Sprawozdanie z Projektu
Implementacja Gry Webowej "Tile Storm"

Krzysztof Róg

czerwiec 2025

Spis treści

1 Cel i Zakres Projektu

1.1 Opis zadania

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

1.2 Założenia projektowe

Główne założenia projektu obejmowały:

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

1.3 Wymagania funkcjonalne

Zdefiniowane zostały nastepujace wymagania:

- Podstawowa mechanika gry - klikanie spadajacych kafelków
- System punktacji z możliwościa combo
- Różne typy kafelków z unikalnymi efektami
- Panel ustawień dźwieku
- Tabela najlepszych wyników
- Progresywne zwiekszanie trudności

2 Wykorzystane Technologie

2.1 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

2.2 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

2.3 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

2.4 Web Audio API

Wykorzystanie Web Audio API do:

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

3 Architektura Aplikacji

3.1 Struktura projektu

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

- Prostote wdrożenia
- Brak potrzeby serwera
- Łatwość debugowania
- Możliwość uruchomienia bezpośrednio z przegladarki

3.2 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

3.3 Wzorce projektowe w JavaScript

W kodzie zastosowano nastepujace wzorce:

3.3.1 Singleton Pattern

Obiekt `gameState` działa jako singleton przechowujacy stan gry:

```
let gameState = {
    isPlaying: false,
    isPaused: false,
    score: 0,
    speed: 1,
    combo: 0,
    lives: 3,
    tiles: [],
    lastTileTime: 0,
    tileInterval: 600,
    gameStartTime: 0,
    soundEnabled: true,
    volume: 0.5,
    musicVolume: 0.1
};
```

Listing 1: Implementacja stanu gry

3.3.2 Object Literal Pattern

Elementy DOM przechowywane sa w obiekcie `elements`:

```
const elements = {
    mainMenu: document.getElementById('mainMenu'),
    gameBoard: document.getElementById('gameBoard'),
    gameUI: document.getElementById('gameUI'),
    gameOver: document.getElementById('gameOver'),
    // ... pozosta e elementy
};
```

Listing 2: Referencje do elementów DOM

4 Implementacja Mechanik Gry

4.1 System kafelków

Zaimplementowano różne typy kafelków z unikalnymi właściwościami:

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

Tabela 1: Implementowane typy kafelków

4.2 Algorytm spawnu kafelków

Kafelki sa generowane w regularnych interwałach z elementem losowości:

```
function createTile() {
    const now = Date.now();
    if (now - gameState.lastTileTime < gameState.tileInterval) return;

    const lane = Math.floor(Math.random() * 4);
    const tile = document.createElement('div');

    // Okre lenie typu kafelka na podstawie prawdopodobie stwa
    const rand = Math.random();
    if (rand < 0.02 && now - gameState.lastBombTime > 10000) {
        tile.className = 'piano-tile bomb';
        gameState.lastBombTime = now;
    } else if (rand < 0.05 && gameState.lives < 3) {
        tile.className = 'piano-tile heal';
    } else {
        tile.className = 'piano-tile';
    }

    // Pozycjonowanie i dodanie do DOM
    tile.style.top = '0px';
    lane.appendChild(tile);
    gameState.tiles.push({element: tile, lane: laneIndex});
    gameState.lastTileTime = now;
}
```

Listing 3: Funkcja tworzenia kafelka

4.3 System kolizji i interakcji

Implementacja obsługi kliknieć wykorzystuje event delegation:

```
document.addEventListener('click', function(event) {
    if (!gameState.isPlaying) return;

```

```
    const tile = event.target.closest('.piano-tile');
    if (!tile) return;

    event.preventDefault();

    // Sprawdzenie czy kafelek jest w obszarze klikni cia
    const rect = tile.getBoundingClientRect();
    const boardRect = elements.gameBoard.getBoundingClientRect();

    if (rect.bottom >= boardRect.bottom - 100) {
        handleTileClick(tile);
    }
});
```

Listing 4: Obsługa kliknieć kafelków

5 System Audio

5.1 Klasa AudioGenerator

Utworzono dedykowana klase do zarzadzania dźwiekiem:

```
class AudioGenerator {
    constructor() {
        this.audioContext = null;
        this.backgroundMusicSource = null;
        this.gameOverMusicSource = null;
        this.initAudioContext();
    }

    initAudioContext() {
        try {
            this.audioContext = new (window.AudioContext ||
                                     window.webkitAudioContext)();
        } catch (error) {
            console.warn('Audio Context nie jest wspierany:', error);
        }
    }
}
```

