

# **Sprawozdanie z Projektu Implementacja Gry Webowej "Tile Storm"**

Krzysztof Róg

czerwiec 2025

# Spis treści

<b>1</b>	<b>Cel i Zakres Projektu</b>	<b>4</b>
1.1	Opis zadania . . . . .	4
1.2	Założenia projektowe . . . . .	4
1.3	Wymagania funkcjonalne . . . . .	4
<b>2</b>	<b>Przebieg działania gry i wygląd interfejsu</b>	<b>4</b>
2.1	Ekran główny (menu startowe) . . . . .	5
2.2	Instrukcje dla gracza . . . . .	5
2.3	Panel ustawień dźwięku . . . . .	6
2.4	Tabela najlepszych wyników . . . . .	7
2.5	Przebieg rozgrywki . . . . .	8
2.6	Ekran zakończenia gry . . . . .	9
<b>3</b>	<b>Wykorzystane Technologie</b>	<b>9</b>
3.1	HTML5 . . . . .	9
3.2	CSS3 . . . . .	9
3.3	JavaScript ES6+ . . . . .	10
3.4	Web Audio API . . . . .	10
<b>4</b>	<b>Architektura Aplikacji</b>	<b>10</b>
4.1	Struktura projektu . . . . .	10
4.2	Organizacja kodu CSS . . . . .	10
4.3	Wzorce projektowe w JavaScript . . . . .	11
4.3.1	Singleton Pattern . . . . .	11
4.3.2	Object Literal Pattern . . . . .	11
<b>5</b>	<b>Implementacja Mechanik Gry</b>	<b>11</b>
5.1	System kafelków . . . . .	11
5.2	Algorytm spawnu kafelków . . . . .	12
5.3	System kolizji i interakcji . . . . .	12
<b>6</b>	<b>System Audio</b>	<b>13</b>
6.1	Klasa AudioGenerator . . . . .	13
6.2	Generowanie dźwięków pianina . . . . .	13
6.3	Muzyka tła . . . . .	14
<b>7</b>	<b>Interfejs Użytkownika</b>	<b>14</b>
7.1	Responsywny design . . . . .	14
7.2	Animacje CSS . . . . .	15
7.3	System modalnych okien . . . . .	15
<b>8</b>	<b>Zarządzanie Stanem</b>	<b>16</b>
8.1	Cykl życia gry . . . . .	16
8.2	Główna petla gry . . . . .	16
<b>9</b>	<b>Trwałość Danych</b>	<b>17</b>
9.1	System zapisywania wyników . . . . .	17
9.2	Zarządzanie ustawieniami . . . . .	17

<b>10 Efekty Wizualne</b>	<b>18</b>
10.1 Animacje kafelków . . . . .	18
10.2 Efekty czasteczkowe . . . . .	18
<b>11 Obsługa Błędów</b>	<b>19</b>
11.1 Walidacja danych wejściowych . . . . .	19
11.2 Obsługa błędów Audio API . . . . .	19
<b>12 Testowanie i Debugowanie</b>	<b>19</b>
12.1 Metody testowania . . . . .	19
12.2 Znalezione problemy i rozwiązania . . . . .	20
<b>13 Wnioski i Obserwacje</b>	<b>20</b>
13.1 Napotkane wyzwania . . . . .	20
13.2 Nabyte umiejętności . . . . .	20
13.3 Możliwości rozwoju . . . . .	21
13.4 Refleksje techniczne . . . . .	21
13.5 Narzędzia wykorzystane . . . . .	21

# 1 Cel i Zakres Projektu

## 1.1 Opis zadania

W ramach projektu zostało zrealizowane zadanie stworzenia interaktywnej gry webowej "Tile Storm" wykorzystującej wyłącznie technologie front-endowe. Gra należy do gatunku rhythm game, gdzie gracz musi reagować na spadające elementy w odpowiednim momencie.

## 1.2 Założenia projektowe

Główne założenia projektu obejmowały:

- Implementacje w czystym JavaScript bez zewnętrznych bibliotek
- Wykorzystanie Web Audio API do generowania dźwięków
- Stworzenie responsywnego interfejsu użytkownika
- Implementacje systemu zapisywania wyników lokalnie
- Zapewnienie płynności animacji i efektów wizualnych

## 1.3 Wymagania funkcjonalne

Zdefiniowane zostały następujące wymagania:

