

# 게임 프로젝트 발표

발표자 : 신정근

# 테트리스 게임



게 임 소 개

## 목차

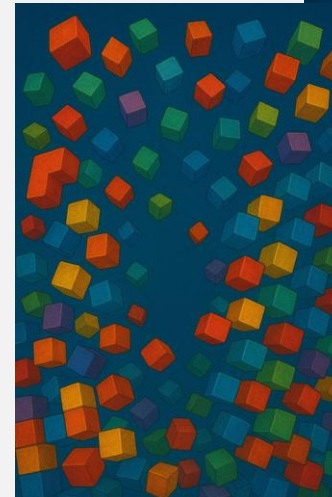
- 기획
- 디자인   구상
- 코드 설명
- 마무리

## 게임 기획

- 일반 테트리스들과 동일
- 난이도 구상 쉬움, 일반, 어려움 모드 존재
- 2분 간격으로 블록 스피드 증가 (난이도에 따라 증가율은 다름)
- 게임의 난이도 조정을 위해 각 난이도마다 특성을 추가함

## 디자인 구상

- 기본적인 디자인은 CSS를 이용하여 구상
- 그외 배경그림은 chat gpt를 이용



< = a i 그림

```
/* 스코어 */
.ServeContainerScore{
  font-family: fantasy;
  font-size: 30px;
  text-align: center;
  color: ■rgb(255, 247, 236);
  letter-spacing: 10px;
}

/* 타이머 */
.ServeContainerTimer{
  font-family: fantasy;
  font-size: 50px;
  text-align: center;
  color: ■rgb(255, 247, 236);
  margin-top: 5%;
  letter-spacing: 10px;
}

/*다음블록 표시판 컨테이너*/
div.NBwrap{
  width: 200px;
  height: 200px;
  background-color: □rgb(0, 0, 0);
  box-shadow: 0 0 10px 1px □black;
```

```
/* 백그라운드 관련 */
.main {
  background-image: url("../img/normar.png");
  background-size: cover;
  background-position: center;
  background-repeat: no-repeat;
  display: flex;
}

/*전체 컨테이너*/
div.container {
  width: 60%;
  height: 700px;
  background-color: □rgb(97, 97, 97);
  box-shadow: 0 0 5px 1px □black;
  background-position: center;
  margin: 0 auto;
  text-align: center;
}

div.Tetriswrap{
  display: flex;
  margin: 0 auto;
  gap: 3%;
```

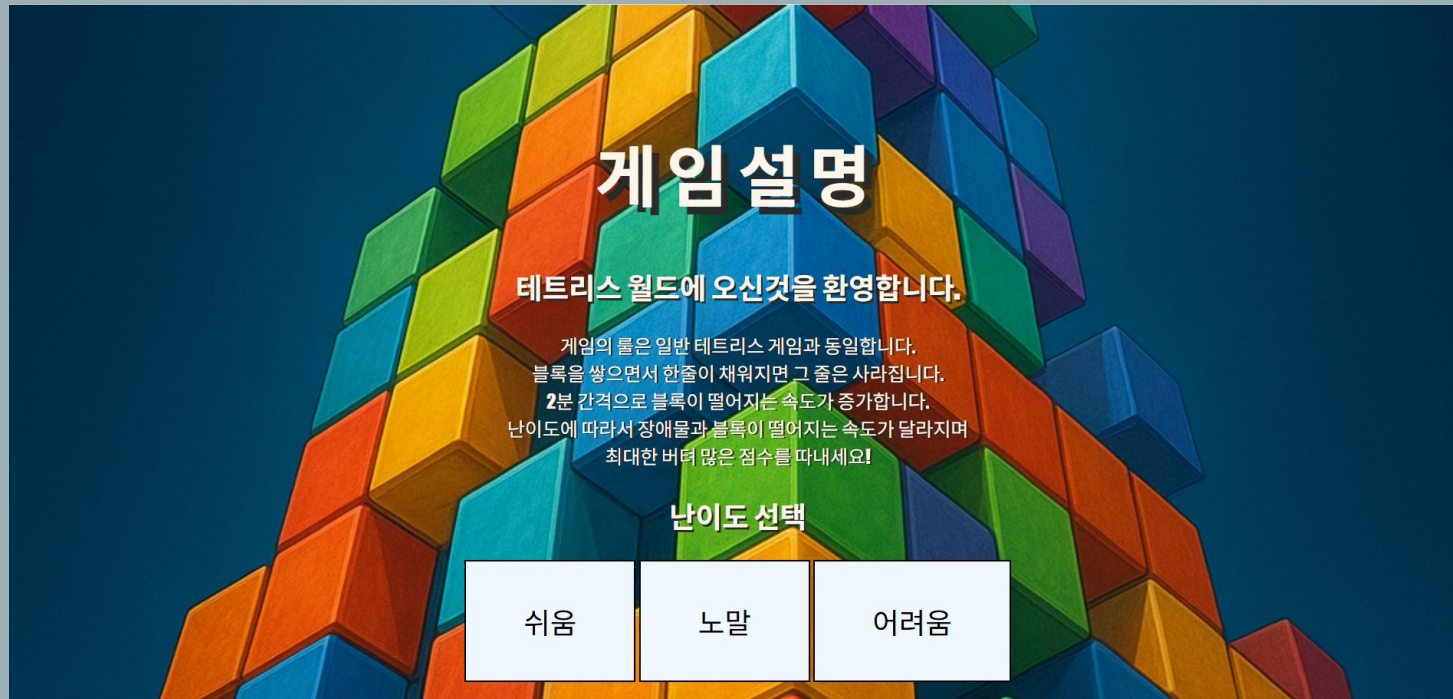
# 메인 화면

The background of the main screen is a 3D rendering of a tall, jagged tower constructed from colorful Tetris blocks. The blocks are in various colors including blue, orange, green, yellow, and purple. The tower is set against a dark blue gradient background. The text 'Tetris World' is centered over the tower in a bold, white, sans-serif font with a slight shadow effect.