Listing 5: Konstruktor klasy AudioGenerator

5.2 Generowanie dźwieków pianina

Implementacja wykorzystuje oscylatory do tworzenia tonów:

```
playPianoNote(frequency = 440, duration = 0.3) {
    if (!this.audioContext || !gameState.soundEnabled) return;

    const oscillator = this.audioContext.createOscillator();
    const gainNode = this.audioContext.createGain();

    oscillator.type = 'triangle';
    oscillator.frequency.setValueAtTime(frequency,
                                        this.audioContext.currentTime);

```

```
    gainNode.gain.setValueAtTime(0, this.audioContext.currentTime);
    gainNode.gain.linearRampToValueAtTime(gameState.volume,
                                          this.audioContext.currentTime +
                                              0.01);
    gainNode.gain.exponentialRampToValueAtTime(0.01,
                                               this.audioContext.
                                                   currentTime + duration)
                                                   ;

    oscillator.connect(gainNode);
    gainNode.connect(this.audioContext.destination);

    oscillator.start(this.audioContext.currentTime);
    oscillator.stop(this.audioContext.currentTime + duration);
}
```

Listing 6: Generowanie dźwieku pianina

5.3 Muzyka tła

Zaimplementowano system generowania prostej muzyki tła opartej na sekwencji nut:

```
createBackgroundMusic() {
    if (!this.audioContext || !gameState.soundEnabled) return;

    const melody = [523.25, 587.33, 659.25, 698.46]; // C5, D5, E5, F5
    let noteIndex = 0;

    const playNote = () => {
        if (!gameState.isPlaying) return;

        const oscillator = this.audioContext.createOscillator();
        const noteGain = this.audioContext.createGain();

        oscillator.frequency.setValueAtTime(melody[noteIndex],
                                            this.audioContext.currentTime
                                               );
        oscillator.type = 'sine';

        // Konfiguracja envelope
        noteGain.gain.setValueAtTime(0, this.audioContext.currentTime);
        noteGain.gain.linearRampToValueAtTime(0.05,
                                              this.audioContext.
                                                 currentTime + 0.1);
        noteGain.gain.exponentialRampToValueAtTime(0.01,
                                                   this.audioContext.
                                                      currentTime + 0.8);

        oscillator.connect(noteGain);
        noteGain.connect(this.audioContext.destination);

        oscillator.start(this.audioContext.currentTime);
        oscillator.stop(this.audioContext.currentTime + 1);

        noteIndex = (noteIndex + 1) % melody.length;
        setTimeout(playNote, 1000);
    };

```

```
    playNote();
}
```

Listing 7: Tworzenie muzyki tła

6 Interfejs Użytkownika

6.1 Responsywny design

Zaimplementowano responsywność poprzez media queries:

```
@media (max-width: 768px) {
    .game-board {
        width: 90vw;
        max-width: 400px;
    }

    .main-menu {
        padding: 20px;
        margin: 20px;
    }

    .menu-title {
        font-size: 2em;
    }

    .score-display {
        font-size: 1.2em;
        padding: 10px 15px;
    }
}
```

Listing 8: Media queries dla urzadzeń mobilnych

6.2 Animacje CSS

Utworzono biblioteke animacji wykorzystujaca keyframes:

```
@keyframes pulse {
    0% { transform: scale(1); }
    50% { transform: scale(1.05); }
    100% { transform: scale(1); }
}

.piano-tile.active {
    animation: pulse 0.3s ease-in-out;
}
```

Listing 9: Animacja pulsowania kafelka

6.3 System modalnych okien

Zaimplementowano system okien modalnych bez zewnetrznych bibliotek:

```css
.name-input-modal {
    position: absolute;
    top: 50%;
    left: 50%;
    transform: translate(-50%, -50%);
    background: rgba(255, 255, 255, 0.95);
    padding: 30px;
    border-radius: 20px;
    text-align: center;
    box-shadow: 0 20px 40px rgba(0, 0, 0, 0.3);
    backdrop-filter: blur(10px);
    display: none;
    z-index: 250;
}
```

Listing 10: Style dla okien modalnych

7 Zarzadzanie Stanem

7.1 Cykl życia gry

Zaimplementowano kompletny cykl zarzadzania stanem gry:

1. **Inicjalizacja** - ustawienie poczatkowych 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 poczatkowego

7.2 Główna petla gry

Zaimplementowano główna petle wykorzystujaca requestAnimationFrame:

```javascript
function gameLoop() {
    if (!gameState.isPlaying || gameState.isPaused) return;

    updateTiles();
    createTile();
    checkCollisions();
    updateGameSpeed();
    updateUI();

    requestAnimationFrame(gameLoop);
}

function updateTiles() {
    gameState.tiles.forEach((tileData, index) => {
        const tile = tileData.element;
        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

8 Trwałość Danych

8.1 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('B   d odczytu wynik w:', error);
22         return [];
23     }
24 }
```

