Uniwersytet Śląski

INTERAKTYWNE APLIKACJE MULTIMEDIALNE

Dokumentacja projektu zaliczeniowego gry Scoop-Shooting wykonanej w technologii JavaScript + Canvas

Wykonali:

Daniel Gandyra

Przemysław Zgoda-Giczański

Grupa PAW 3

1. Wstęp

Aplikacja webowa prostej gry przeglądarkowej, wykorzystująca interakcję użytkownika poprzez lewy przycisk myszy. Użytkownik zdobywa punkty za każde trafienie w cel, są one widoczne w górnej części ekranu. Wrogowie pojawiają się w losowych miejsach oraz zbliżają się w kierunku gracza. Gra kończy się w momencie kontaktu przeciwnika z modelem gracza. Po zakończeniu gry wyświetlany jest komunikat o zdobytej ilości punktów oraz możliwości ponownej rozgrywki. Aplikacja napisana przy użyciu języka JavaScript oraz HTML5 Canvas, który umożliwia renderowania kształtów i obrazów bitmapowych bez dodatkowych wtyczek.

2. Specyfikacja wewnętrzna

Do korzystania z naszej aplikacji niezbędna będzie przeglądarka internetowa oraz mysz komputerowa lub touchpad, po rozpakowaniu archiwum wystarczy uruchomić plik index.html. Wszystkie istotne funkcje aplikacji znajdują się w pliku index.js.

Obiektem odpowiedzialnym za stworzenie modelu gracza jest class ***Player{}***, określamy w nim takie parametry jak współrzędne ‘x’ oraz ‘y’ odpowiedzialne za rozlokowanie gracza w środku pola gry. Radius określa promień naszego gracza, color określa barwę naszego gracza.

```
class Player {
    constructor(x, y, radius, color) {
        this.x = x;
        this.y = y;
        this.radius = radius;
        this.color = color;
    }

    draw() {
        c.beginPath();
        c.arc(this.x, this.y, this.radius, 0, Math.PI * 2, false);
        c.fillStyle = this.color;
        c.fill();
    }
}
```

Rysunek 2.1 przedstawiający obiekt Player{}

Funkcja ***draw()*** odpowiada za wyświetlenie gracza na ekranie, ***beginPath()*** określa początek rysowania modelu, za okrągły kształt odpowiedzialna jest metoda ***arc()***, w której definiujemy koordynaty x oraz y, promień, oraz początek i koniec rysowania. ***fillStyle*** oraz ***fill*** odpowiadają za wypełnienie okręgu zadeklarowanym kolorem.

```
const x = canvas.width / 2;
const y = canvas.height / 2;

let player = new Player(x, y, 15, "pink");
```

Rysunek 2.2 przedstawiający zadeklarowanie gracza oraz współrzędnych środka ekranu

Obiekt odpowiedzialny za tworzenie modelu pocisków, konstrukcja następuje w sposób analogiczny do klasy ***Player{}*** z uwzględnieniem i dodaniem prędkości pociskom ***velocity***. Występuje tutaj także funkcja ***update()***, która na bieżąco aktualizuje położenie pocisku.

```
class Projectile {
    constructor(x, y, radius, color, velocity) {
        this.x = x;
        this.y = y;
        this.radius = radius;
        this.color = color;
        this.velocity = velocity;
    }
    draw() {
        c.beginPath();
        c.arc(this.x, this.y, this.radius, 0, Math.PI * 2, false);
        c.fillStyle = this.color;
        c.fill();
    }

    update() {
        this.draw();
        this.x = this.x + this.velocity.x;
        this.y = this.y + this.velocity.y;
    }
}
```

Rysunek 2.3 przedstawiający zadeklarowanie pocisków

Math.atan2() odpowiedzialne jest za obliczenie kąta między miejscem kliknięcia myszy a środkiem pola gry.

Następnie określamy prędkość za pomocą ***Math.cos***, jest ona odpowiedzialna za prędkość w linii X, natomiast ***Math.sin*** w linii Y.

```
addEventListener("click", (event) => {
    const angle = Math.atan2(
        event.clientY - canvas.height / 2,
        event.clientX - canvas.width / 2
    );

    const velocity = {
        x: Math.cos(angle) * 2,
        y: Math.sin(angle) * 2,
    };
    projectiles.push(
        new Projectile(canvas.width / 2, canvas.height / 2, 5, "gold", velocity)
    );
});
```

Rysunek 2.4 przedstawiający metodę umożliwiającą interakcję przy klikaniu myszką

Obiekt odpowiedzialny za tworzenie wrogów, stworzony analogicznie do klasy tworzącej pociski - ***Projectile{}***

```
class Enemy {
    constructor(x, y, radius, color, velocity) {
        this.x = x;
        this.y = y;
        this.radius = radius;
        this.color = color;
        this.velocity = velocity;
    }
    draw() {
        c.beginPath();
        c.arc(this.x, this.y, this.radius, 0, Math.PI * 2, false);
        c.fillStyle = this.color;
        c.fill();
    }

    update() {
        this.draw();
        this.x = this.x + this.velocity.x;
        this.y = this.y + this.velocity.y;
    }
}
```

