Przedmiot: Technologie Internetowe

Dokumentacja Projektu Aplikacji W JavaScript "tetrisJS"

Wykonał: Konrad Szyszlak 131524

Rzeszów 2024

Nazwa pliku

index.html, style.css, game.js input.js main.js shapes.js utils.js

Typ dokumentu

HTML 5, Javascript

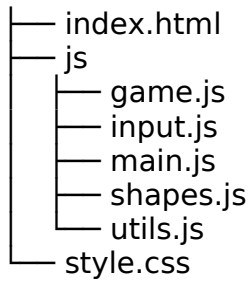
Kodowanie znaków

UTF-8

Tytuł strony

Tetris

Budowa strony



plik główny to index.html do niego podpięte są wszystkie skrypty i style.css, postanowiłem podzielić kod na “moduły” w celu ułatwienia znalezienia się w nim

Opis plików

index.html

- Opis: Główny plik HTML, który zawiera strukturę strony i wczytuje pliki JavaScript oraz CSS.
- Zawartość: Definiuje podstawowy układ strony, elementy HTML dla gry (np. canvas do rysowania gry), przyciski, elementy interfejsu, takie jak przycisk startowy, przycisk resetowania, obszar na najlepszy wynik itp.

js/
- Opis: Katalog zawierający wszystkie skrypty JavaScript.

game.js
- Opis: Główny plik logiczny gry.
- Zawartość:
  - Inicjalizacja planszy, zmienne globalne (board, score, gameOver itd.).
  - Funkcje do obsługi głównej pętli gry (startGame, startGameLoop), aktualizacji gry (update), renderowania planszy (render, drawBoard), logiki gry (mergeShape, removeFullLines), zarządzania punktacją (updateScore), sprawdzania końca gry (checkGameOver), zarządzania prędkością (updateSpeed), obsługi końca gry (endGame) oraz pauzowania (togglePause).
  - Renderowanie następnego klocka (renderNextShape).

input.js
- Opis: Obsługa wejścia użytkownika.
- Zawartość:
  - Funkcje do obsługi klawiszy sterujących grą (strzałki do przesuwania klocków, spacji do twardego opadania klocka).
  - Funkcja handleInput nasłuchująca na zdarzenia keydown i wywołująca odpowiednie akcje (moveLeft, moveRight, moveDown, rotateCurrentShape, hardDrop).

main.js
- Opis: Inicjalizacja gry oraz obsługa przycisków interfejsu.
- Zawartość:
  - Nasłuchiwanie na zdarzenia DOMContentLoaded w celu załadowania najlepszych wyników z localStorage.
  - Inicjalizacja elementów interfejsu użytkownika (startButton, resetButton), obsługa zdarzeń kliknięcia.
  - Wywoływanie startGame w celu rozpoczęcia gry.
  - Definicje nextShape i nextColor oraz renderowanie następnego klocka za pomocą drawNextPiece.

shapes.js
- Opis: Funkcje związane z kształtami klocków.
- Zawartość:
  - Funkcje rotateShape do obracania kształtu.
  - Funkcje drawShape do rysowania kształtu na planszy.
  - Funkcje checkCollision do sprawdzania kolizji klocka z planszą.

utils.js
- Opis: Funkcje pomocnicze.
- Zawartość:
  - Definicje kolorów (colors).
  - Funkcje getRandomShape do losowania kształtu.
  - Funkcje getRandomColor do losowania koloru.
  - Funkcje createBoard do tworzenia planszy gry.

style.css
- Opis: Plik CSS zawierający style dla strony.
- Zawartość: Style dla elementów HTML, takie jak plansza gry, przyciski, wyniki itp.

Index.html nie jest skomplikowany i jego jedyna zawartość to:

```
<!DOCTYPE html>
<html lang="pl">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Tetris</title>
<link rel="stylesheet" href="style.css">
<link rel="preconnect" href="https://fonts.googleapis.com">
<link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
<link href="https://fonts.googleapis.com/css2?family=VT323&display=swap"
rel="stylesheet">
</head>
<body>
<h1>Tetris</h1>
<button id="startButton">Start</button>
<canvas id="gameCanvas" width="300" height="600" ></canvas>
<div id="scoresContainer">
<div id="score">Score: 0</div>
<div id="bestScore" class="bestScorePause">Best: 0</div>
</div>


<div id="nextPieceContainer">
<span>Next Piece</span>
<canvas id="nextPieceCanvas" width="120" height="120"></canvas>
</div>
<div id="pause"><div>PAUZA</div><button
id="restart">Restart</button></div>

<script src="js/utils.js"></script>
<script src="js/shapes.js"></script>
<script src="js/input.js"></script>
<script src="js/game.js"></script>
<script src="js/main.js"></script>
</body>
</html>
```

podobnie style.css, również mało linii kodu, w końcu w tym projekcie chodziło o znajomość javaScript, poniżej jego zawartość:

```
*{
font-family: "VT323", serif;
font-weight: 400;
font-style: normal;
}

body {
font-family: Arial, sans-serif;
text-align: center;
margin: 0;
padding: 0;
background-color: #222;
color: white;
display: flex;
flex-direction: column;
justify-content: center;
align-items: center;
overflow: hidden;
}
h1{
font-size: 2rem;
}

canvas {
border: 2px solid #fff;
background-color: #111;
display: none;
}

#scoresContainer{
display: flex;
justify-content: center;
}


#score {
display: none;
padding: 10px;
font-size: 20px;
}
#bestScore{
padding: 10px;
font-size: 20px;
}

.bestScorePause {
position: absolute;
top: 30%;
z-index: 10;
```

```
margin: 0;
padding: 0;
font-weight: 600;
font-size: 50px !important;
color: white;
width: 100%;
text-align: center;
}

.scorePause {
position: absolute;
top: 20%;
z-index: 10;
margin: 0;
padding: 0;
font-weight: 600;
font-size: 50px !important;
color: white;
width: 100%;
text-align: center;
}


#startButton{
aspect-ratio: 2;
padding: 2%;
font-size: xx-large;
border-radius: 2rem;
}
#nextPieceContainer {
margin-top: 20px;
padding: 10px;
border: 2px solid #fff;
background-color: #111;
border-radius: 8px;
display: none;
text-align: center;
}

#nextPieceContainer canvas {
display: block;
margin: auto;
}

#nextPieceContainer span {
display: block;
font-size: 16px;
margin-bottom: 5px;
color: #ddd;
```

```
}

#pause{
font-size: 5rem;
margin: 0;
padding: 0;
position: absolute;
height: 100%;
width: 100%;
top: 0;
left: 0;
display: none;
flex-direction: column;
justify-content: center;
}
#restart{
width: 10rem;
font-size: xx-large;
position: absolute;
top: 60%;
left: 50%;
transform: translateX(-5rem);
aspect-ratio: 2;
border-radius: 2rem;
}

button{
cursor: pointer;
}
```