**Tetris World**

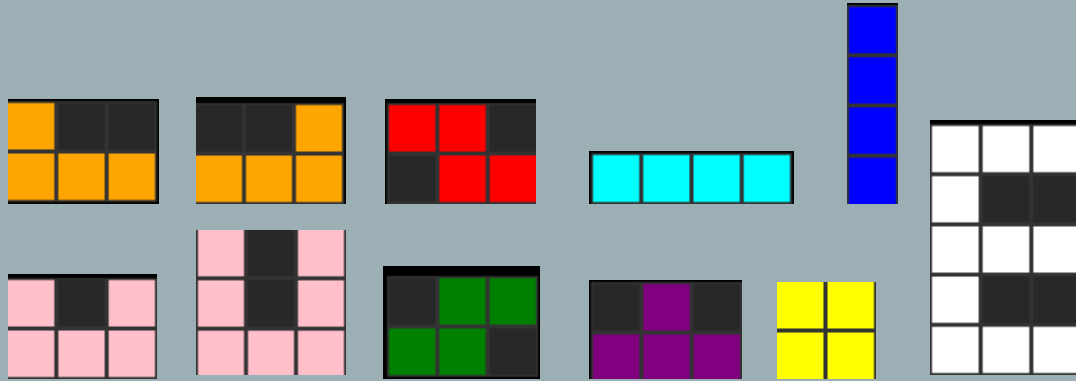
시작하기

# 난이도 선택

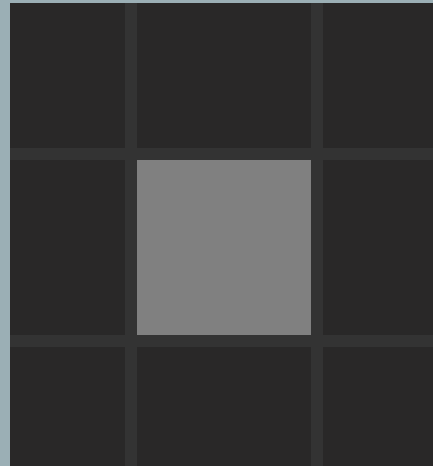
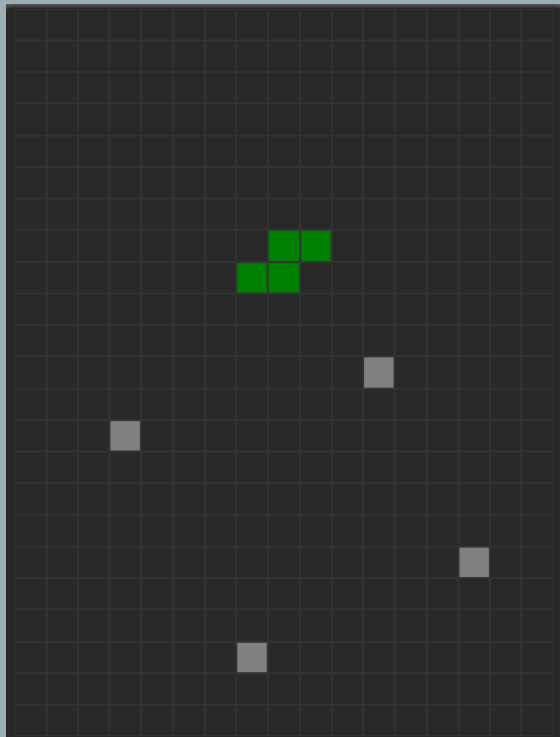


- 난이도 선택 화면에 접속하면 기본적인 게임설명이 적혀져있음.
- 난이도는 쉬움, 노말, 어려움 총 3가지로 구분되어 있음.



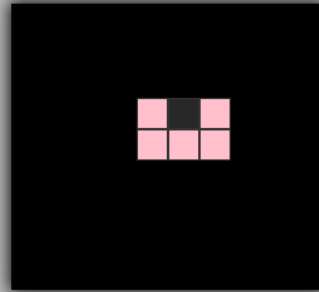
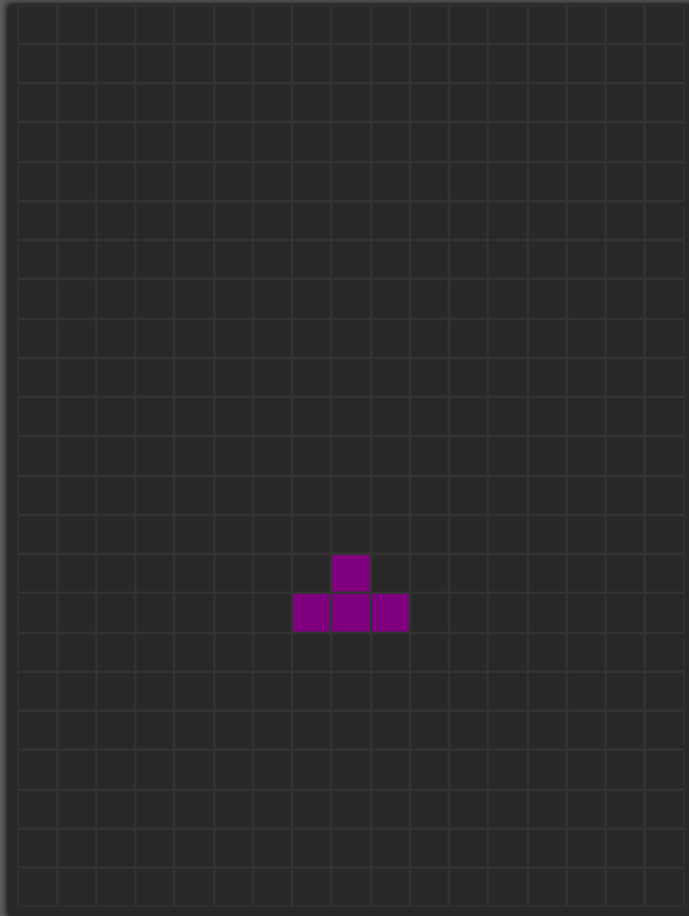


- 게임에 등장하는 테트리스 블록은 총 11개로 난이도에 따라서 등장하는 블록이 정해져 있음.



