Projekt gra „CORN QUEST”

1) **Opis gry**

   W grze wcielamy się w jednorożca, którego zadaniem jest skakanie po chmurach i zbieranie ziarenek kukurydzy. Gracz musi uważać na przeciwników, którymi są kolby kukurydzy. Przeciwnicy strzelają do gracza szarymi ziarenkami, a po każdym trafieniu gracz traci jedno serduszko z pięciu. Po utracie wszystkich serduszek gra się kończy.

2) **Działanie**

Przed rozpoczęciem gry na canvasie wyświetlany jest ekran startowy.

```
if (!gameStarted) { //jeśli gra jest nie rozpoczęta wyświetl ekran startowy
   if (startScreen.complete) {
       ctx.drawImage(startScreen, 0, 0, canvas.width, canvas.height);
       } else { //jeśli obrazek się nie ładuje wyświetla się niebieski
prostokąt
       ctx.fillStyle = 'blue';
       ctx.fillRect(0, 0, canvas.width, canvas.height);
       }
       requestAnimationFrame(gameLoop);
       return;
       }
```

Po wciśnięciu przycisku start gra rozpoczyna się.

```
  startButton.addEventListener("click", () => {
  gameStarted = true;
  startButton.style.display = "none";
  requestAnimationFrame(gameLoop);
});
```

Gdy gracz zginie, na canvasie wyświetla się ekran z napisem „GAME OVER”. Jeśli zdrowie gracza ma wartość powyżej 0, gra jest kontynuowana.

```
if (health > 0) { //jeśli poziom zdrowia gracza jest większy od 0
wyświetlamy grę
    requestAnimationFrame(gameLoop);
  } else { // w innym przypadku wyświetlamy ekran gameover
    if (gameOverScreen.complete) {
      ctx.drawImage(gameOverScreen, 0, 0, canvas.width, canvas.height);
    } else { // jeśli obrazek się nie załaduje wyświetla się zielony
prostokąt
      ctx.fillStyle = 'green';
      ctx.fillRect(0, 0, canvas.width, canvas.height);
    }
  }
```

a) **Gracz**

Graczem poruszamy za pomocą klawiszy W (skok), A (w lewo), D (w prawo). W funkcji updatePlayer definiujemy grawitacje i siłę skoku gracza. Do prędkości pionowej gracza

dodajemy siłę skoku. Jeśli wciśnięto W i gracz nie jest w trakcie skoku, gracz skacze w górę.

```
  const gravity = 0.24; //grawitacja
  const jumpStrength = -10; //siła skoku (gracz skacze 10px w górę)

   //obsługa skoku
  if (keysPressed['w'] && !player.isJumping) {
    player.vY = jumpStrength; //nadanie prędkości skoku
    player.isJumping = true; //gracz skacze (jest w trakcie skoku)
  }

  player.vY += gravity; //dodanie grawitacji do prędkości
  player.y += player.vY; //przesunięcie gracza w górę
```

Sprawdzamy kolizję gracza z górną krawędzią chmury. Jeśli gracz skoczył na chmurę(i nie skacze w tym momencie) i dotknął jej górnej krawędzi ustawiamy pozycje gracza na górnej krawędzi chmury.

```
  for (let cloud of clouds) {
    if ( //gdy gracz opada sprawdzamy czy dotknął górnej krawędzi chmury
      player.vY >= 0 &&
      player.x + player.sizeX > cloud.x &&
      player.x < cloud.x + cloud.sizeX &&
      player.y + player.sizeY <= cloud.y + 10 &&
      player.y + player.sizeY + player.vY >= cloud.y
    ) {
      player.y = cloud.y - player.sizeY; //ustawienie pozycji gracza na
górnej krawędzi chmury
      player.vY = 0; //zatrzymanie ruchu w pionie
      player.isJumping = false; //gracz nie skacze
      onPlatform = true; //gracz stoi na chmurze
      break;
    }
  }
```