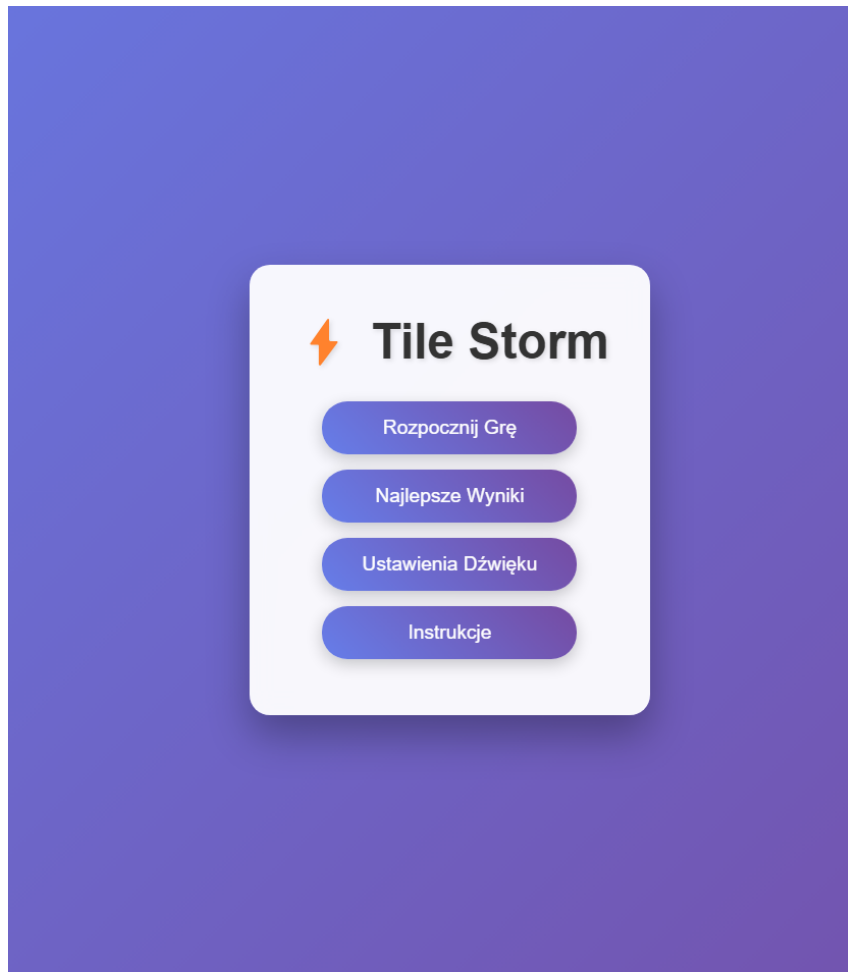
- Podstawowa mechanika gry - klikanie spadających kafelków
- System punktacji z możliwością combo
- Różne typy kafelków z unikalnymi efektami
- Panel ustawień dźwięku
- Tabela najlepszych wyników
- Progresywne zwiększanie trudności

# 2 Przebieg działania gry i wygląd interfejsu

Gra "Tile Storm" została zaprojektowana z myślą o przejrzystości i intuicyjnej obsłudze. Poniżej przedstawiam najważniejsze widoki oraz przebieg rozgrywki wraz z omówieniem kluczowych elementów interfejsu.

## 2.1 Ekran główny (menu startowe)

Po uruchomieniu gry użytkownik widzi główne menu, z którego może rozpocząć rozgrywkę, przejść do tabeli wyników, ustawić dźwięk lub zapoznać się z instrukcjami.



Rysunek 1: Ekran główny gry z podstawowymi opcjami menu.

## 2.2 Instrukcje dla gracza

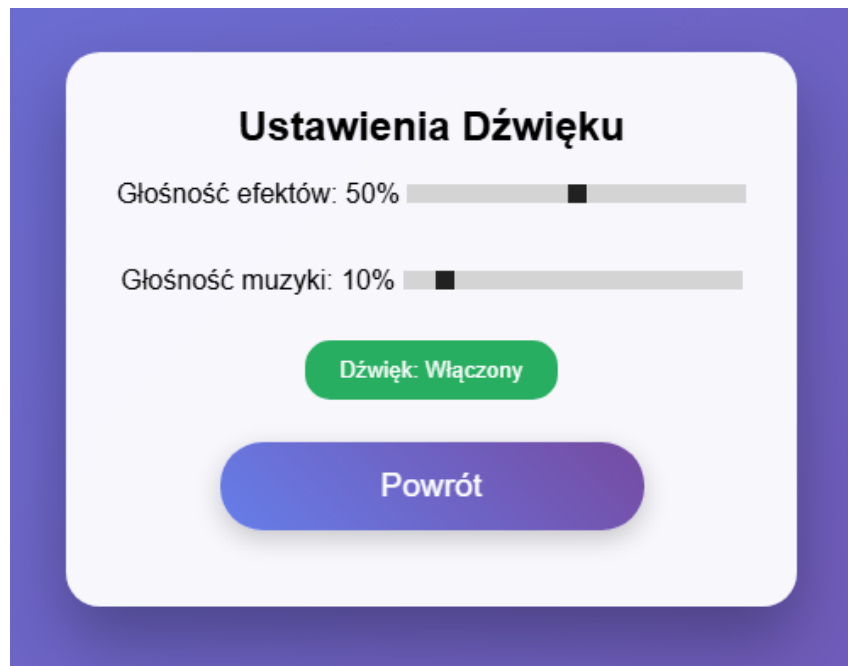
Z poziomu menu gracz może przejść do sekcji instrukcji, gdzie wyjaśnione są zasady rozgrywki, typy kafelków oraz mechanika punktacji.