- 난이도 어려움의 경우 블록 배치  
를 방해하는 방해블록 배치(위치는 랜덤으로 배치함)

## 난이도 Normal



다음 블록

점 수 : 0

00:07

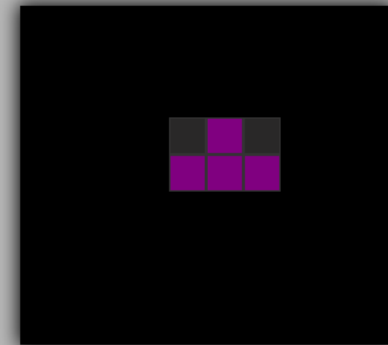
조작키

블록 이동 : ←, →

블록 회전 : ↑

블록 빠르게 떨어트리기 : ↓

- 전반적인 게임화면
- 점수, 플레이 시간, 다음에 나타날 블록, 기본 조작키등이 명시되어 있음.



다음 블록

점 수 : 100

01:24

조작키

블록 이동 : ←, →

블록 회전 : ↑

블록 빠르게 떨어트리기 : ↓

이 페이지 내용:

게임 오버! 내 점수 : 100 점 다시 하시겠습니까?

확인

취소

- **확인을 누를 경우 게임 재시작**

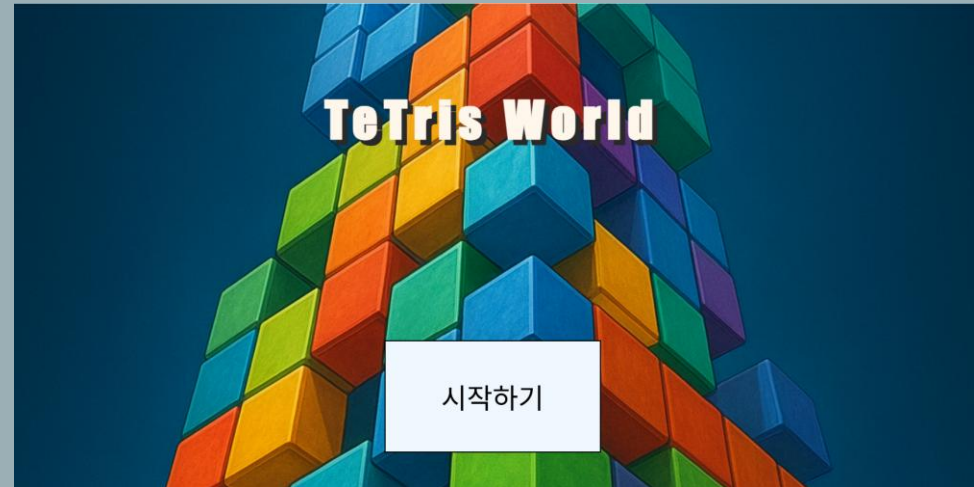


이 페이지 내용:

메인화면으로 돌아갑니다.