Następnie sprawdzamy kolizje gracza z polem, na którym jest kukurydza. Jeśli gracz dotknie kukurydzy jego wynik zwiększa się o jeden punkt, a kukurydza znika.

```
for (let cloud of clouds) {
    if (cloud.hasCorn) {
      const cornRect = { //pole (prostokąt), w którym znajduje się kukurydza
        x: cloud.x + cloud.sizeX / 2 - 10,
        y: cloud.y - 20,
        sizeX: 20,
        sizeY: 20
      };

      //gdy gracz dotknie kukurydzy wynik zwiększa się o jeden, a kukurydza
znika
      if (checkCollision(player, cornRect)) {
        cloud.hasCorn = false;
        score++;
      }
```

```
    }
  }
```

Jeśli gracz się porusza zwiększamy licznik (klatek) oraz przechodzimy do następnej klatki, a następnie resetujemy licznik. Gdy gracz stoi w miejscu wyświetla się domyślna klatka.

```
if (isMoving) {
    player.frameCounter++; //jeśli gracz się porusz zwiększ licznik klatek
    if (player.frameCounter > 10) { //co 10 klatek zmień klatkę animacji
      player.currentFrame = (player.currentFrame + 1) %
player.imgFrames.length; //przejście do następnej klatki
      player.frameCounter = 0; //reset licznika
    }
  } else {
    player.currentFrame = 0; // domyślna klatka, gdy player stoi
    player.frameCounter = 0; //reset licznika, gdy player stoi
  }
```

Za każdym razem, gdy gracz się porusza w bok wyświetlają się kolejne klatki rysowane w funkcji drawPlayer. Gdy gracz porusza się w lewo animacja jest odbijana lustrzanie.

```
const flip = keysPressed['a'] && !keysPressed['d']; //warunek potrzebny do
odbicia lustrzanego gracza
  const img = player.imgFrames[player.currentFrame]; //obecna klatka
animacji

  if (img.complete) { //rysowanie klatki animacji
    ctx.save();
    ctx.scale(flip ? -1 : 1, 1);  //lustrzane odbicie postaci w lewo
    //jeśli wciśnięto 'a' i nie wciśnięto 'd' odbij lustrzanie obrazek
gracza
    const drawX = flip ? -player.x - player.sizeX : player.x;
    ctx.drawImage(img, drawX, player.y, player.sizeX, player.sizeY);
    ctx.restore();
  } else { //jeśli klatka się nie ładuje wyświetlany jest szary prostokąt
    ctx.fillStyle = 'gray';
    ctx.fillRect(player.x, player.y, player.sizeX, player.sizeY);
  }
```

b) Strzelanie

Pociskami (tęczą) strzelamy za pomocą klawisza S, kierunek/miejsce wyznaczamy za pomocą wskaźnika myszki (pocisk leci do punktu, w którym aktualnie znajduje się wskaźnik myszy). Po każdym ruchu myszy pobierana jest aktualna pozycja wskaźnika względem całej strony. Następnie obliczamy pozycje kursora względem canvasu odejmując od odczytanych pozycji wskaźnika pozycję canvasu.

```
    let mouseX = 0; // pozycja x wskaźnika myszy
    let mouseY = 0; // pozycja y wskaźnika myszy

    //rejestrowanie pozycji myszy
    canvas.addEventListener('mousemove', (e) => {
    const rect = canvas.getBoundingClientRect();
    mouseX = e.clientX - rect.left; //obliczona pozycja x kursora
```

```
względem canvasu
    mouseY = e.clientY - rect.top; //obliczona pozycja y kursora
względem  canvasu
    });
```

Po wciśnięciu klawisza S uruchamia się funkcja shoot i gracz strzela pociskiem. Przytrzymując przycisk gracz strzela w kółko przyciskami w ustalonym odstępie 20 klatek.

```
let shootCooldown = 0; //opóźnienie między strzałami

 if (keysPressed['s'] && shootCooldown <= 0) {
    shoot(); //wywołanie funkcji strzelania
    shootCooldown = 20; //opóźnienie między strzałami
  }
  if (shootCooldown > 0) shootCooldown--;
```