Rysunek 2: Widok instrukcji gry.

## 2.3 Panel ustawień dźwięku

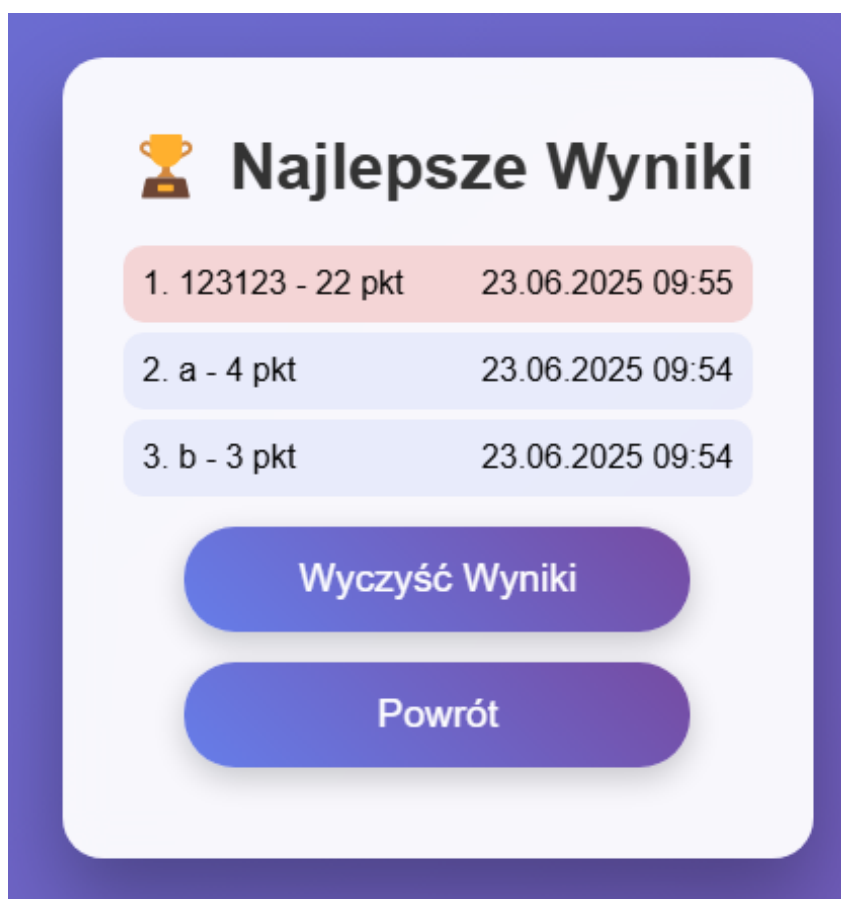
Dostępny jest także panel ustawień, gdzie można sterować głośnością efektów, muzyki oraz włączyć lub wyłączyć dźwięk.



Rysunek 3: Panel ustawień dźwięku.

## 2.4 Tabela najlepszych wyników

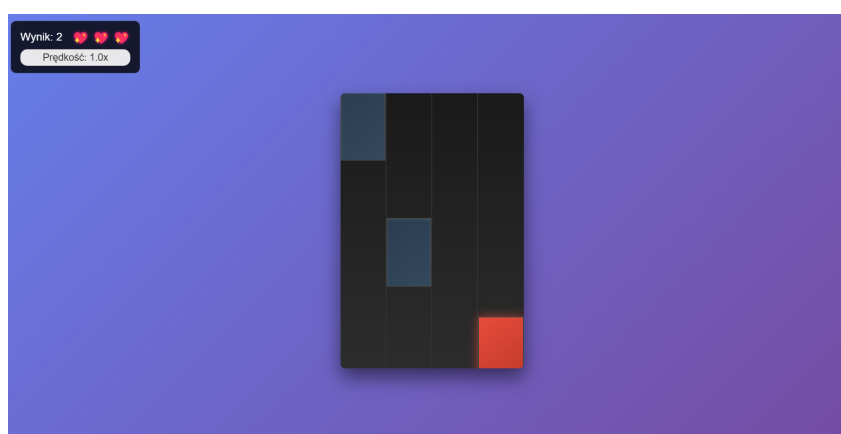
Gracz może w każdej chwili sprawdzić tabele najlepszych wyników oraz porównać swój wynik z poprzednimi rozgrywkami.