확인

- **취소를 누를 경우 알림 후  
메인화면 복귀**



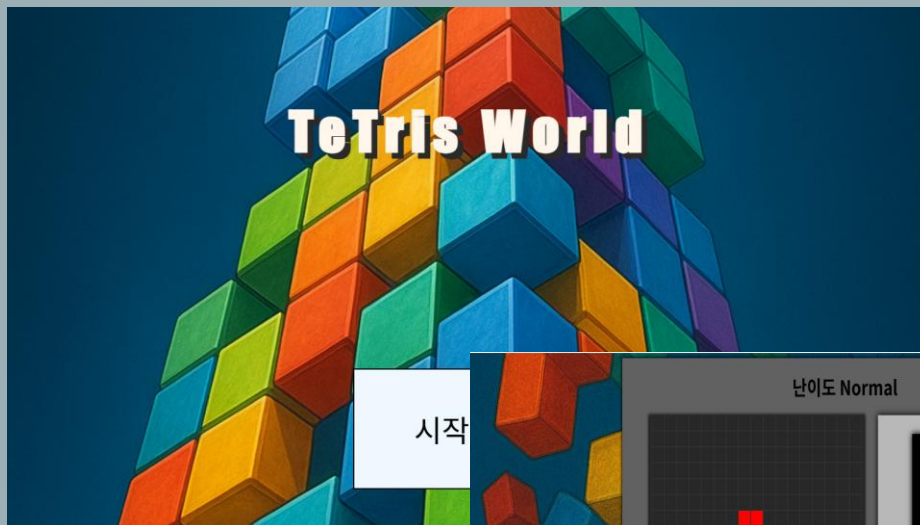
# 코드 설명

- 사용 언어 html, css, JavaScript

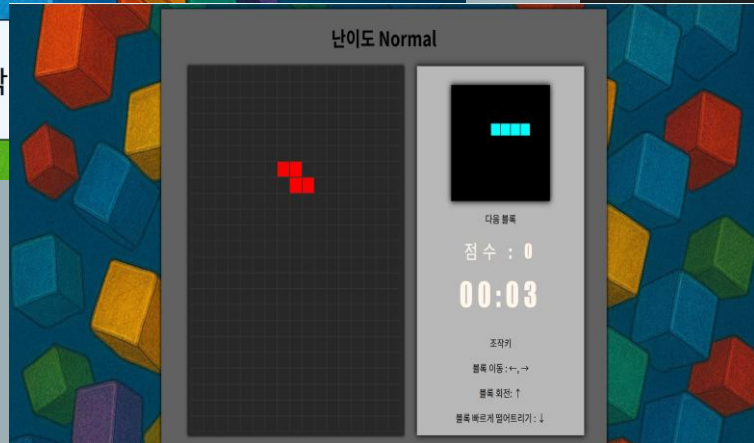
```
css > # game.css > ServeContainerTimer
4
5 /* 백그라운드 관련 */
6 .main {
7     background-image: url("../img/norwar.png");
8     background-size: cover;
9     background-position: center;
10    background-repeat: no-repeat;
11    display: flex;
12 }
13
14 /*전체 컨테이너나*/
15 div.container {
16     width: 608px;
17     height: 700px;
18     background-color: #rgb(97, 97, 97);
19     box-shadow: 0 0 5px 1px #black;
20     background-position: center;
21     margin: 0 auto;
22     text-align: center;
23 }
24 div.Tetriswrap{
25     display: flex;
26     margin: 0 auto;
27     gap: 3%;
28 }
29 /*테트리스 메인 컨테이너*/
30 div.MainContainer{
31     width: 400px;
32 }
33
34 <DOCTYPE html>
35 <html lang="en">
36
37 <head>
38     <meta charset="UTF-8">
39     <meta name="viewport" content="width=device-width, initial-scale=1.0">
40     <title>Tetris</title>
41     <link href="../css/game.css" rel="stylesheet" type="text/css">
42 </head>
43
44 <body class="main">
45     <div class="container">
46         <h1 id = "check">난이도 Hard</h1>
47         <div class="Tetriswrap">
48             <div class="MainContainer">
49                 <div class="MainContainerIn"></div>
50             </div>
51             <div class="ServeContainer">
52                 <div class="NBWrap">
53                     <div class="nextBlockContainer"></div>
54                 </div>
55                 <p>다음 블록</p>
56                 <div class="ServeContainerScore">점수 : 0</div>
57                 <div id="Timer" class="ServeContainerTimer">00:00</div>
58                 <div style="margin-top: 10px;">조작키
59                     <p>블록 이동 : ←, → </p>
60                     <p>블록 회전: ↑</p>
61                     <p>블록 빠르게 떨어트리기 : ↓ </p>
62                 </div>
63             </div>
64         </div>
65     </body>
66 </html>
```

```
JavaScript > Tetris.normal.js > document.addEventListener("DOMContentLoaded") callback > DrawBoard
1 document.addEventListener("DOMContentLoaded", () => { // DOMContentLoaded html이 완전히 로딩된 뒤 실행
2     const MainContainer = document.querySelector(".MainContainerIn"); // html 요소를 찾을 수 있게함
3     const NextBlockContainer = document.querySelector(".nextBlockContainer");
4     const ServeContainerScore = document.querySelector(".ServeContainerScore");
5     const row = 23; // 세로칸
6     const col = 17; // 가로칸
7     const cellSize = 25; // 각 칸 크기(px)
8     let nowBlock;
9     let nextBlock;
10    let score = 0; // 점수
11    let dropSpeed = 500;
12    let gameInterval = setInterval(MovingBlock, dropSpeed); // 낙하속도
13
14
15    // 2차원 배열 Array.from은 배열을 만드는 함수, length : row => 길이가 row인 배열을 만들어라
16    // () => Array(col).fill(0));은 출력함수
17    const board = Array.from({ length: row }, () => Array(col).fill(0));
18
19    // 화면(크지무늬) 그리는 함수
20    function DrawBoard() {
21        MainContainer.innerHTML = "";
22        for (let r = 0; r < row; r++) {
23            for (let c = 0; c < col; c++) {
24                const cell = document.createElement("div"); // <div></div>생성
25                cell.style.width = `${cellSize}px`;
26                cell.style.height = `${cellSize}px`;
27                cell.style.border = "1px solid #333";
28                cell.style.boxSizing = "border-box"; // 테두리도 칸 크기에 포함하도록
```

# 일반 화면 생성



```
1 <!DOCTYPE html>
2 <html lang="en">
3
4 <head>
5   <meta charset="UTF-8">
6   <meta name="viewport" content="width=device-width, initial-scale=1.0">
7   <title>TeTris</title>
8   <link href="../../css/main.css" rel="stylesheet" type="text/css">
9 </head>
10
11 <body class="main">
12   <div>
13     <div>
14       <h1>TeTris World</h1>
15       <div class="maindiv"><a href="../../html/SelectDifficult.html" class="btn">시작하기</a></div>
16     </div>
17   </div>
18 </body>
19
20 </html>|
```



```
<body class="main">
  <div class="container">
    <h1 id = "check">난이도 Hard</h1>
    <div class="Tetriswrap">
      <div class="MainContainer">
        <div class="MainContainerIn"></div>
      </div>
      <div class="ServeContainer">
        <div class="NBwrap">
          <div class="nextBlockContainer"></div>
        </div>
        <p>다음 블록</p>
        <div class="ServeContainerScore">점수 : 0</div>
        <div id="Timer" class="ServeContainerTimer">00:00</div>
        <div style="margin-top: 10px;">조작키
        <p>블록 이동 : ←, → </p>
        <p>블록 회전 : ↑ </p>
        <p>블록 빠르게 떨어뜨리기 : ↓ </p>
      </div>
    </div>
  </div>
```

```

1  <!DOCTYPE html>
2  <html lang="en">
3
4  <head>
5      <meta charset="UTF-8">
6      <meta name="viewport" content="width=device-width, initial-scale=1.0">
7      <title>TeTris</title>
8      <link href="../css/main.css" rel="stylesheet" type="text/css">
9  </head>
10
11  <body class="main">
12      <div>
13          <div>
14              <h1>TeTris World</h1>
15              <div class="maindiv"><a href="../html/Sel
16          </div>
17      </div>
18  </body>
19
20  </html>

```

```

/* 배경그라운드 관련 */
.main {
    background-image: url("../img/noramar.png");
    background-size: cover;
    background-position: center;
    background-repeat: no-repeat;
    display: flex;
}

/*전체 컨테이너*/
div.container {
    width: 60%;
    height: 700px;
    background-color: #rgb(97, 97, 97);
    box-shadow: 0 0 5px 1px #black;
    background-position: center;
    margin: 0 auto;
    text-align
}

/* 버튼 관련 */
a.btn:link, a.btn:visited {
    background-color: #aliceblue;
    color: #black;
    text-align: center;
    text-decoration: none;
    display: inline-block;
    padding: 50px 80px;
    border: 2px solid #black;
    font-family: fantasy;
    font-size: 40px;
}

a.btn:hover,
a.btn:active {
    background-color: #aquamarine;
}

a.btn2:link, a.btn2:visited {
    background-color: #aliceblue;
    color: #black;
    text-align: center;
    text-decoration: none;
    display: inline-block;
    padding: 40px 60px;
}