znajduje sie tu wyśrodkowanie canvasa, stylizacja buttonu startu i restaru oraz zmiana czcionki na taką bardziej retro, wziąłem ją z google fonts

main.js

Ten plik odpowiada za inicjalizację gry oraz obsługę interfejsu użytkownika. Poniżej znajduje się szczegółowy opis poszczególnych fragmentów kodu:

1. Nasłuchiwanie zdarzenia DOMContentLoaded

   ```
   document.addEventListener('DOMContentLoaded', () => {
   ```

   - Kod wewnątrz tej funkcji zostanie wykonany, gdy cała zawartość DOM zostanie załadowana i sparsowana. Dzięki temu mamy pewność, że wszystkie elementy HTML są dostępne do manipulacji.

2. Wczytanie najlepszego wyniku z localStorage

   ```
   const bestScore = localStorage.getItem('bestScore') || 0;
   document.getElementById('bestScore').innerText = `Best: ${bestScore}`;
   ```

   - Pobiera zapisany najlepszy wynik z localStorage. Jeśli nie ma zapisanego wyniku, ustawia bestScore na 0.
   - Aktualizuje tekst elementu HTML o identyfikatorze bestScore, aby wyświetlić najlepszy wynik.

3. Inicjalizacja przycisków start i reset

   ```
   const startButton = document.getElementById('startButton');
   const resetButton = document.getElementById("restart");
   ```

4. Obsługa zdarzenia kliknięcia przycisku reset

   ```
   resetButton.addEventListener("click", () => {
      location.reload();
   });
   ```

   - Dodaje nasłuchiwacza zdarzeń kliknięcia do przycisku resetowania. Po kliknięciu strony zostanie odświeżona, a gra rozpocznie się od nowa, było to najprostsze rozwiązanie, ponieważ jak widać po strukturze wszystko odbywa się w jednym pliku html, nie chciałem tworzyć “podstron”

5. Obsługa zdarzenia kliknięcia przycisku start

```
startButton.addEventListener('click', () => {
    document.getElementById('gameCanvas').style.display = 'block';
    document.getElementById('score').style.display = 'block';
    document.getElementById('nextPieceContainer').style.display = 'block';
    document.getElementById('bestScore').style.display = 'block';
    startButton.style.display = 'none';
    startGame();
});
```

- Dodaje nasłuchiwacza zdarzeń kliknięcia do przycisku startowego. Po kliknięciu:
    - Wyświetla elementy interfejsu użytkownika, takie jak plansza gry (gameCanvas), wynik (score), kontener następnego klocka (nextPieceContainer), najlepszy wynik (bestScore).
    - Ukrywa przycisk startowy.
    - Rozpoczyna grę, wywołując funkcję startGame().

6. Inicjalizacja następnego klocka

```
const nextCanvas = document.getElementById('nextPieceCanvas');
const nextCtx = nextCanvas.getContext('2d');
let nextShape = getRandomShape();
let nextColor = getRandomColor();
```

- Pobiera element nextPieceCanvas i uzyskuje jego kontekst 2D (nextCtx), który jest używany do rysowania na canvasie.
- Losuje kształt (nextShape) i kolor (nextColor) następnego klocka.

7. Funkcja drawNextPiece

```
function drawNextPiece() {
    nextCtx.clearRect(0, 0, nextCanvas.width, nextCanvas.height);
    nextShape.forEach((row, y) => {
        row.forEach((value, x) => {
            if (value) {
                nextCtx.fillStyle = nextColor;
                nextCtx.fillRect(x * 30, y * 30, 30, 30);
                nextCtx.strokeStyle = '#222';
                nextCtx.strokeRect(x * 30, y * 30, 30, 30);
            }
        });
    });
}
```

- Czyści obszar canvasu dla następnego klocka.
- Iteruje przez elementy nextShape i rysuje je na canvasie przy użyciu koloru nextColor.

- Używa funkcji fillRect do wypełniania prostokątów dla każdego elementu klocka.
- Używa funkcji strokeRect, aby narysować obramowanie każdego prostokąta, co wizualnie rozdziela elementy klocka.

W funkcji drawNextPiece, wartości x * 30 i y * 30 są używane do określenia pozycji rysowania klocków na canvasie. Wyjaśnię, jak to działa:

1. Rozmiar klocka:
    - Każdy element klocka (czyli każda komórka) ma szerokość i wysokość równą 30 pikseli. Dlatego, aby poprawnie rysować klocki na canvasie, musimy przeliczyć współrzędne elementów klocka na piksele.
    - Użycie wartości x * 30 i y * 30 oznacza, że każda komórka jest przeskalowana na canvasie o 30 pikseli, co ustawia ich odpowiednie pozycje w siatce canvasu.
2. Pętla iterująca przez kształt:
    - nextShape.forEach((row, y) => { row.forEach((value, x) => { ... }) }); - Ta pętla iteruje przez każdą komórkę kształtu. y to indeks wiersza, a x to indeks kolumny w danym wierszu.
    - value reprezentuje wartość komórki klocka, która może być 0 (brak klocka) lub 1 (obecność klocka).
3. Rysowanie komórek na canvasie:
    - nextCtx.fillRect(x * 30, y * 30, 30, 30); - Ta linia rysuje prostokąt (komórkę klocka) na canvasie. Wartości x * 30 i y * 30 ustalają pozycję lewego górnego rogu prostokąta:
        - x * 30 - określa pozycję w osi X (poziomej), przesuwając prostokąt o 30 pikseli za każdym razem, gdy x rośnie.
        - y * 30 - określa pozycję w osi Y (pionowej), przesuwając prostokąt o 30 pikseli za każdym razem, gdy y rośnie.
    - nextCtx.strokeRect(x * 30, y * 30, 30, 30); - Ta linia rysuje obramowanie prostokąta na canvasie, wykorzystując te same współrzędne.