Rysunek 4: Tabela najlepszych wyników.

## 2.5 Przebieg rozgrywki

Rozgrywka polega na klikaniu spadających kafelków w odpowiednich torach. Na planszy widoczne są również aktualny wynik, liczba żyć oraz predkość gry.

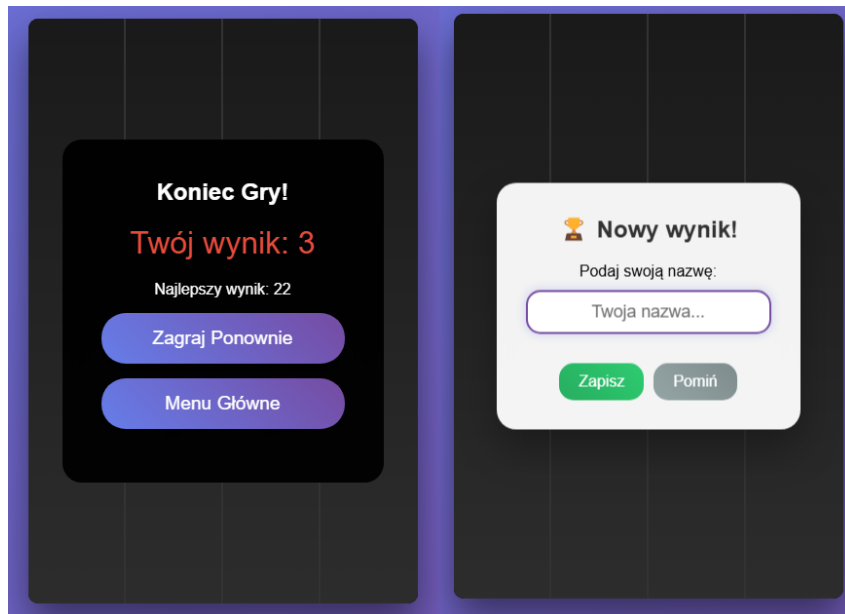


Rysunek 5: Aktywny widok gry podczas rozgrywki.



## 2.6 Ekran zakończenia gry

Po utracie wszystkich żyć lub zakończeniu rozgrywki, pojawia się ekran podsumowujący wynik z możliwością powrotu do menu lub rozpoczęcia nowej gry.



Rysunek 6: Ekran zakończenia gry (Game Over).

## 3 Wykorzystane Technologie

### 3.1 HTML5

Struktura dokumentu została zbudowana w oparciu o semantyczne znaczniki HTML5. Wykorzystano nowoczesne elementy takie jak:

- `<input type="range">` - dla suwaków głośności
- `<meta name="viewport">` - dla responsywności
- Semantyczne klasy CSS dla lepszej organizacji

### 3.2 CSS3

Implementacja stylów wykorzystuje zaawansowane funkcje CSS3:

- **Flexbox** - do pozycjonowania elementów
- **CSS Grid** - nie został wykorzystany w tym projekcie
- **Transforms 3D** - dla animacji kafelków
- **Keyframes** - dla złożonych animacji
- **Media Queries** - dla responsywności
- **CSS Variables** - nie zostały zastosowane

### 3.3 JavaScript ES6+

Kod JavaScript wykorzystuje nowoczesne funkcjonalności:

- **Classes** - dla klasy AudioGenerator
- **Arrow functions** - w callback'ach
- **Template literals** - nie zostały szerzej wykorzystane
- **Destructuring** - nie został zastosowany
- **Async/Await** - nie było potrzeby w tym projekcie

### 3.4 Web Audio API

Wykorzystanie Web Audio API do:

- Generowania tonów pianina w czasie rzeczywistym
- Tworzenia prostej muzyki tła
- Implementacji efektów dźwiękowych
- Kontroli głośności i mikowania

## 4 Architektura Aplikacji

### 4.1 Struktura projektu

Projekt zrealizowany został jako Single Page Application (SPA) zawarta w jednym pliku HTML. Taka struktura została wybrana ze względu na:

- Prostota wdrożenia
- Brak potrzeby serwera
- Łatwość debugowania
- Możliwość uruchomienia bezpośrednio z przeglądarki

### 4.2 Organizacja kodu CSS

Style zostały pogrupowane w logiczne sekcje:

1. **Resetowanie stylów** - zerowanie margin i padding
2. **Layout główny** - pozycjonowanie kontenerów
3. **Komponenty menu** - stylowanie interfejsu nawigacji
4. **Elementy gry** - kafelki, plansze, efekty
5. **Animacje** - definicje keyframes
6. **Responsywność** - media queries

## 4.3 Wzorce projektowe w JavaScript

W kodzie zastosowano następujące wzorce:

### 4.3.1 Singleton Pattern

Obiekt `gameState` działa jako singleton przechowujący stan gry:

```
1 let gameState = {  
2   isPlaying: false,  
3   isPaused: false,  
4   score: 0,  
5   speed: 1,  
6   combo: 0,  
7   lives: 3,  
8   tiles: [],  
9   lastTileTime: 0,  
10  tileInterval: 600,  
11  gameStartTime: 0,  
12  soundEnabled: true,  
13  volume: 0.5,  
14  musicVolume: 0.1  
15 };
```

Listing 1: Implementacja stanu gry

### 4.3.2 Object Literal Pattern

Elementy DOM przechowywane są w obiekcie `elements`:

```
1 const elements = {  
2   mainMenu: document.getElementById('mainMenu'),  
3   gameBoard: document.getElementById('gameBoard'),  
4   gameUI: document.getElementById('gameUI'),  
5   gameOver: document.getElementById('gameOver'),  
6   // ... pozosta e elementy  
7 };
```

Listing 2: Referencje do elementów DOM

## 5 Implementacja Mechanik Gry

### 5.1 System kafelków

Zaimplementowano różne typy kafelków z unikalnymi właściwościami:

Typ	Opis implementacji	Klasa CSS
Normalny	Podstawowy kafelek, standardowe zachowanie	<code>.piano-tile</code>
Bomba	Dodaje emoji i efekt eksplozji	<code>.piano-tile.bomb</code>
Życie	Zielony kafelek z emoji serca	<code>.piano-tile.heal</code>
Multi-hit	Wyświetla liczbę wymaganych kliknięć	<code>.piano-tile.multi-hit</code>
Speed boost	Fioletowy kafelek z efektem świecenia	<code>.piano-tile.speed-boost</code>

Tabela 1: Implementowane typy kafelków

## 5.2 Algorytm spawnu kafelków

Kafelki są generowane w regularnych interwałach z elementem losowości:

```
1 function createTile() {
2   const now = Date.now();
3   if (now - gameState.lastTileTime < gameState.tileInterval) return;
4
5   const lane = Math.floor(Math.random() * 4);
6   const tile = document.createElement('div');
7
8   // Określenie typu kafelka na podstawie prawdopodobieństwa
9   const rand = Math.random();
10  if (rand < 0.02 && now - gameState.lastBombTime > 10000) {
11    tile.className = 'piano-tile bomb';
12    gameState.lastBombTime = now;
13  } else if (rand < 0.05 && gameState.lives < 3) {
14    tile.className = 'piano-tile heal';
15  } else {
16    tile.className = 'piano-tile';
17  }
18
19  // Pozycjonowanie i dodanie do DOM
20  tile.style.top = '0px';
21  lane.appendChild(tile);
22  gameState.tiles.push({element: tile, lane: laneIndex});
23  gameState.lastTileTime = now;
24 }
```

Listing 3: Funkcja tworzenia kafelka

## 5.3 System kolizji i interakcji

Implementacja obsługi kliknięć wykorzystuje event delegation:

```
1 document.addEventListener('click', function(event) {
2   if (!gameState.isPlaying) return;
3
4   const tile = event.target.closest('.piano-tile');
5   if (!tile) return;
6
7   event.preventDefault();
8
9   // Sprawdzenie czy kafelek jest w obszarze kliknięcia
10  const rect = tile.getBoundingClientRect();
11  const boardRect = elements.gameBoard.getBoundingClientRect();
12
13  if (rect.bottom >= boardRect.bottom - 100) {
14    handleTileClick(tile);
15  }
16 });
```