```

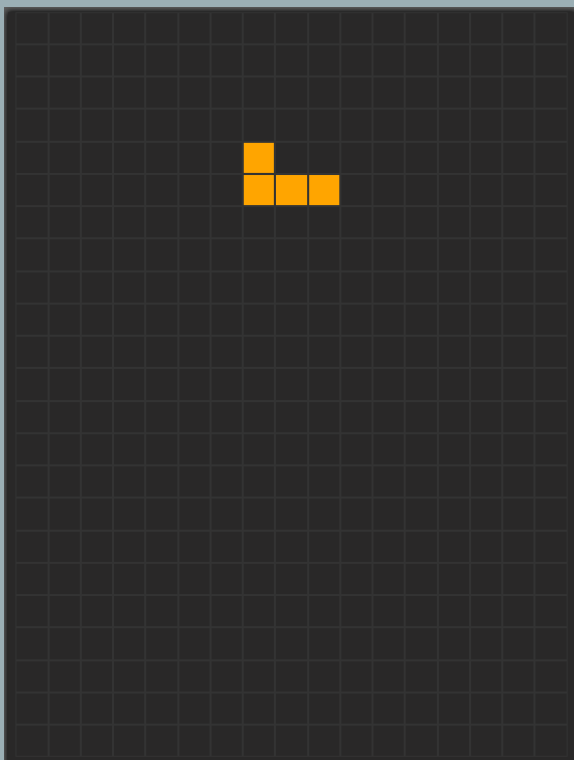
```

<body class="main">
  <div class="container">
    <h1 id = "check">난이도 Hard</h1>
    <div class="Tetriswrap">
      <div class="MainContainer">
        <div class="MainContainerIn"></div>
      </div>
      <div class="ServeContainer">
        <div class="NBwrap">
          <div class="nextBlockContainer"></div>
        </div>
        <p>다음 블록</p>
        <div class="ServeContainerScore">점수 : 0</div>
        <div id="Timer" class="ServeContainerTimer">00:00</div>
        <div style="margin-top: 10px;">조작키
          <p>블록 이동 : ←, → </p>
          <p>블록 회전 : ↑</p>
          <p>블록 빠르게 떨어트리기 : ↓ </p>
        </div>
      </div>
    </div>
  </div>

```

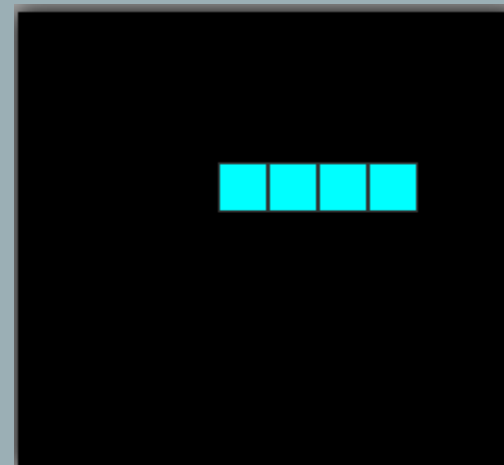
기본적인 배경 디자인은  
CSS언어 사용,  
class 속성들을 적용해,  
div 태그들을 이용하여  
간단하게 구현

# 자바스크립트 기능 설명



점 수 : 0

01:13



다음 블록

```
document.addEventListener("DOMContentLoaded", () => { // DomContentLoaded html이 완전히 로드된 뒤 실행
```

**D o m C o n t e n e t L o a d e d :**

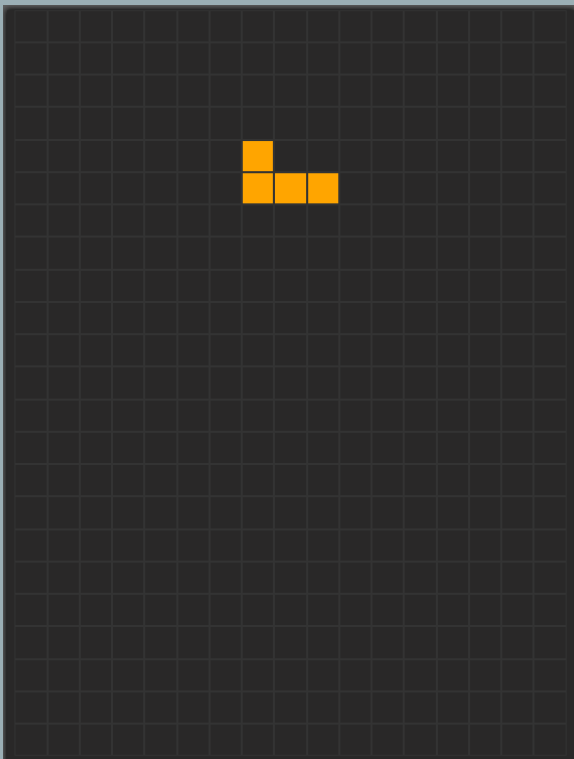
H t m l 이 로드되기전에 자바스크립트가 실행되어 오류가 발생하는 것을 방지하기 위해 사용

```
const MainContainer = document.querySelector(".MainContainerIn"); // html 요소들을 찾을 수 있게함
const NextBlockContainer = document.querySelector(".nextBlockContainer");
const ServeContainerScore = document.querySelector(".ServeContainerScore");
const row = 23; // 세로칸
const col = 17; // 가로칸
const cellSize = 25; // 각 칸 크기(px)
let nowBlock;
let nextBlock;
let score = 0; // 점수
let dropSpeed = 500;
let gameInterval = setInterval(MovingBlock, dropSpeed); // 낙하속도
```

document.querySelector() 를 이용해 클래스 요소들을 찾고 필요한 변수들 선언



# 격자무늬 화면 생성



```
// 2차원 배열 Array.from은 배열을 만드는 함수, length : row => 길이가 row인 배열을 만들어라
// () => Array(col).fill(0));은 콜백함수
const board = Array.from({ length: row }, () => Array(col).fill(0));

// 화면(격자무늬) 그리는 함수
function DrawBoard() {
  MainContainer.innerHTML = "";
  for (let r = 0; r < row; r++) {
    for (let c = 0; c < col; c++) {
      const cell = document.createElement("div"); // <div></div>생성
      cell.style.width = `${cellSize}px`;
      cell.style.height = `${cellSize}px`;
      cell.style.border = "1px solid #333";
      cell.style.boxSizing = "border-box"; // 테두리도 칸 크기에 포함하도록
      // 칸 색깔 정하기 ===0이 참이면 #292828 불이면 board[r][c]색 사용
      cell.style.backgroundColor = board[r][c] === 0 ? "#292828" : board[r][c];
      cell.style.float = "left"; // 왼쪽으로 이어 붙이게 가로정렬
      MainContainer.appendChild(cell); // MainContainer에 적용
    }
  }
  MainContainer.style.height = `${row * cellSize}px`; // 칸 높이
  MainContainer.style.width = `${col * cellSize}px`; // 칸 넓이
  MainContainer.style.position = "relative"; // 상대적인 위치로 지정, 화면크기 고려해 칸위치 고정
}
```