Dzięki temu, używając współrzędnych x i y, możemy precyzyjnie rysować elementy klocka w odpowiednich pozycjach na siatce canvasu, zapewniając, że każdy element jest dokładnie tam, gdzie powinien być, i że wszystkie elementy mają odpowiedni rozmiar.

shapes.js

Ten plik zawiera funkcje związane z manipulowaniem kształtami klocków w grze. Poniżej znajduje się szczegółowy opis poszczególnych funkcji:

1. Funkcja rotateShape

```
function rotateShape(shape) {
    // Utwórz nową macierz, która będzie obrotem klocka
    const rotatedShape = shape[0].map((_, index) => shape.map(row =>
row[index]).reverse());
    return rotatedShape;
}
```

   - Opis: Funkcja ta obraca kształt klocka o 90 stopni w prawo.
   - Działanie:
      - shape[0].map((_, index) => shape.map(row => row[index]).reverse());
         - Pierwsza map iteruje przez kolumny oryginalnego kształtu (shape[0]).
         - Druga map tworzy nową macierz, przekształcając wiersze na kolumny (transpozycja macierzy), a następnie odwracając kolejność elementów, co powoduje obrót o 90 stopni. W życiu bym nie pomyślał że to tak działa ale szukając rozwiązań tego problemu podczas pracy nad projektem trafiłem właśnie na to
      - return rotatedShape; zwraca obrócony kształt.

2. Funkcja drawShape

```
function drawShape(ctx, shape, position, color) {
    shape.forEach((row, y) => {
        row.forEach((value, x) => {
            if (value) {
                ctx.fillStyle = color;
                ctx.fillRect((position.x + x) * 30, (position.y + y) * 30, 30, 30);
                ctx.strokeStyle = '#222';
                ctx.strokeRect((position.x + x) * 30, (position.y + y) * 30, 30, 30);
            }
        });
    });
}
```

   - Opis: Funkcja ta rysuje kształt klocka na planszy gry.
   - Parametry:
      - ctx: Kontekst renderowania canvasu.
      - shape: Macierz reprezentująca kształt klocka.

        - position: Obiekt z właściwościami x i y, określający pozycję klocka na planszy.
        - color: Kolor, którym będzie wypełniony kształt klocka.
    - Działanie:
        - shape.forEach((row, y) => { row.forEach((value, x) => { ... }) }); - Pętla iterująca przez wiersze (row) i kolumny (value) kształtu klocka.
        - if (value) { ... } - Sprawdza, czy komórka klocka ma wartość (czyli jest częścią klocka).
        - ctx.fillStyle = color; - Ustawia kolor wypełnienia na przekazany parametr color.
        - ctx.fillRect((position.x + x) * 30, (position.y + y) * 30, 30, 30); - Rysuje prostokąt reprezentujący komórkę klocka na planszy, skalując współrzędne x i y o 30 pikseli.
        - ctx.strokeRect((position.x + x) * 30, (position.y + y) * 30, 30, 30); - Rysuje obramowanie prostokąta.

3. Funkcja checkCollision

```
function checkCollision(shape, position, board) {
    for (let y = 0; y < shape.length; y++) {
        for (let x = 0; x < shape[y].length; x++) {
            if (shape[y][x] !== 0) {
                const boardY = position.y + y;
                const boardX = position.x + x;
                if (boardX < 0 || boardX >= board[0].length || boardY >= board.length
|| boardY < 0 || board[boardY][boardX].value !== 0) {
                    return true;
                }
            }
        }
    }
    return false;
}
```

    - Opis: Funkcja ta sprawdza, czy kształt klocka koliduje z innymi klockami na planszy lub z granicami planszy.
    - Parametry:
        - shape: Macierz reprezentująca kształt klocka.
        - position: Obiekt z właściwościami x i y, określający pozycję klocka na planszy.
        - board: Macierz reprezentująca planszę gry.
    - Działanie:

- for (let y = 0; y < shape.length; y++) { ... } - Pętla iterująca przez wiersze kształtu.
- for (let x = 0; x < shape[y].length; x++) { ... } - Pętla iterująca przez kolumny kształtu.
- if (shape[y][x] !== 0) { ... } - Sprawdza, czy komórka klocka ma wartość (czyli jest częścią klocka).
- const boardY = position.y + y; i const boardX = position.x + x; - Przelicza współrzędne komórki klocka na współrzędne planszy.
- if (boardX < 0 || boardX >= board[0].length || boardY >= board.length || boardY < 0 || board[boardY][boardX].value !== 0) { return true; } - Sprawdza, czy komórka znajduje się poza granicami planszy lub koliduje z innym klockiem na planszy. Jeśli tak, zwraca true, wskazując na kolizję.
- return false; - Jeśli żadna kolizja nie zostanie wykryta, funkcja zwraca false.

input.js

Najprostszy z plików, odpowiada za łapanie przycisków wciśniętych przez gracza Poniżej znajduje się szczegółowy opis poszczególnych funkcji:

1. Funkcja handleInput

```
function handleInput() {
    document.addEventListener('keydown', (event) => {
        if (gameOver) return;
        if (event.key === 'p' || event.key === 'P') {
            togglePause();
        }
    });

    document.addEventListener('keydown', (event) => {
        if (isPaused) return;
        switch (event.key) {
            case 'ArrowLeft':
                moveLeft();
                render(); // Renderuj po ruchu
                break;
            case 'ArrowRight':
                moveRight();
                render(); // Renderuj po ruchu
                break;
            case 'ArrowDown':
                moveDown();
                render(); // Renderuj po ruchu
                break;
            case 'ArrowUp':
                rotateCurrentShape();
                render(); // Renderuj po obrocie
                break;
            case ' ':
                hardDrop(); // Twardy spadek
                render();
                break;
        }
    });
}
```

   - Opis: Funkcja ta nasłuchuje na zdarzenia klawiatury i wywołuje odpowiednie akcje w grze.
   - Działanie:
      - document.addEventListener('keydown', (event) => { ... }); - Dodaje nasłuchiwacza zdarzeń keydown.
      - Pierwszy nasłuchiwacz:
         - Sprawdza, czy gra jest zakończona (if (gameOver) return;).

- Sprawdza, czy naciśnięto klawisz 'p' lub 'P', aby przełączyć pauzę (togglePause();).

- Drugi nasłuchiwacz:
  - Sprawdza, czy gra jest w stanie pauzy (if (isPaused) return;).
  - Używa switch do wywoływania odpowiednich funkcji w zależności od naciśniętego klawisza:
    - ArrowLeft: Wywołuje moveLeft() i render().
    - ArrowRight: Wywołuje moveRight() i render().
    - ArrowDown: Wywołuje moveDown() i render().
    - ArrowUp: Wywołuje rotateCurrentShape() i render().
    - Spacja (' '): Wywołuje hardDrop() i render().

Wywołanie render po zmianie ruchu sprawia że klocek porusza sie płynnie I daje możliwość “ślizgu” gdy już dotyka I koliduje z czymś, a jednak można go jeszcze przesunąć póki nie wykona sie główna pętla gry

2. Funkcja moveLeft

```
function moveLeft() {
    shapePosition.x -= 1;
    if (checkCollision(currentShape, shapePosition, board)) {
        shapePosition.x += 1; // Cofnij ruch w razie kolizji
    }
}
```

   - Opis: Przesuwa kształt w lewo na planszy.
   - Działanie:
     - Zmniejsza współrzędną x pozycji kształtu (shapePosition.x -= 1;).
     - Sprawdza, czy kształt koliduje z planszą (checkCollision(currentShape, shapePosition, board)).
     - Jeśli wystąpi kolizja, cofa ruch (shapePosition.x += 1;).

3. Funkcja moveRight

```
function moveRight() {
    shapePosition.x += 1;
    if (checkCollision(currentShape, shapePosition, board)) {
        shapePosition.x -= 1; // Cofnij ruch w razie kolizji
    }
}
```

- Opis: Przesuwa kształt w prawo na planszy.
- Działanie:
    - Zwiększa współrzędną x pozycji kształtu (shapePosition.x += 1;).
    - Sprawdza, czy kształt koliduje z planszą (checkCollision(currentShape, shapePosition, board)).
    - Jeśli wystąpi kolizja, cofa ruch (shapePosition.x -= 1;).

4. Funkcja moveDown

```
function moveDown() {
   shapePosition.y += 1;
   if (checkCollision(currentShape, shapePosition, board)) {
      shapePosition.y -= 1; // Cofnij ruch w razie kolizji
      mergeShape(currentShape, shapePosition, board, currentColor); // Osadź
kształt
      currentShape = nextShape; // Przypisz nowy kształt
      currentColor = nextColor; // Przypisz nowy kolor
      nextShape = getRandomShape(); // Wylosuj następny kształt
      nextColor = getRandomColor(); // Wylosuj następny kolor
      shapePosition = { x: Math.floor(cols / 2) - 1, y: 0 };

      renderNextShape(); // Odśwież podgląd na kolejne klocki
      if (checkGameOver(board)) {
         endGame();
      }
   }
}
```

- Opis: Przesuwa kształt w dół na planszy.
- Działanie:
    - Zwiększa współrzędną y pozycji kształtu (shapePosition.y += 1;).
    - Sprawdza, czy kształt koliduje z planszą (checkCollision(currentShape, shapePosition, board)).
    - Jeśli wystąpi kolizja:
        - Cofnij ruch (shapePosition.y -= 1;).
        - Osadza kształt na planszy (mergeShape(currentShape, shapePosition, board, currentColor);).
        - Przypisuje nowe kształty i kolory (currentShape = nextShape; currentColor = nextColor;).
        - Losuje nowe kształty i kolory (nextShape = getRandomShape(); nextColor = getRandomColor();).

- Ustawia pozycję kształtu na środek u góry planszy (shapePosition = { x: Math.floor(cols / 2) - 1, y: 0 };).
- Odświeża podgląd na następne klocki (renderNextShape();).
- Sprawdza, czy gra się kończy (checkGameOver(board)), a jeśli tak, kończy grę (endGame();).

5. Funkcja rotateCurrentShape

```
function rotateCurrentShape() {
    const rotatedShape = rotateShape(currentShape);
    if (!checkCollision(rotatedShape, shapePosition, board)) {
        currentShape = rotatedShape; // Jeśli nie ma kolizji, obróć kształt
    }
}
```

- Opis: Obraca kształt klocka.
- Działanie:
    - Obraca kształt, wywołując funkcję rotateShape(currentShape) i przypisuje wynik do rotatedShape.
    - Sprawdza, czy obrócony kształt koliduje z planszą (checkCollision(rotatedShape, shapePosition, board)).
    - Jeśli nie ma kolizji, przypisuje obrócony kształt jako aktualny (currentShape = rotatedShape;).