Listing 4: Obsługa kliknięć kafelków

## 6 System Audio

### 6.1 Klasa AudioGenerator

Utworzono dedykowana klasę do zarządzania dźwiękiem:

```
1 class AudioGenerator {
2     constructor() {
3         this.audioContext = null;
4         this.backgroundMusicSource = null;
5         this.gameOverMusicSource = null;
6         this.initAudioContext();
7     }
8
9     initAudioContext() {
10        try {
11            this.audioContext = new (window.AudioContext ||
12                                   window.webkitAudioContext)();
13        } catch (error) {
14            console.warn('Audio Context nie jest wspierany:', error);
15        }
16    }
17 }
```

Listing 5: Konstruktor klasy AudioGenerator

### 6.2 Generowanie dźwięków pianina

Implementacja wykorzystuje oscylatory do tworzenia tonów:

```
1 playPianoNote(frequency = 440, duration = 0.3) {
2     if (!this.audioContext || !gameState.soundEnabled) return;
3
4     const oscillator = this.audioContext.createOscillator();
5     const gainNode = this.audioContext.createGain();
6
7     oscillator.type = 'triangle';
8     oscillator.frequency.setValueAtTime(frequency,
9                                         this.audioContext.currentTime);
10
11     gainNode.gain.setValueAtTime(0, this.audioContext.currentTime);
12     gainNode.gain.linearRampToValueAtTime(gameState.volume,
13                                           this.audioContext.currentTime +
14                                           0.01);
15     gainNode.gain.exponentialRampToValueAtTime(0.01,
16                                                this.audioContext.
17                                                    currentTime + duration)
18     ;
19
20     oscillator.connect(gainNode);
21     gainNode.connect(this.audioContext.destination);
22
23     oscillator.start(this.audioContext.currentTime);
24     oscillator.stop(this.audioContext.currentTime + duration);
25 }
```

Listing 6: Generowanie dźwięku pianina

## 6.3 Muzyka tła

Zaimplementowano system generowania prostej muzyki tła opartej na sekwencji nut:

```
1 createBackgroundMusic() {
2   if (!this.audioContext || !gameState.soundEnabled) return;
3
4   const melody = [523.25, 587.33, 659.25, 698.46]; // C5, D5, E5, F5
5   let noteIndex = 0;
6
7   const playNote = () => {
8     if (!gameState.isPlaying) return;
9
10    const oscillator = this.audioContext.createOscillator();
11    const noteGain = this.audioContext.createGain();
12
13    oscillator.frequency.setValueAtTime(melody[noteIndex],
14                                       this.audioContext.currentTime
15                                       );
16
17    oscillator.type = 'sine';
18
19    // Konfiguracja envelope
20    noteGain.gain.setValueAtTime(0, this.audioContext.currentTime);
21    noteGain.gain.linearRampToValueAtTime(0.05,
22                                         this.audioContext.
23                                         currentTime + 0.1);
24    noteGain.gain.exponentialRampToValueAtTime(0.01,
25                                              this.audioContext.
26                                              currentTime + 0.8);
27
28    oscillator.connect(noteGain);
29    noteGain.connect(this.audioContext.destination);
30
31    oscillator.start(this.audioContext.currentTime);
32    oscillator.stop(this.audioContext.currentTime + 1);
33
34    noteIndex = (noteIndex + 1) % melody.length;
35    setTimeout(playNote, 1000);
36  };
37
38  playNote();
39 }
```

Listing 7: Tworzenie muzyki tła

## 7 Interfejs Użytkownika

### 7.1 Responsywny design

Zaimplementowano responsywność poprzez media queries:

```
1 @media (max-width: 768px) {
2   .game-board {
3     width: 90vw;
4     max-width: 400px;
5   }
6 }
```

```
7   .main-menu {
8       padding: 20px;
9       margin: 20px;
10  }
11
12  .menu-title {
13      font-size: 2em;
14  }
15
16  .score-display {
17      font-size: 1.2em;
18      padding: 10px 15px;
19  }
20 }
```

Listing 8: Media queries dla urządzeń mobilnych

## 7.2 Animacje CSS

Utworzono bibliotekę animacji wykorzystującą keyframes:

```
1 @keyframes pulse {
2     0% { transform: scale(1); }
3     50% { transform: scale(1.05); }
4     100% { transform: scale(1); }
5 }
6
7 .piano-tile.active {
8     animation: pulse 0.3s ease-in-out;
9 }
```

Listing 9: Animacja pulsowania kafelka

## 7.3 System modalnych okien

Zaimplementowano system okien modalnych bez zewnętrznych bibliotek:

```
1 .name-input-modal {
2     position: absolute;
3     top: 50%;
4     left: 50%;
5     transform: translate(-50%, -50%);
6     background: rgba(255, 255, 255, 0.95);
7     padding: 30px;
8     border-radius: 20px;
9     text-align: center;
10    box-shadow: 0 20px 40px rgba(0, 0, 0, 0.3);
11    backdrop-filter: blur(10px);
12    display: none;
13    z-index: 250;
14 }
```