W funkcji shoot ustawiamy startową pozycję pocisku (środek gracza). Następnie obliczamy wektor kierunku pocisku od środka gracza do wskaźnika myszki. Na koniec do tablicy rainbows dodajemy pocisk z obliczoną pozycją i składowymi prędkości.

```
      function shoot() {
  //pozycja startowa pocisku
      const startX = player.x + player.sizeX / 2;
      const startY = player.y + player.sizeY / 2;
      //obliczanie wektora kierunku pocisku (od gracza do
      wskaźnika myszki)
      const dx = mouseX - startX;
      const dy = mouseY - startY;
      const length = Math.sqrt(dx * dx + dy * dy); //obliczanie długości
      wektora

  //dodanie nowego pocisku do tablicy rainbows
  rainbows.push({
    x: startX,
    y: startY,
    sizeX: rainbow.sizeX,
    sizeY: rainbow.sizeY,
    vx: (dx / length) * rainbow.speed, //składowa X wektora prędkości
    vy: (dy / length) * rainbow.speed //składowa Y wektora prędkości
  });
}
```

Pociski są rysowane w funkcji drawRainbows. W funkcji updateRainbows iterujemy od końca po tablicy rainbows. Aktualizujemy pozycje przycisku zgodnie z jego prędkością (obliczonymi składowymi prędkości).

```
for (let i = rainbows.length - 1; i >= 0; i--) {
    const r = rainbows[i];
    //aktualizacja pozycji przycisku zgodnie z jego prędkością
    r.x += r.vx;
    r.y += r.vy;
```

Następnie sprawdzamy kolizję pocisku z polem przeciwnika. Po każdej kolizji przycisk znika. Po trafieniu przeciwnika zwiększa się licznik trafień, a gdy trafimy 3 razy przeciwnik ginie i staje się „ukryty” (cloud.hasEnemy = false; - to jest potrzebne do warunku do rysowania obrazka zabitego przeciwnika w funkcji drawClouds)

```
for (let cloud of clouds) {
      if (cloud.hasEnemy && cloud.enemyHitCount < 3) {
        const enemyRect = { //prostokąt (pole) w którym znajduje się
przeciwnik
          x: cloud.x + cloud.sizeX / 2 - enemy.sizeX / 2,
          y: cloud.y - enemy.sizeY,
          sizeX: enemy.sizeX,
          sizeY: enemy.sizeY
        };

        // Jeśli przycisk trafi w przeciwnika zwiększa sie licznik
trafień i pocisk znika
        if (checkCollision(r, enemyRect)) {
          cloud.enemyHitCount++;
          rainbows.splice(i, 1);
```

c) Chmury

Chmury są tworzone w funkcji spawnClouds. Do wcześniej utworzonej tablicy dynamicznej dodajemy chmury na dwóch różnych pozycjach (chmury na wyższej wysokości są rysowane 500px poza canvasem). Każda chmura oprócz pozycji, obrazka, szybkości poruszania oraz rozmiaru, posiada też pola hasCorn i hasEnemy, które definiują czy na danej chmurze jest przeciwnik lub kukurydza. Nowe chmury są tworzone co 3 sekundy.

```
function spawnCloud() {
  const hasCorn = Math.random() < 0.5; //50% szans na to, że na chmurze
pojawi się kukurydza
  const hasEnemy = Math.random() < 0.2; //20% szans na to, że na chmurze
pojawi się przeciwnik

  clouds.push({
    x: canvas.width,
    y: canvas.height - 210,
    sizeX: cloud.sizeX,
    sizeY: cloud.sizeY,
    speed: 1,
    hasCorn: hasCorn, //czy zawiera kukurydze
    hasEnemy: hasEnemy, //czy zawiera przeciwnika
    enemyShootCooldown: 0, //opóźnienie między strzałami przeciwnika
    enemyHitCount: 0, //liczba trafień w przeciwnika
    img: cloud.img
  }, {
    x: canvas.width + 500,
    y: canvas.height - 380,
    sizeX: cloud.sizeX,
    sizeY: cloud.sizeY,
```

```
    speed: 1,
    hasCorn: hasCorn, //czy zawiera kukurydze
    hasEnemy: hasEnemy, //czy zawiera przeciwnika
    enemyShootCooldown: 0, //opóźnienie między strzałami przeciwnika
    enemyHitCount: 0, //liczba trafień w przeciwnika
    img: cloud.img
  });
}
setInterval(spawnCloud, 3000);// co 3 sekundy tworzona jest nowa chmura
```

