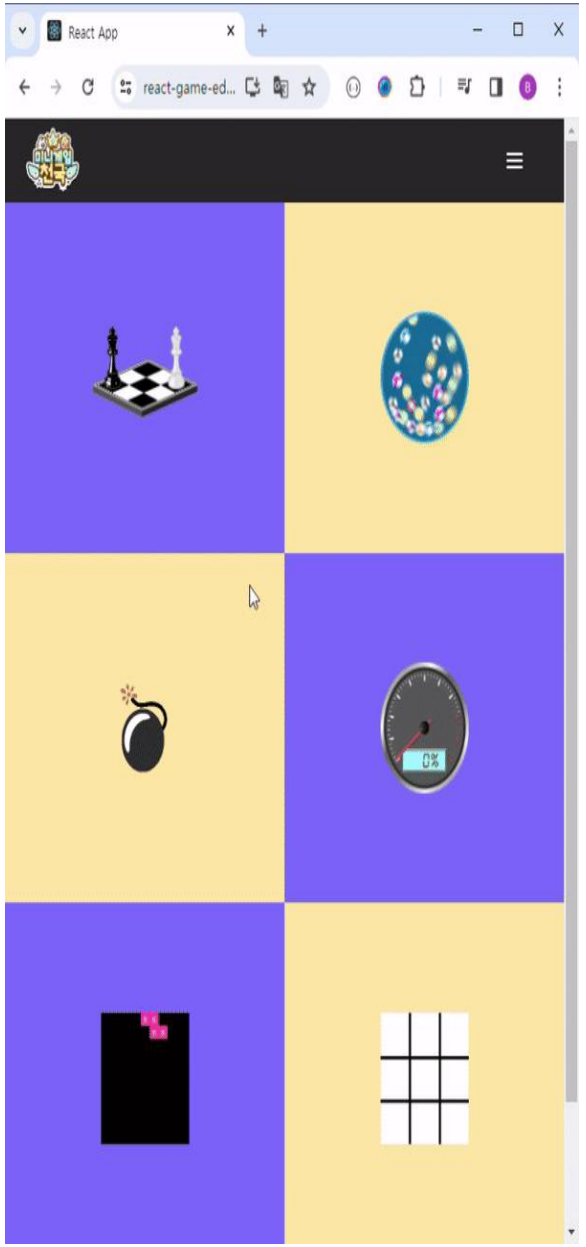
투명도 조절 및 규칙과 어긋나는 곳

이동시에 원래 자리로 돌아온다.

폰 승격 : 폰 승격 시 4개의 말이 나오고

그 중 선택하여 말을 바꿀 수 있다.





주요 기능 및 동작

Chess

React-DnD를 사용하여 Drag와 Drop 상태를 설정하였고, Chess.js와 RxJS 라이브러리를 사용하여 체스 규칙과 체스 정보를 가져와 조작 조건을 만들었다.

```
// move(변경전, 변경된 자리)
// chess.js를 통해 체스 인스턴스 업데이트
// 합법적인 이동
export function move(from, to, promotion) {
    let tempMove = { from, to }
    if (promotion) {
        tempMove.promotion = promotion
    }
    const legalMove = chess.move(tempMove)
    if (legalMove) {
        updateGame()
    }
}
```

```
const [{ isDragging }, drag, preview] = useDrag({
      type: 'piece',
      item:{type:'piece', id: `${position}_${type}_${color}`},
      collect: (monitor) => {
        return { isDragging: !!monitor.isDragging() }
      },
  })
```

```
const [, drop] = useDrop({
    accept: 'piece',
    drop: (item) => {
        const [fromPostion] = item.id.split('_')
        handleMove(fromPostion, position) // 변경된 자리
    },
})
```

```
import { BehaviorSubject } from 'rxjs';

const chess = new Chess()

export const gameSubject = new BehaviorSubject()
```

```
import { DndProvider } from 'react-dnd';
import { HTML5Backend } from 'react-dnd-html5-backend';

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <DndProvider backend={HTML5Backend}>
    <App />
    </DndProvider>
  </React.StrictMode>
);
```


You are some kind of Wonderful
C'EST LA VIE
EVER TRIED.
EVER FAILED.
NO MATTER.
TRY AGAIN.
FAIL AGAIN.
FAIL BETTER.
ШРИФТЫ В APPFORTYPE
Summer
IS IN
SIMPLICITY
music
IN ME
I want to go back to sleep
Paris bonjour
APPFORTYPE

시연


프로젝트 보러가기

감사합니다.