Listing 10: Style dla okien modalnych

## 8 Zarządzanie Stanem

### 8.1 Cykl życia gry

Zaimplementowano kompletny cykl zarządzania stanem gry:

1. **Inicjalizacja** - ustawienie początkowych wartości
2. **Start gry** - uruchomienie głównej petli
3. **Gameplay** - obsługa interakcji gracza
4. **Pauza** - zatrzymanie i wznowienie
5. **Game Over** - zakończenie i zapis wyników
6. **Reset** - powrót do stanu początkowego

### 8.2 Główna petla gry

Zaimplementowano główną petle wykorzystującą requestAnimationFrame:

```
1 function gameLoop() {
2   if (!gameState.isPlaying || gameState.isPaused) return;
3
4   updateTiles();
5   createTile();
6   checkCollisions();
7   updateGameSpeed();
8   updateUI();
9
10  requestAnimationFrame(gameLoop);
11 }
12
13 function updateTiles() {
14   gameState.tiles.forEach((tileData, index) => {
15     const tile = tileData.element;
16     const currentTop = parseInt(tile.style.top) || 0;
17     const newTop = currentTop + (2 + gameState.speed);
18
19     tile.style.top = newTop + 'px';
20
21     // Sprawdzenie czy kafelek opu ci plansz
22     if (newTop > elements.gameBoard.offsetHeight) {
23       handleMissedTile(tile);
24       gameState.tiles.splice(index, 1);
25     }
26   });
27 }
```

Listing 11: Główna petla gry



## 9 Trwałość Danych

### 9.1 System zapisywania wyników

Wykorzystano localStorage do przechowywania danych:

```
1 function saveScore(playerName, score) {
2   let scores = getLeaderboard();
3   scores.push({
4     name: playerName || 'Anonim',
5     score: score,
6     date: new Date().toLocaleDateString('pl-PL')
7   });
8
9   // Sortowanie i ograniczenie do 10 najlepszych
10  scores.sort((a, b) => b.score - a.score);
11  scores = scores.slice(0, 10);
12
13  localStorage.setItem('tileStormScores', JSON.stringify(scores));
14 }
15
16 function getLeaderboard() {
17   try {
18     const scores = localStorage.getItem('tileStormScores');
19     return scores ? JSON.parse(scores) : [];
20   } catch (error) {
21     console.warn('Błąd odczytu wynik w:', error);
22     return [];
23   }
24 }
```

Listing 12: Funkcje zarządzania wynikami

### 9.2 Zarządzanie ustawieniami

Ustawienia użytkownika są również przechowywane lokalnie:

```
1 function saveSettings() {
2   const settings = {
3     volume: gameState.volume,
4     musicVolume: gameState.musicVolume,
5     soundEnabled: gameState.soundEnabled
6   };
7   localStorage.setItem('tileStormSettings', JSON.stringify(settings));
8 }
9
10 function loadSettings() {
11   try {
12     const settings = localStorage.getItem('tileStormSettings');
13     if (settings) {
14       const parsed = JSON.parse(settings);
15       gameState.volume = parsed.volume || 0.5;
16       gameState.musicVolume = parsed.musicVolume || 0.1;
17       gameState.soundEnabled = parsed.soundEnabled !== false;
18     }
19   } catch (error) {
20     console.warn('Błąd wczytywania ustawie:', error);
21   }
```

22 }

Listing 13: Zapisywanie ustawień

## 10 Efekty Wizualne

### 10.1 Animacje kafelków

Zaimplementowano różnorodne animacje dla różnych stanów kafelków:

```
1 @keyframes bombPulse {
2   0% {
3     box-shadow: 0 0 15px rgba(231, 76, 60, 0.6);
4     transform: scale(1);
5   }
6   100% {
7     box-shadow: 0 0 25px rgba(231, 76, 60, 0.9);
8     transform: scale(1.03);
9   }
10 }
11
12 @keyframes healGlow {
13   0% {
14     box-shadow: 0 0 15px rgba(39, 174, 96, 0.6);
15     filter: brightness(1);
16   }
17   100% {
18     box-shadow: 0 0 25px rgba(39, 174, 96, 0.9);
19     filter: brightness(1.2);
20   }
21 }
```