W funkcji updateClouds aktualizowana jest pozycja chmur, każda chmura porusza się z ustaloną prędkością w lewą stronę.

```
function updateClouds() {
  clouds.forEach(cloud => {
    cloud.x -= cloud.speed; //przesuwanie chmury w lewo o prędkość speed
```

Chmury, które wyjdą poza lewą krawędź canvasu są usuwane.

```
  for (let i = clouds.length - 1; i >= 0; i--) {
    if (clouds[i].x + clouds[i].sizeX < 0) clouds.splice(i, 1);
  }
}
```

Chmury są rysowane w funkcji drawClouds.

d) Przeciwnicy

Przeciwnicy są rysowani na chmurach w funkcji drawClouds. Jeśli przeciwnik zginie (zostanie trafiony 3 razy) jego obrazek zmienia się na taki o szarej kolorystyce.

```
  if (cloud.hasEnemy) { //rysowanie przeciwnika na chmurach
      if (enemy.img.complete) {
        ctx.drawImage(enemy.img, cloud.x + cloud.sizeX / 2 -
         enemy.sizeX / 2, cloud.y - enemy.sizeY, enemy.sizeX,
         enemy.sizeY);
      } else { //jeśli obrazek sie nie załaduje zobaczymy czarny
       prostokat
        ctx.fillStyle = 'black';
        ctx.fillRect(cloud.x + cloud.sizeX / 2 - 15, cloud.y - 30,
        enemy.sizeX, enemy.sizeY);
      }
   } else if (cloud.enemyIsKilled) { //jeśli przeciwnik zostanie zabity
    jego obrazek zmienia sie na inny(szary)
      ctx.drawImage(enemy.killedImg, cloud.x + cloud.sizeX / 2 -
      enemy.sizeX / 2, cloud.y - enemy.sizeY, enemy.sizeX, enemy.sizeY);
}
```

Przeciwnicy strzelają do gracza dopóki nie zostaną trafieni 3 razy. W funkcji updateClouds obliczamy początkową pozycję pocisku, a następnie wektor kierunku pocisku od chmury do gracza.

```
      if (cloud.enemyShootCooldown <= 0) {
        const startX = cloud.x + cloud.sizeX / 2;
        const startY = cloud.y - enemy.sizeY / 2;
```

```
        //obliczanie wektora kierunku pocisku (od chmury do gracza)
        const dx = player.x + player.sizeX / 2 - startX;
        const dy = player.y + player.sizeY / 2 - startY;
        const length = Math.sqrt(dx * dx + dy * dy); //obliczanie
długości wektora
        const speed = 2.5; //prędkość pocisku
```

Nowy pocisk z obliczoną pozycją startową i składowymi prędkości dodajemy do tablicy enemyBullets. Następnie ustawiamy opóźnienie między pociskami (w klatkach).

```
        //dodanie nowego pocisku do tablicy
        enemyBullets.push({
          x: startX,
          y: startY,
          sizeX: 10,
          sizeY: 10,
          vx: (dx / length) * speed, // składowa X wektora prędkości
          vy: (dy / length) * speed // składowa Y wektora prędkości
        });

        //ustawienie opóźnienia dla pocisków przeciwnika 180 klatek
        cloud.enemyShootCooldown = 180;
      }
    }
```