Rysunek 2.5 przedstawiający zadeklarowanie przeciwników

Funkcja ***spawnEnemies()*** odpowiedzialna jest za generowanie wrogów. ***0 - radius*** tworzy wroga w początkowym punkcie osi X wyłączając promień wroga, natomiast ***canvas.width + radius*** tworzy go w maksymalnym punkcie osi X z całym promieniem. Analogicznie ma to miejsce w przypadku osi Y, jedyną zmianą jest zastąpienie losowego miejsca osi przeciwnej.

color wykorzystana losowa wartość koloru z zakresu ***hsl()*** od 0 - 360
angle odpowiada za pobieranie lokalizacji gracza w celu ukierunkowania wrogów w jego stronę.
enemies.push() służy do dodawania nowej instancji przeciwnika do tablicy ***enemies = [ ]***

```
function spawnEnemies() {
    setInterval(() => {
        const radius = Math.random() * (30 - 6) + 6;
        let x;
        let y;
        if (Math.random() < 0.5) {
            x = Math.random() < 0.5 ? 0 - radius : canvas.width + radius;
            y = Math.random() * canvas.height;
        } else {
            x = Math.random() * canvas.width;

            y = Math.random() < 0.5 ? 0 - radius : canvas.height + radius;
        }

        const color = `hsl(${Math.random() * 360}, 50%, 50%)`;

        const angle = Math.atan2(canvas.height / 2 - y, canvas.width / 2 - x);

        const velocity = {
            x: Math.cos(angle),
            y: Math.sin(angle),
        };

        enemies.push(new Enemy(x, y, radius, color, velocity));
    }, 1000);
}
```

Rysunek 2.6 funkcja odpowiedzialna za generowanie przeciwników

requestAnimationFrame - odpowiada za zapętlenie wszelkich animacji
Math.hypot() - obliczamy dystans pomiędzy pociskiem a wrogiem oraz między graczem a wrogiem.
if (dist - enemy.radius - projectile/player.radius < 1 dzięki temu możemy wyznaczyć kiedy wróg ma zniknąć przy interakcji z pociskiem oraz zakończyć grę w przypadku zetknięcia wroga z graczem.
enemies.splice() - usuwa z tablicy enemies[] trafionego pociskiem przeciwnika
projectiles.splice() - usuwa z tablicy pocisk, który przekroczył granicę pola gry

```
function animate() {
    animationId = requestAnimationFrame(animate);
    c.fillStyle = "rgba(0,0,0,0.1)";
    c.fillRect(0, 0, canvas.width, canvas.height);
    player.draw();
    projectiles.forEach((projectile, index) => {
        projectile.update();

        //usuń z krawędzi ekranu
        if (
            projectile.x + projectile.radius < 0 ||
            projectile.x - projectile.radius > canvas.width ||
            projectile.y + projectile.radius < 0 ||
            projectile.y - projectile.radius > canvas.height
        ) {
            setTimeout(() => {
                projectiles.splice(index, 1);
            }, 0);
        }
    });

    enemies.forEach((enemy, index) => {
        enemy.update();

        const dist = Math.hypot(player.x - enemy.x, player.y - enemy.y);
        //koniec gry
        if (dist - enemy.radius - player.radius < 1) {
            cancelAnimationFrame(animationId);
            modalEl.style.display = "flex";
            scoreUp.innerHTML = score;
        }

        projectiles.forEach((projectile, projectileIndex) => {
            const dist = Math.hypot(projectile.x - enemy.x, projectile.y - enemy.y);

            if (dist - enemy.radius - projectile.radius < 1) {
                score += 100;
                scoreEl.innerHTML = score;

                setTimeout(() => {
                    enemies.splice(index, 1);
                    projectiles.splice(projectileIndex, 1);
                }, 0);
            }
        });
    });
}
```

Rysunek 2.7 funkcja odpowiedzialna za wszelkie animacji gry

Funkcja ***init()*** pozwoli nam na ustawienie tablicy przeciwników oraz pocisków na puste wartości, zresetowanie wyniku oraz wygenerowanie nowego gracza.

```
function init() {
    player = new Player(x, y, 15, "pink");
    projectiles = [];
    enemies = [];
    score = 0;
    scoreEl.innerHTML = score;
    scoreUp.innerHTML = score;
}
```

Rysunek 2.8 funkcja odpowiedzialna za inicjacje nowej rozgrywki

Metoda ***start.addEventListener()*** wywołuje poszczególne funkcje w momencie aktywacji przycisku startu gry

```
start.addEventListener("click", () => {
    audio.play();
    init();
    animate();
    spawnEnemies();
    modalEl.style.display = "none";
});
```

Rysunek 2.9 wykonanie akcji przycisku start gry