6. Funkcja hardDrop

```
function hardDrop() {
    while (!checkCollision(currentShape, { x: shapePosition.x, y: shapePosition.y +
1 }, board)) {
        shapePosition.y += 1;
    }
    mergeShape(currentShape, shapePosition, board, currentColor);
    currentShape = getRandomShape();
    currentColor = getRandomColor();
    shapePosition = { x: Math.floor(cols / 2) - 1, y: 0 };
    currentShape = nextShape; // Przypisz nowy kształt
    currentColor = nextColor; // Przypisz nowy kolor
    nextShape = getRandomShape(); // Wylosuj następny kształt
    nextColor = getRandomColor(); // Wylosuj następny kolor
    shapePosition = { x: Math.floor(cols / 2) - 1, y: 0 };

    renderNextShape(); // Odśwież podgląd na kolejne klocki
    if (checkGameOver(board)) {
        endGame();
    }
}
```

- Opis: Funkcja ta sprawia, że aktualny kształt natychmiastowo spada na najniższe możliwe miejsce na planszy.
- Działanie:
    - while (!checkCollision(currentShape, { x: shapePosition.x, y: shapePosition.y + 1 }, board)) { ... } - Pętla while przesuwa kształt w dół, dopóki nie zostanie wykryta kolizja z planszą.
    - shapePosition.y += 1; - Zwiększa współrzędną y pozycji kształtu, przesuwając go w dół.
    - mergeShape(currentShape, shapePosition, board, currentColor); - Osadza kształt na planszy, gdy osiągnie najniższe możliwe miejsce.
    - currentShape = nextShape; currentColor = nextColor; - Przypisuje nowe kształty i kolory jako aktualne.
    - nextShape = getRandomShape(); nextColor = getRandomColor(); - Losuje nowe kształty i kolory dla następnego klocka.
    - shapePosition = { x: Math.floor(cols / 2) - 1, y: 0 }; - Ustawia pozycję nowego kształtu na środek u góry planszy.
    - renderNextShape(); - Odświeża podgląd na następne klocki.
    - if (checkGameOver(board)) { endGame(); } - Sprawdza, czy gra się kończy, a jeśli tak, kończy grę.

utils.js

Ten plik zawiera funkcje pomocnicze, które są używane do generowania losowych kształtów i kolorów klocków w grze. Poniżej znajduje się szczegółowy opis poszczególnych fragmentów kodu:

1. Tablica colors

```
const colors = ['#FF5733', '#33FF57', '#3357FF', '#FFFF33', '#FF33FF', '#33FFFF',
'#FF9933'];
```

   - Opis: Tablica colors zawiera zestaw kolorów, które mogą być przypisane do klocków.
   - Zawartość: Kolory zapisane jako wartości szesnastkowe (hex), które reprezentują różne barwy.

2. Funkcja getRandomShape

```
function getRandomShape() {
    const shapes = [
        [[1, 1, 1, 1]], // I

        [[1, 1],
         [1, 1]], // O

        [[0, 1, 0],
         [1, 1, 1]], // T

        [[1, 1, 0],
         [0, 1, 1]], // Z

        [[0, 1, 1],
         [1, 1, 0]], // S

        [[1, 0, 0],
         [1, 1, 1]], // L

        [[0, 0, 1],
         [1, 1, 1]], // J
    ];
    const randomIndex = Math.floor(Math.random() * shapes.length);
    return shapes[randomIndex];
}
```

   - Opis: Funkcja ta losuje jeden z predefiniowanych kształtów klocków.
   - Kształty:
      - [[1, 1, 1, 1]] - Kształt "I" (prosta linia).
      - [[1, 1], [1, 1]] - Kształt "O" (kwadrat).
      - [[0, 1, 0], [1, 1, 1]] - Kształt "T".

        - [[1, 1, 0], [0, 1, 1]] - Kształt "Z".
        - [[0, 1, 1], [1, 1, 0]] - Kształt "S".
        - [[1, 0, 0], [1, 1, 1]] - Kształt "L".
        - [[0, 0, 1], [1, 1, 1]] - Kształt "J".
    - Działanie:
        - Math.floor(Math.random() * shapes.length); - Losuje indeks z zakresu długości tablicy shapes.
        - return shapes[randomIndex]; - Zwraca losowo wybrany kształt klocka.

3. Funkcja getRandomColor

```
function getRandomColor() {
    return colors[Math.floor(Math.random() * colors.length)];
}
```

    - Opis: Funkcja ta losuje jeden z predefiniowanych kolorów.
    - Działanie:
        - Math.floor(Math.random() * colors.length); - Losuje indeks z zakresu długości tablicy colors.
        - return colors[randomIndex]; - Zwraca losowo wybrany kolor z tablicy colors.

game.js

W tym pliku dzieje się najwięcej, ten plik zawiera główną logikę gry, w tym renderowanie, aktualizację stanu gry oraz funkcje pomocnicze. Poniżej znajduje się szczegółowy opis poszczególnych sekcji kodu:

Plik ten podzieliłem komentarzami na “sekcje” aby jeszcze łatwiej można było się połapać co się w nim dzieje

Sekcja: Inicjalizacja planszy

```
const canvas = document.getElementById('gameCanvas');
const ctx = canvas.getContext('2d');

const rows = 20;
const cols = 10;
const board = Array.from({ length: rows }, () => Array.from({ length: cols }, () =>
({ value: 0, color: '#111' })));
let score = 0;
let gameOver = false;
let speed = 500; // Początkowa prędkość opadania klocków
let isPaused = false;
let intervalId;

let currentShape = getRandomShape();
let currentColor = getRandomColor();
let shapePosition = { x: Math.floor(cols / 2) - 1, y: 0 }; // Początkowa pozycja
```

- Opis: Inicjalizuje planszę gry oraz zmienne globalne.
- Działanie:
    - const canvas = document.getElementById('gameCanvas'); - Pobiera element canvas z DOM.
    - const ctx = canvas.getContext('2d'); - Pobiera kontekst 2D do rysowania na canvasie.
    - const rows = 20; i const cols = 10; - Definiuje liczbę wierszy i kolumn planszy.
    - const board = Array.from({ length: rows }, () => Array.from({ length: cols }, () => ({ value: 0, color: '#111' }))); - Tworzy dwuwymiarową tablicę reprezentującą planszę gry, gdzie każda komórka ma początkową wartość 0 i kolor '#111'.
    - let score = 0; - Inicjalizuje zmienną score (wynik).
    - let gameOver = false; - Flaga wskazująca, czy gra się zakończyła.
    - let speed = 500; - Początkowa prędkość opadania klocków (w milisekundach).
    - let isPaused = false; - Flaga wskazująca, czy gra jest w stanie pauzy.