## 테트리스 격자무늬판은 2차원 배열을 이용하여 표현

```
// 2차원 배열 Array.from은 배열을 만드는 함수, length : row => 길이가 row인 배열을 만들어라
// () => Array(col).fill(0));은 콜백함수
const board = Array.from({ length: row }, () => Array(col).fill(0));


// 화면(격자무늬) 그리는 함수
function DrawBoard() {
  MainContainer.innerHTML = "";
  for (let r = 0; r < row; r++) {
    for (let c = 0; c < col; c++) {
      const cell = document.createElement("div"); // <div></div>생성
      cell.style.width = `${cellSize}px`;
      cell.style.height = `${cellSize}px`;
      cell.style.border = "1px solid #333";
      cell.style.boxSizing = "border-box"; // 테두리도 칸 크기에 포함하도록
      // 칸 색깔 정하기 ==0이 참이면 #292828 붙이면 board[r][c]색 사용
      cell.style.backgroundColor = board[r][c] === 0 ? "#292828" : board[r][c];
      cell.style.float = "left"; // 왼쪽으로 이어 붙이게 가로정렬
      MainContainer.appendChild(cell); // MainContainer에 적용
    }
  }
  MainContainer.style.height = `${row * cellSize}px`; // 칸 높이
  MainContainer.style.width = `${col * cellSize}px`; // 칸 넓이
  MainContainer.style.position = "relative"; // 상대적인 위치로 지정, 화면크기 고려해 칸위치 고정
}
```

```
// 2차원 배열 Array.from은 배열을 만드는 함수, length : row => 길이가 row인 배열을 만들어라  
// () => Array(col).fill(0));은 콜백함수  
const board = Array.from({ length: row }, () => Array(col).fill(0));
```

Array.from 함수를 이용하여 세로 길이가 row, 가로 길이가 col인 2차원 배열을 선언

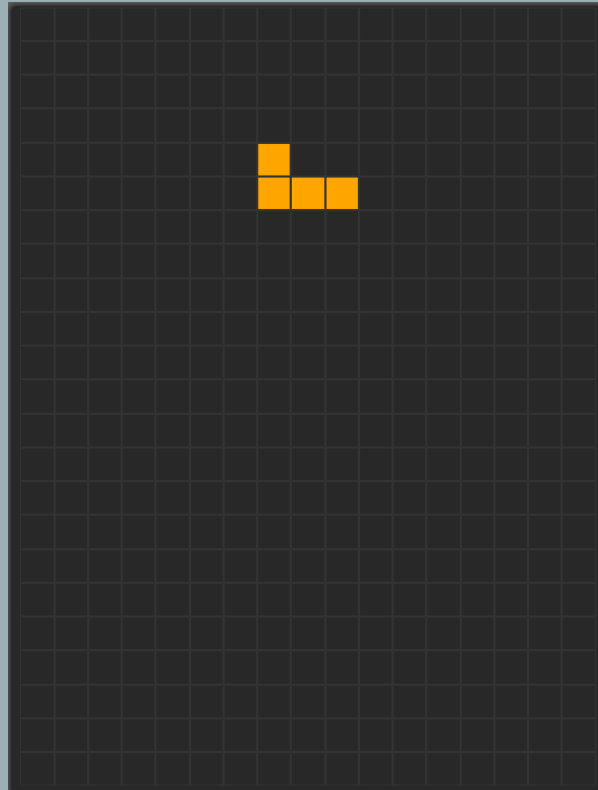
```
for (let r = 0; r < row; r++) {  
  for (let c = 0; c < col; c++) {  
    const cell = document.createElement("div"); // <div></div>생성
```

2중 for문을 이용하여 <div>를 계속해서 선언(세로가 r줄 가로가 c줄인 화면을 만들기 위함)

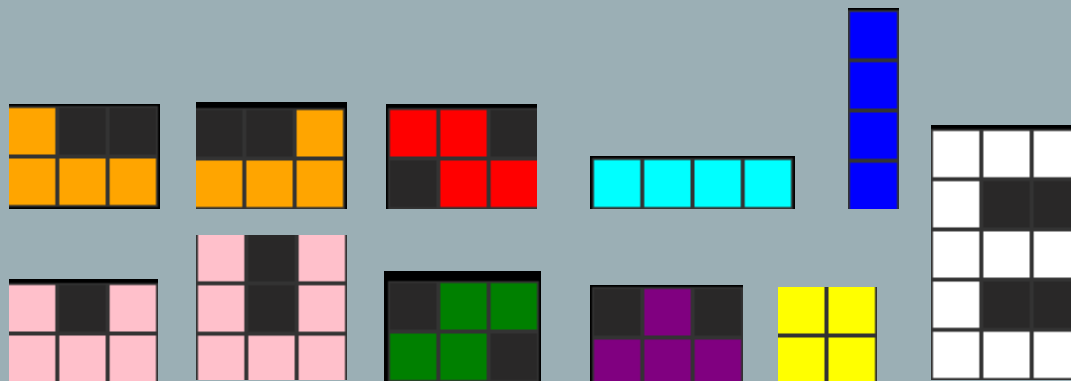
```
cell.style.border = "1px solid  #333";  
cell.style.boxSizing = "border-box"; // 테두리도 칸 크기에 포함하도록
```