Listing 14: Animacje efektów specjalnych

### 10.2 Efekty czasteczkowe

Dodano proste efekty czasteczkowe dla eksplozji:

```
1 .bomb-explosion {
2   position: absolute;
3   top: 0;
4   left: 0;
5   right: 0;
6   bottom: 0;
7   background: radial-gradient(circle,
8     rgba(231, 76, 60, 0.9) 0%,
9     rgba(255, 193, 7, 0.8) 30%,
10    transparent 70%);
11   animation: explosion 1s ease-out;
12   pointer-events: none;
13   z-index: 75;
14 }
15
16 @keyframes explosion {
17   0% { opacity: 0; transform: scale(0); }
18   50% { opacity: 1; transform: scale(1.5); }
```

```
19 100% { opacity: 0; transform: scale(2); }
20 }
```

Listing 15: Efekt eksplozji

## 11 Obsługa Błędów

### 11.1 Walidacja danych wejściowych

Zaimplementowano podstawowa walidacje:

```
1 function validatePlayerName(name) {
2   if (!name || typeof name !== 'string') {
3     return 'Anonim';
4   }
5
6   // Usuni cie potencjalnie niebezpiecznych znak w
7   return name.replace(/[<>]/g, '').substring(0, 20) || 'Anonim';
8 }
```

Listing 16: Walidacja nazwy gracza

### 11.2 Obsługa błędów Audio API

Dodano obsługę błędów dla Web Audio API:

```
1 initAudioContext() {
2   try {
3     this.audioContext = new (window.AudioContext ||
4                             window.webkitAudioContext)();
5   } catch (error) {
6     console.warn('Audio Context nie jest wspierany:', error);
7     // Graceful degradation - gra dzia a bez d wi ku
8   }
9 }
```

Listing 17: Obsługa błędów audio

## 12 Testowanie i Debugowanie

### 12.1 Metody testowania

Podczas rozwoju projektu zastosowano następujące metody testowania:

- **Testowanie manualne** - sprawdzanie funkcjonalności w różnych przeglądarkach
- **DevTools** - analiza wydajności i pamięci
- **Testowanie responsywności** - sprawdzanie na różnych rozdzielczościach

## 12.2 Znalezione problemy i rozwiązania

W trakcie implementacji napotkano następujące problemy:

1. **Problem:** Audio autoplay blokowany przez przeglądarki
2. **Rozwiązanie:** Uruchomienie audio dopiero po pierwszej interakcji użytkownika
3. **Problem:** Niestabilna wydajność animacji na słabszych urządzeniach
4. **Rozwiązanie:** Użycie CSS transforms zamiast zmiany właściwości position
5. **Problem:** Memory leaks w kontekście audio
6. **Rozwiązanie:** Prawidłowe zatrzymywanie oscylatorów po użyciu

## 13 Wnioski i Obserwacje

### 13.1 Napotkane wyzwania

Podczas realizacji projektu główne wyzwania obejmowały:

- Synchronizacja animacji z logiką gry
- Zarządzanie pamięcią w kontekście Web Audio API
- Zapewnienie responsywności na różnych urządzeniach
- Implementacja płynnych animacji bez zewnętrznych bibliotek
- Obsługa różnic między przeglądarkami w implementacji audio

### 13.2 Nabycie umiejętności

Realizacja projektu pozwoliła na praktyczne poznanie:

- Zaawansowanych możliwości Web Audio API
- Technik optymalizacji wydajności aplikacji webowych
- Implementacji animacji CSS keyframes
- Zarządzania stanem aplikacji w vanilla JavaScript
- Tworzenia responsywnych interfejsów użytkownika
- Obsługi localStorage i trwałości danych

### 13.3 Możliwości rozwoju

Projekt może być rozwinięty w następujących kierunkach:

- Implementacja edytora poziomów
- Integracja z zewnętrznymi API muzycznymi (Spotify Web API)
- Przepisanie na framework (React, Vue.js)
- Dodanie Progressive Web App funkcjonalności
- Implementacja systemu achievementów

### 13.4 Refleksje techniczne

Projekt potwierdził możliwości nowoczesnych technologii webowych w tworzeniu złożonych aplikacji interaktywnych. Szczególnie Web Audio API okazało się potężnym narzędziem, choć wymagającym starannej obsługi memory management.

Zastosowanie vanilla JavaScript pokazało zarówno elastyczność, jak i złożoność zarządzania stanem bez frameworków. W przyszłych projektach warto rozważyć użycie bibliotek wspomagających, szczególnie przy większej skali aplikacji.

### 13.5 Narzędzia wykorzystane

- Visual Studio Code - edytor kodu
- Git - kontrola wersji
- GitHub - hosting kodu źródłowego