- let intervalId; - Zmienna do przechowywania identyfikatora interwału głównej pętli gry.
- let currentShape = getRandomShape(); - Losowy początkowy kształt klocka.
- let currentColor = getRandomColor(); - Losowy początkowy kolor klocka.
- let shapePosition = { x: Math.floor(cols / 2) - 1, y: 0 }; - Początkowa pozycja klocka na planszy (środek u góry).

Sekcja: Inicjalizacja gry

```
function startGame() {
    console.log('Gra rozpoczęta');
    renderNextShape(); // Pokaż początkowy podgląd
    document.getElementById('bestScore').classList.remove("bestScorePause");
    handleInput();
    startGameLoop();
}

function startGameLoop() {
    if (intervalId) {
        clearInterval(intervalId);
    }
    console.log(`Aktualna prędkość: ${speed} ms`);
    intervalId = setInterval(() => {
        if (!gameOver) {
            update();
            render();
        }
    }, speed);
}
```

- Opis: Zawiera funkcje do inicjalizacji gry oraz uruchomienia głównej pętli gry.
- Działanie:
    - function startGame() { ... } - Rozpoczyna grę:
        - Wywołuje renderNextShape() w celu pokazania podglądu następnego klocka.
        - Usuwa klasę CSS bestScorePause z elementu bestScore.
        - Wywołuje handleInput() do obsługi wejścia użytkownika.
        - Uruchamia główną pętlę gry, wywołując startGameLoop().
    - function startGameLoop() { ... } - Uruchamia główną pętlę gry:
        - Jeśli interwał jest już uruchomiony, zatrzymuje go (clearInterval(intervalId);).

- Ustawia nowy interwał pętli gry (intervalId = setInterval(...);), który wywołuje update() i render() co speed milisekund, dopóki gra nie jest zakończona.

Sekcja: Aktualizacja stanu gry

```
function update() {
   shapePosition.y += 1;
   if (checkCollision(currentShape, shapePosition, board)) {
      shapePosition.y -= 1; // Cofnij ruch w razie kolizji
      mergeShape(currentShape, shapePosition, board, currentColor);
      if (checkGameOver(board)) {
         endGame();
      } else {
         currentShape = nextShape; // Ustaw obecny kształt jako wcześniej wylosowany
         currentColor = nextColor; // Ustaw obecny kolor
         nextShape = getRandomShape(); // Wylosuj nowy kształt
         nextColor = getRandomColor(); // Wylosuj nowy kolor
         shapePosition = { x: Math.floor(cols / 2) - 1, y: 0 };
         renderNextShape(); // Odśwież podgląd
      }
   }
}
```

- Opis: Aktualizuje stan gry w głównej pętli.
- Działanie:
    - shapePosition.y += 1; - Przesuwa kształt w dół o jedną jednostkę.
    - if (checkCollision(currentShape, shapePosition, board)) { ... } - Sprawdza, czy kształt koliduje z planszą.
        - Jeśli wystąpi kolizja:
            - Cofnij ruch (shapePosition.y -= 1;).
            - Osadza kształt na planszy (mergeShape(currentShape, shapePosition, board, currentColor);).
            - Sprawdza, czy gra się kończy (checkGameOver(board)), a jeśli tak, kończy grę (endGame();).
            - Jeśli gra się nie kończy, ustawia nowy kształt i kolor jako obecne, losuje nowe kształty i kolory dla następnego klocka oraz resetuje pozycję kształtu.
            - Odświeża podgląd na następne klocki (renderNextShape();).

Sekcja: Renderowanie i odświeżanie

```
function render() {
   ctx.clearRect(0, 0, canvas.width, canvas.height);
   drawBoard();
   drawShape(ctx, currentShape, shapePosition, currentColor);
}

function drawBoard() {
   for (let row = 0; row < rows; row++) {
      for (let col = 0; col < cols; col++) {
         ctx.fillStyle = board[row][col].color;
         ctx.fillRect(col * 30, row * 30, 30, 30);
         ctx.strokeStyle = '#222';
         ctx.strokeRect(col * 30, row * 30, 30, 30);
      }
   }
}

function renderNextShape() {
   const nextCanvas = document.getElementById('nextPieceCanvas');
   const nextCtx = nextCanvas.getContext('2d');
   nextCtx.clearRect(0, 0, nextCanvas.width, nextCanvas.height);

   const cellSize = 30;
   const offsetX = (nextCanvas.width - nextShape[0].length * cellSize) / 2;
   const offsetY = (nextCanvas.height - nextShape.length * cellSize) / 2;

   nextShape.forEach((row, y) => {
      row.forEach((value, x) => {
         if (value) {
            nextCtx.fillStyle = nextColor;
            nextCtx.fillRect(offsetX + x * cellSize, offsetY + y * cellSize, cellSize, cellSize);
            nextCtx.strokeStyle = '#222';
            nextCtx.strokeRect(offsetX + x * cellSize, offsetY + y * cellSize, cellSize,
cellSize);
         }
      });
   });
}
```

- Opis: Zawiera funkcje odpowiedzialne za renderowanie i odświeżanie planszy oraz klocków.
    - Działanie:
        - function drawBoard() { ... } - Rysuje planszę gry:
        - Iteruje przez wiersze (row) i kolumny (col) planszy.
        - Dla każdej komórki planszy ustawia kolor (ctx.fillStyle = board[row][col].color;) i rysuje prostokąt (ctx.fillRect(col * 30, row * 30, 30, 30);).