Border 테두리선에 의해 크기가 커지는 것을 방지하기 위해 border-box를 사용, \${cellsize} 크기가 선언한 그대로 반영될 수 있게함

결과물



# 테트리스 블록관련



# 1. 블록 표현

```
// 블록모양
const Tetbox = {
  I: [[1, 1, 1, 1]],
  I2: [[1],
  [1],
  [1],
  [1]],
  T: [
    [0, 1, 0],
    [1, 1, 1]
  ],
  O: [
    [1, 1],
    [1, 1]
  ],
  S: [
    [0, 1, 1],
    [1, 1, 0]
  ],
  Z: [
    [1, 1, 0],
    [0, 1, 1]
  ],
  L: [
// 블록 색깔
const COLORS = {
  I: "cyan",
  I2: "blue",
  T: "purple",
  O: "yellow",
  S: "green",
  Z: "red",
  L: "orange",
  L2: "orange",
  U: "pink",
  U2: "pink",
  E: "white"
};
```

블록 모양선언(2차원 배열), 색깔 적용.

```
// 블록 생성 및 초기 위치
let currentBlock = {
  shape: Tetbox.I,      //블록 모양 가져오기
  row: 0,               // 생성 초기 위치
  col: Math.floor(col / 2) - 1, // 가운데 정렬
  color: COLORS.I       // 블록 색깔 가져오기
};
```

currentBlock 객체 선언(블록 정보를 담을 수 있음)하여 블록 모양을 가져오고  
블록이 생성 될때 위치할 초기 위치를 선언.

```
// 블록을 화면에 표현
function drawBlock() {
  const { shape, row, col, color } = currentBlock; // 객체 안에서 필요한 값만 꺼낼수 있게하는 문법
  for (let r = 0; r < shape.length; r++) {
    for (let c = 0; c < shape[r].length; c++) {
      if (shape[r][c]) {
        board[row + r][col + c] = color;
      }
    }
  }
}
```

2중 for문을 이용해 모든칸을 확인하여 shape(r)(c)가 true면(칸에 블록이  
감지되면) color를 적용해 테트리스 블록을 화면에 출력.

## 2. 블록 충돌 검사

```
// 충돌 검사
function checkCollision() {
  const { shape, row: br, col: bc } = currentBlock;
  for (let r = 0; r < shape.length; r++) {
    for (let c = 0; c < shape[r].length; c++) {
      if (shape[r][c]) { // 블록이 있으면
        const newrow = br + r;
        const newcol = bc + c;
        if (
          newrow >= row || //바닥에 닿음
          newcol < 0 || newcol >= col || // 벽에 닿음
          board[newrow][newcol] !== 0 // 다른 블록에 닿음
        ) {
          return true;
        }
      }
    }
  }
  return false;
}
```



```
for (let r = 0; r < shape.length; r++) {  
  for (let c = 0; c < shape[r].length; c++) {  
    if (shape[r][c]) { // 블록이 있으면
```

2중 for문으로 모든 칸을 확인해 if문 선언(충돌이 있음 | 없으면)

```
const newrow = br + r;  
const newcol = bc + c;
```

보드판상 현재 블록이 있는 위치값(newrow, newcol)을 구한다.

(r,c = 블록 좌표(TeTbox 배열값) br,bc = 블록이 보드위에서 시작한위치)

```
if (  
    newrow >= row || //바닥에 닿음  
    newcol < 0 || newcol >= col || // 벽에 닿음  
    board[newrow][newcol] != 0 // 다른 블록에 닿음  
) {  
    return true;  
}
```

그 후 조건문을 이용하여 충돌조건을 정의해 true를 반환

### 3. 블록 낙하

```
// 블록 낙하
function MovingBlock() {
  ICBlockSpeed() // 낙하속도 변경 호출
  // 기존 위치 지우기
  for (let r = 0; r < currentBlock.shape.length; r++) {
    for (let c = 0; c < currentBlock.shape[r].length; c++) {
      if (currentBlock.shape[r][c]) { // 블록이 있으면
        board[currentBlock.row + r][currentBlock.col + c] = 0; // 블록 이동시 이전위치 0으로 초기화
      }
    }
  }
  // 한 칸 아래로
  currentBlock.row++;
}
```

기존 블록이 있는 곳을 0으로 바꾸고 currentBlock의 row값을 ++함으로 블록이 내려가는 듯한 느낌을 준다.

```
// 충돌 체크
if (checkCollision()) {
  currentBlock.row--;
  drawBlock();
  ClearBlock();
  SpawnBlock();
}
```

이후 충돌 체크를 하여 블록이 계속해서 내려가는 것을 방지(선언 안할시 에러발생)

## 4. 블록 지우기, 점수 계산

```
// 한줄 완료시 지워지는 기능, 점수 계산
function ClearBlock() {
  let linesCleared = 0; // 한번에 지운 라인 수
  for (let r = row - 1; r >= 0; r--) { // 아래줄 부터 체크를 하기위해 보드 아래쪽 row-1 부터 위쪽으로
    if (board[r].every(cell => cell !== 0)) { // 한줄이 모두 1인가 확인
      board.splice(r, 1); // r 번째 행 제거
      // 맨위 빈줄 추가, 배열 구조상 위쪽에 새로운 요소를 넣음 이미 있는 배열들은 밀리기에 자동으로 블록들은 내려가짐
      board.unshift(Array(col).fill(0));
      linesCleared++; // 지운줄 갯수
      r++; // 안하면 여러줄 지울시 한번에 안지워짐
    }
  }
  // 점수 계산
  if (linesCleared >= 5) {
    score += 400;
  }
  switch (linesCleared) {
    case 1: score += 50; break;
    case 2: score += 90; break;
    case 3: score += 150; break;
    case 4: score += 250; break;
  }
  ServeContainerScore.textContent = `점수 : ${score}`;
}
```

```
let linesCleared = 0; // 한번에 지운 라인 수
for (let r = row - 1; r >= 0; r--) { // 아래줄 부터 체크를 하기위해 보드 아래쪽 row-1 부터 위쪽으로
  if (board[r].every(cell => cell !== 0)) { // 한줄이 모두 1인가 확인
    board.splice(r, 1); // r 번째 행 제거
    // 맨위 빈줄 추가, 배열 구조상 위쪽에 새로운 요소를 넣음 이미 있는 배열들은 밀리기에 자동으로 블록들은 내려가짐
    board.unshift(Array(col).fill(0));
    linesCleared++; // 지운줄 갯수
    r++; // 안하면 여러줄 지울시 한번에 안지워짐
  }
}
```

linesCleared선언(점수체크를 하기위함)

맨 아래줄부터 체크하기위해 for문을 작성하고 row가 0이 아닌지 체크.