Listing 12: Funkcje zarzadzania wynikami

8.2 Zarzadzanie ustawieniami

Ustawienia użytkownika sa również przechowywane lokalnie:

```
1  function saveSettings() {
2      const settings = {
3          volume: gameState.volume,
4          musicVolume: gameState.musicVolume,
5          soundEnabled: gameState.soundEnabled
```

```
    };
    localStorage.setItem('tileStormSettings', JSON.stringify(settings));
}

function loadSettings() {
    try {
        const settings = localStorage.getItem('tileStormSettings');
        if (settings) {
            const parsed = JSON.parse(settings);
            gameState.volume = parsed.volume || 0.5;
            gameState.musicVolume = parsed.musicVolume || 0.1;
            gameState.soundEnabled = parsed.soundEnabled !== false;
        }
    } catch (error) {
        console.warn('B   d wczytywania ustawie :', error);
    }
}
```

Listing 13: Zapisywanie ustawień

9 Efekty Wizualne

9.1 Animacje kafelków

Zaimplementowano różnorodne animacje dla różnych stanów kafelków:

```
@keyframes bombPulse {
    0% {
        box-shadow: 0 0 15px rgba(231, 76, 60, 0.6);
        transform: scale(1);
    }
    100% {
        box-shadow: 0 0 25px rgba(231, 76, 60, 0.9);
        transform: scale(1.03);
    }
}

@keyframes healGlow {
    0% {
        box-shadow: 0 0 15px rgba(39, 174, 96, 0.6);
        filter: brightness(1);
    }
    100% {
        box-shadow: 0 0 25px rgba(39, 174, 96, 0.9);
        filter: brightness(1.2);
    }
}
```

Listing 14: Animacje efektów specjalnych

9.2 Efekty czasteczkowe

Dodano proste efekty czasteczkowe dla eksplozji:

```
.bomb-explosion {
    position: absolute;
```

```
    top: 0;
    left: 0;
    right: 0;
    bottom: 0;
    background: radial-gradient(circle,
                                rgba(231, 76, 60, 0.9) 0%,
                                rgba(255, 193, 7, 0.8) 30%,
                                transparent 70%);
    animation: explosion 1s ease-out;
    pointer-events: none;
    z-index: 75;
}

@keyframes explosion {
    0% { opacity: 0; transform: scale(0); }
    50% { opacity: 1; transform: scale(1.5); }
    100% { opacity: 0; transform: scale(2); }
}
```

Listing 15: Efekt eksplozji

10 Obsługa Błedów

10.1 Walidacja danych wejściowych

Zaimplementowano podstawowa walidacje:

```
function validatePlayerName(name) {
    if (!name || typeof name !== 'string') {
        return 'Anonim';
    }

    // Usuni cie potencjalnie niebezpiecznych znak w
    return name.replace(/[<>]/g, '').substring(0, 20) || 'Anonim';
}
```

Listing 16: Walidacja nazwy gracza

10.2 Obsługa błedów Audio API

Dodano obsługe błedów dla Web Audio API:

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

Listing 17: Obsługa błedów audio

11 Testowanie i Debugowanie

11.1 Metody testowania

Podczas rozwoju projektu zastosowano nastepujace metody testowania:

- **Testowanie manualne** - sprawdzanie funkcjonalności w różnych przegladarkach
- **DevTools** - analiza wydajności i pamieci
- **Testowanie responsywności** - sprawdzanie na różnych rozdzielczościach

11.2 Znalezione problemy i rozwiazania

W trakcie implementacji napotkano nastepujace problemy:

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

12 Wnioski i Obserwacje

12.1 Napotkane wyzwania

Podczas realizacji projektu główne wyzwania obejmowały:

- Synchronizacja animacji z logika gry
- Zarzadzanie pamiecia w kontekście Web Audio API
- Zapewnienie responsywności na różnych urzadzeniach
- Implementacja płynnych animacji bez zewnetrznych bibliotek
- Obsługa różnic miedzy przegladarkami w implementacji audio

12.2 Nabyte umiejetności

Realizacja projektu pozwoliła na praktyczne poznanie:

- Zaawansowanych możliwości Web Audio API
- Technik optymalizacji wydajności aplikacji webowych
- Implementacji animacji CSS keyframes

- Zarzadzania stanem aplikacji w vanilla JavaScript
- Tworzenia responsywnych interfejsów użytkownika
- Obsługi localStorage i trwałości danych

12.3 Możliwości rozwoju

Projekt może być rozwiniety w nastepujacych kierunkach:

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

12.4 Refleksje techniczne

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

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

12.5 Narzedzia wykorzystane

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