        - Rysuje obramowanie prostokąta (ctx.strokeRect(col * 30, row * 30, 30, 30);).
    - function renderNextShape() { ... } - Rysuje podgląd następnego klocka:
        - Pobiera element canvas dla podglądu (nextCanvas) i kontekst 2D (nextCtx).
        - Czyści canvas (nextCtx.clearRect(0, 0, nextCanvas.width, nextCanvas.height);).
        - Oblicza przesunięcie (offsetX, offsetY), aby wyśrodkować kształt na canvasie.
        - Iteruje przez wiersze (row) i kolumny (value) kształtu nextShape.
        - Dla każdej komórki kształtu ustawia kolor wypełnienia (nextCtx.fillStyle = nextColor;) i rysuje prostokąt (nextCtx.fillRect(offsetX + x * cellSize, offsetY + y * cellSize, cellSize, cellSize);).
        - Rysuje obramowanie prostokąta (nextCtx.strokeRect(offsetX + x * cellSize, offsetY + y * cellSize, cellSize, cellSize);).

Sekcja: Funkcje pomocnicze

```
function mergeShape(shape, position, board, color) {
    shape.forEach((row, y) => {
        row.forEach((value, x) => {
            if (value) {
                const boardY = position.y + y;
                const boardX = position.x + x;
                if (boardY >= 0 && boardY < board.length && boardX >= 0 && boardX <
board[0].length) {
                    board[boardY][boardX] = { value, color }; // Przypisz wartość i kolor
                }
            }
        });
    });

    const linesRemoved = removeFullLines();
    if (linesRemoved > 0) {
        updateScore(linesRemoved);
    }
}

function removeFullLines() {
    let linesToRemove = [];
    for (let row = 0; row < rows; row++) {
```

```
        if (board[row].every(cell => cell.value !== 0)) {
            linesToRemove.push(row);
        }
    }

    if (linesToRemove.length > 0) {
        linesToRemove.forEach(row => {
            board[row].forEach(cell => (cell.color = 'white')); // Migotanie
        });

        setTimeout(() => {
            linesToRemove.forEach(row => {
                board.splice(row, 1);
                board.unshift(Array.from({ length: cols }, () => ({ value: 0, color: '#111' })));
            });
            updateScore(linesToRemove.length);
            render();
        }, 200); // Opóźnienie przed usunięciem linii
    }
}

function updateScore(linesRemoved) {

let previousScore = score;
score += linesRemoved * 100;
document.getElementById('score').innerText = `Score: ${score}`;

const bestScore = localStorage.getItem('bestScore') || 0;
if (score > bestScore) {
localStorage.setItem('bestScore', score);
document.getElementById('bestScore').innerText = `Best: ${score}`;
}

// Przyspieszanie gry co 300 punktów
let previousLevel = Math.floor(previousScore / 300);
let currentLevel = Math.floor(score / 300);

if (currentLevel > previousLevel) {
updateSpeed();
}
}



function updateSpeed() {
    speed = Math.max(100, speed - 50); // Zwiększ prędkość, minimalna wartość to 100
ms
    startGameLoop(); // Zaktualizuj interwał
}

function checkGameOver(board) {
```

```
    return board[0].some(cell => cell.value !== 0);
}

function endGame() {
    gameOver = true;
    togglePause();
    document.querySelector("#pause div").textContent = "You Lose";
    clearInterval(intervalId); // Zatrzymaj pętlę gry
}

function togglePause() {
    isPaused = !isPaused;
    if (isPaused) {
        document.querySelector("#pause").style.backgroundColor = "#222";
        clearInterval(intervalId);
        document.getElementById('pause').style.display = "flex";
        document.getElementById('bestScore').classList.add("bestScorePause");
        document.getElementById('score').classList.add("scorePause");
    } else {
        document.querySelector("#pause").style.backgroundColor = "transparent";
        startGameLoop();
        document.getElementById('pause').style.display = "none";
        document.getElementById('bestScore').classList.remove("bestScorePause");
        document.getElementById('score').classList.remove("scorePause");
    }
}

window.startGame = startGame;
window.rotateShape = rotateShape;
```

- Opis: Zawiera funkcje pomocnicze używane do obsługi gry, takie jak łączenie kształtów, usuwanie pełnych linii, aktualizacja wyniku, sprawdzanie końca gry, obsługa pauzy i zwiększanie prędkości gry.
- Działanie:
- Funkcja mergeShape:
    - Opis: Osadza kształt na planszy i sprawdza, czy zostały usunięte pełne linie.
    - Działanie:
        - Iteruje przez komórki kształtu i przypisuje je do odpowiednich pozycji na planszy.
        - Sprawdza i usuwa pełne linie za pomocą removeFullLines.
        - Aktualizuje wynik, jeśli zostały usunięte pełne linie.
- Funkcja removeFullLines:
    - Opis: Sprawdza planszę i usuwa pełne linie.

    - Działanie:
        - Iteruje przez wiersze planszy, sprawdza, czy każdy wiersz jest pełny (czy każda komórka ma wartość inną niż 0).
        - Usuwa pełne wiersze i wstawia nowe wiersze u góry planszy.
        - Aktualizuje wynik i renderuje planszę po usunięciu linii.
- Funkcja updateScore:
    - Opis: Aktualizuje wynik i sprawdza, czy należy zwiększyć prędkość gry.
    - Działanie:
        - Dodaje punkty za usunięte linie.
        - Sprawdza i aktualizuje najlepszy wynik w localStorage.
        - Co 300 punktów przyspiesza grę, wywołując updateSpeed.
- Funkcja updateSpeed:
    - Opis: Zwiększa prędkość gry, skracając czas interwału pętli gry.
    - Działanie:
        - Zmniejsza wartość speed, minimalnie do 100 ms.
        - Aktualizuje interwał pętli gry, wywołując startGameLoop.
- Funkcja checkGameOver:
    - Opis: Sprawdza, czy gra się zakończyła.
    - Działanie:
        - Sprawdza, czy w górnym wierszu planszy są jakiekolwiek wartości różne od 0, co oznacza koniec gry.
- Funkcja endGame:
    - Opis: Kończy grę i zatrzymuje pętlę gry.
    - Działanie:
        - Ustawia flagę gameOver na true.
        - Przełącza grę na pauzę.
        - Wyświetla komunikat "You Lose".
        - Zatrzymuje pętlę gry, zatrzymując interwał.
- Funkcja togglePause:
    - Opis: Przełącza stan pauzy gry.
    - Działanie:
        - Ustawia flagę isPaused na przeciwną wartość.