!==0이 true면 splice(r,1)(r값이 1인줄 모드제거)실행 후 unshift함수에 의해 배열 맨 앞 새로운 요소 추가(기존 값들은 뒤로 밀려남), 다채워진 줄은 없어지고 블록들이 밑으로 내려가는 것을 표현할 수 있다.

```
// 점수 계산
if (linesCleared >= 5) {
    score += 400;
}
switch (linesCleared) {
    case 1: score += 50; break;
    case 2: score += 90; break;
    case 3: score += 150; break;
    case 4: score += 250; break;
}
ServeContainerScore.textContent = `점수 : ${score}`;
```

방금전 선언한 linesCleared를 이용 if, switch문을 사용해 점수 계산

## 5. 블록 이동 및 회전

```
// 블록 좌우 이동
function ControlBlock(dir) {    // dir = 문서의 텍스트 방향성 왼쪽 or 오른쪽을 나타내는 속성
    for (let r = 0; r < currentBlock.shape.length; r++) {
        for (let c = 0; c < currentBlock.shape[r].length; c++) {
            if (currentBlock.shape[r][c]) {
                board[currentBlock.row + r][currentBlock.col + c] = 0;
            }
        }
    }
    currentBlock.col += dir;

    // 벽 충돌 검사
    if (checkCollision()) {
        currentBlock.col -= dir;
    }
}
```

요소 내용의 텍스트 방향을 결정하는 dir속성을 이용해 좌우 이동을 할수있게 만든다.

```

// 블록 회전
function rotationBlock() {
  const { shape, row: br, col: bc } = currentBlock;
  const rotated = shape[0].map((_, i) => shape.map(row => row[i]).reverse());

  for (let r = 0; r < shape.length; r++) {
    for (let c = 0; c < shape[r].length; c++) {
      if (shape[r][c]) {
        board[currentBlock.row + r][currentBlock.col + c] = 0;
      }
    }
  }
  currentBlock.shape = rotated;

  // 충돌시 취소
  if (checkCollision()) {
    currentBlock.shape = shape;
  }
  drawBlock();
  DrawBoard();
}

```

블록 회전의 경우 reverse()함수를 이용해 배열을 돌려 표현.



## Keydown 이벤트문으로 조작키 지정

```
// 블록 조작
document.addEventListener("keydown", (e) => {
  switch (e.key) {
    case "ArrowLeft": // ←
      ControlBlock(-1);
      break;
    case "ArrowRight": // →
      ControlBlock(1);
      break;
    case "ArrowDown": // ↓
      MovingBlock();
      break;
    case "ArrowUp": // ↑
      rotationBlock();
      break;
  }
})
```

# 결과물



그 외 기능

01:55



다음 블록

01:55

시계 : 플레이어가 게임을 얼마나 버티고  
있는지 확인하기 위함



블록 미리보기 : 다음 블록이 무엇이 나올지  
확인하여 플레이어가 미리 대비할 수 있도록  
하기 위함

다음 블록

## 시계

```
let second = 0; // 시작값

const timerElement = document.getElementById("Timer");

setInterval(() =>{ // 일정 시간마다 반복 실행되는 함수
    second++;

    const min = Math.floor(second/60); //분
    const sec = second % 60;          //초

    const timermin = min < 10 ? "0" + min : min;    // 조건식 참이면 "0" + min입력 아니면 min 출력
    const timersec = sec < 10 ? "0" + sec : sec;    // 조건식 참이면 "0" + sec입력 아니면 sec 출력

    timerElement.textContent = `${timermin}:${timersec}`;
}, 1000); // 1000 => 1s 마다 실행
```

setInterval함수(일정 시간마다 반복되는 실행 함수)를 이용

01:55

```
const min = Math.floor(second/60); //분  
const sec = second % 60;          //초
```

```
const timermin = min < 10 ? "0" + min : min;    // 조건식 참이면 "0" + min입력 아니면 min 출력  
const timersec = sec < 10 ? "0" + sec : sec;    // 조건식 참이면 "0" + sec입력 아니면 sec 출력
```

Min, sec 변수 선언(min변수에 소수점 제거를 위해 Math.floor()사용),  
Timermin, timersec = 조건식 사용 시계를 XX : XX로 표현하기 위함

## 블록 미리보기

```
// 다음 블록 표시
function ShowNextBlock() {
    NextBlockContainer.innerHTML = "";
    const size = 20;
    for (let r = 0; r < nextBlock.shape.length; r++) { // nextBlock.shape을 가져와서 다음 블록이 무엇인지 확인
        for (let c = 0; c < nextBlock.shape[r].length; c++) {
            const cell = document.createElement("div");
            cell.style.width = `${size}px`;
            cell.style.height = `${size}px`;
            cell.style.border = "1px solid □#333";
            cell.style.boxSizing = "border-box";
            cell.style.backgroundColor = nextBlock.shape[r][c] ? nextBlock.color : "□#292828";
            cell.style.float = "left";
            NextBlockContainer.appendChild(cell);
        }
        const br = document.createElement("div");
        br.style.clear = "both";
        NextBlockContainer.appendChild(br);
    }
}
```

스크립트에 선언한 nextBlock 객체를 가져와 테트리스 화면을 만들때와 동일한 방법으로 표현

## 마 무 리

- 코드가 길고 여러 함수들을 연동해야하는 부분이 많아 복잡함  
이 많았음
- 여러 조건들(충돌조건, 블록 제거)등이 많아 잘못하면 코드가 꼬  
이는 경우가 생겨 많은 어려움을 겪었음
- 디자인부분에서 디그림과 기본 백그라운드컬러만을 이용해 만들  
어 부족하다는 생각이듬