Pociski rysowane są w funkcji drawEnemyBullets. W funkcji updateEnemyBullets iterujemy od końca po tablicy enemyBullets. Aktualizujemy pozycje przycisku zgodnie z jego prędkością.

```
for (let i = enemyBullets.length - 1; i >= 0; i--) {
    const bullet = enemyBullets[i];

    //aktualizacja pozycji pocisku zgodnie z jego prędkością
    bullet.x += bullet.vx;
    bullet.y += bullet.vy;
```

Następnie sprawdzamy kolizję pocisku z graczem. Po każdej kolizji przycisk znika. Po trafieniu gracza jego poziom zdrowia spada o 1 punkt. Sprawdzamy, czy poziom zdrowia gracza nie spadł poniżej 0, jeśli nie sprawdzamy kolizję z następnym pociskiem.

```
//jeśli pocisk trafił w gracza zmniejsza się jego zdrowie o jeden, a
pocisk znika
    if (checkCollision(player, bullet)) {
      health--;
      enemyBullets.splice(i, 1);

      //sprawdzanie czy gracz nie zginął
      if (health <= 0) {
        return;
      }
```

```
      continue; //przejście do następnego pocisku
    }
```

Pocisk jest usuwany, gdy wyjdzie poza canvas.

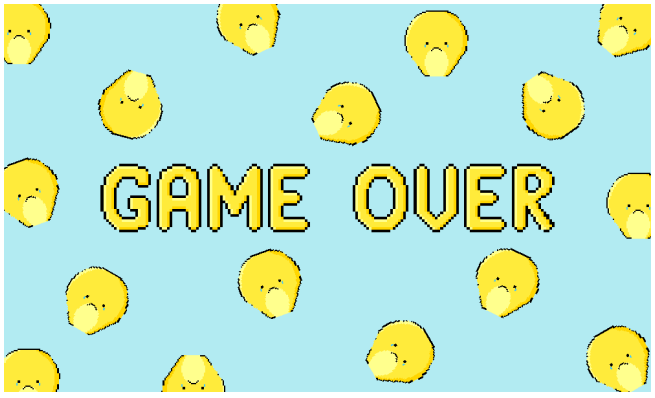
```
// Jeśli pocisk wyleciał poza krawędź ekranu jest usuwany
    if (
      bullet.x < 0 || bullet.x > canvas.width ||
      bullet.y < 0 || bullet.y > canvas.height
    ) {
      enemyBullets.splice(i, 1);
    }
```

3) Grafika

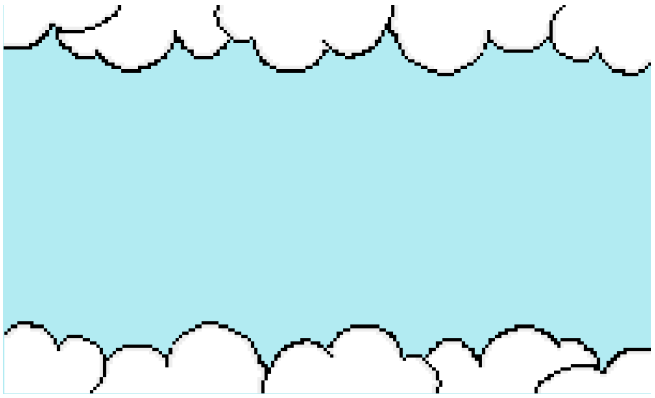
Wszystkie grafiki użyte w grze zostały stworzone za pomocą aplikacji https://www.pixilart.com/ .

a) Ekran startowy

b) Ekran „GAME OVER”



c) Tło



d) Player – klatki animacji

e) Enemy

f) Chmura

g) Kukurydza

h) Serce (health)

i) Pociski gracza

j) Pociski przeciwnika