        - Jeśli gra jest w stanie pauzy:
            - Zatrzymuje pętlę gry.
            - Wyświetla element pause.
            - Dodaje klasy CSS bestScorePause i scorePause.
        - Jeśli gra jest kontynuowana:
            - Usuwa klasy CSS bestScorePause i scorePause.
            - Uruchamia pętlę gry.

- Rejestracja funkcji globalnych:
    - window.startGame = startGame;
    - window.rotateShape = rotateShape;

Plansza Gry jako Tablica 2D

Plansza gry jest reprezentowana jako tablica 2D, co oznacza, że jest to tablica tablic. Każda komórka tej tablicy odpowiada jednej komórce na planszy gry. W przypadku klasycznej gry Tetris plansza ma zwykle 20 wierszy i 10 kolumn.

Reprezentacja Tablicy 2D

- **Wiersze:** Poziome rzędy na planszy (20 wierszy).
- **Kolumny:** Pionowe kolumny na planszy (10 kolumn).

Każda komórka w tablicy 2D może przechowywać informacje o klocku:

- **value:** Czy komórka jest częścią klocka (1) czy jest pusta (0).
- **color:** Kolor klocka (w przypadku pustej komórki może to być np. kolor tła).

Struktura Tablicy 2D

```
const rows = 20;
const cols = 10;
const board = Array.from({ length: rows }, () =>
    Array.from({ length: cols }, () => ({ value: 0, color: '#111' }))
);
```

- **Array.from({ length: rows })**: Tworzy tablicę o długości `rows` (20).
- **Array.from({ length: cols })**: Tworzy wewnętrzną tablicę o długości `cols` (10) dla każdej komórki zewnętrznej tablicy.
- **({ value: 0, color: '#111' })**: Każda komórka jest inicjalizowana jako obiekt z wartością `0` (pusta) i kolorem `#111`.

Przykładowa Reprezentacja w Tablicy 2D

Poniżej znajduje się przykładowa reprezentacja planszy 2D z wierszami i kolumnami:

```
[
    [ { value: 0, color: '#111' }, { value: 0, color: '#111' }, ... ],  // Wiersz 0
    [ { value: 0, color: '#111' }, { value: 1, color: '#FF5733' }, ... ],  // Wiersz 1
    [ { value: 0, color: '#111' }, { value: 1, color: '#FF5733' }, ... ],  // Wiersz 2
    ...
    [ { value: 0, color: '#111' }, { value: 0, color: '#111' }, ... ],  // Wiersz 19
]
```

Rysowanie na Canvasie

- **Komórka:** Każda komórka na planszy jest rysowana jako prostokąt o wymiarach 30x30 pikseli.
- **Pozycja:** Pozycja komórki na canvasie jest określona przez jej indeks wiersza (`row`) i kolumny (`col`).

Algorytm Rysowania Planszy

1. Iterujemy przez każdą komórkę w tablicy 2D.
2. Dla każdej komórki rysujemy prostokąt na canvasie.
    - Współrzędne X: `col * 30` (kolumna mnożona przez szerokość komórki).
    - Współrzędne Y: `row * 30` (wiersz mnożony przez wysokość komórki).
3. Ustawiamy kolor wypełnienia na wartość `color` komórki.
4. Rysujemy prostokąt w określonym miejscu.

Przykład Kodowania

javascript

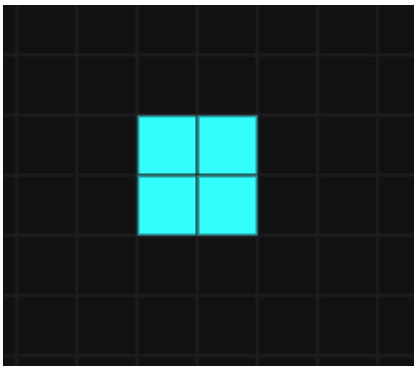
```
function drawBoard() {
    for (let row = 0; row < rows; row++) {
        for (let col = 0; col < cols; col++) {
            ctx.fillStyle = board[row][col].color;
            ctx.fillRect(col * 30, row * 30, 30, 30);
            ctx.strokeStyle = '#222';
            ctx.strokeRect(col * 30, row * 30, 30, 30);
        }
    }
}
```

Wizualizacja

Wyobraźmy sobie, że plansza ma 4 wiersze i 4 kolumny (dla uproszczenia):

```
Tablica 2D:
[
    [ {0, '#111'}, {0, '#111'}, {0, '#111'}, {0, '#111'} ],
    [ {0, '#111'}, {1, '#FF5733'}, {1, '#FF5733'}, {0, '#111'} ],
    [ {0, '#111'}, {1, '#FF5733'}, {1, '#FF5733'}, {0, '#111'} ],
    [ {0, '#111'}, {0, '#111'}, {0, '#111'}, {0, '#111'} ]
]
```

`Canvas:`



- **Pozycja:** Każda komórka w tablicy 2D odpowiada jednemu prostokątowi na canvasie.
- **Wartość:** `value` (0 lub 1) określa, czy komórka jest częścią klocka.
- **Kolor:** `color` określa kolor wypełnienia prostokąta.