SYMULACJA RUCHU

Nagłówki i definicje

```
#include <SDL2/SDL.h>
#include <math.h>
#include <stdlib.h>
#include <time.h>
```

Nagłówki te dołączają biblioteki takie jak:

1. **SDL2** - do obsługi grafiki
2. **math.h** - pozwala używać funkcji matematycznych
3. **stdlib.h** - zawiera funkcje do generowania liczb losowych i zarządzania pamięcią
4. **time.h** - służy do pracy z czasem potrzebna do inicjalizacji generatora liczb losowych

Ustawienia programu

```
#define WINDOW_WIDTH 1000
#define WINDOW_HEIGHT 800
#define NUM_PARTICLES 100
#define PARTICLE_RADIUS 20
#define MAX_SPEED 20.0f
```

Są to zmienne globalne które odpowiadają za:

1. **WINDOW_WIDTH** i **WINDOW_HEIGHT** - rozmiary okna symulacji
2. **NUM_PARTICLES** - liczba cząsteczek w symulacji
3. **PARTICLE RADIUS** - promien każdej cząsteczki
4. **MAX_SPEED** - maksymalna prędkość z jaką kulki mogą się poruszać

STRUKTURA CZĄSTECZKI

```
typedef struct {
    float x, y;       // Pozycja cząsteczki
    float dx, dy;     // Prędkość (ruch w kierunku x i y)
    Uint8 r, g, b;    // Kolor cząsteczki (RGB)
} Particle;
```

Jest to struktura pozwalająca na określenie jakie parametry ma przypisana każda cząsteczka:

1. **x, y** - pozycja na ekranie
2. **dx, dy** - prędkość w poziomie i pionie
3. **r, g, b** - kolor cząsteczki w formacie RGB

FUNKCJE POMOCNICZE

```
void drawCircle(SDL_Renderer *renderer, int x, int y, int radius) {
    for (int w = 0; w < radius * 2; w++) {
        for (int h = 0; h < radius * 2; h++) {
            int dx = radius - w;
            int dy = radius - h;
            if ((dx * dx + dy * dy) <= (radius * radius)) {
                SDL_RenderDrawPoint(renderer, x + dx, y + dy);
            }
        }
    }
}
```

Funkcja **drawCircle()** opowiada za narysowanie kółka na ekranie (aktualnie nie jest używana):

1. Przesuwa się przez kwadrat o wymiarach równych średnicy kółka
2. Jeśli punkt **(dx, dy)** znajduje się wewnątrz okręgu (wykorzystując równanie: **$dx^2 + dy^2 <= r^2$**), rysuje go na ekranie

```
int checkCollision(Particle *a, Particle *b) {
    float dist = sqrtf((a->x - b->x) * (a->x - b->x) + (a->y - b->y) * (a->y - b->y));
    return dist < (2 * PARTICLE_RADIUS);
}
```

Funkcja **checkCollision** odpowiada za sprawdzenie czy cząsteczki uderzają o siebie:

1. Oblicza odległość między dwiema cząsteczkami (a i b) za pomocą wzoru Pitagorasa
2. Sprawdza, czy odległość jest mniejsza niż dwa razy promień (czyli czy cząsteczki się dotykają)

```
void changeColor(Particle *p) {
    p->r = rand() % 200;
    p->g = rand() % 200;
    p->b = rand() % 200;
}
```

Funkcja **changeColor()** odpowiada za zmiane koloru cząsteczek:

1. Ustawia losowy kolor dla cząsteczki w zakresie od 0 do 199

```
void limitSpeed(Particle *p) {
    float speed = sqrtf(p->dx * p->dx + p->dy * p->dy);
    if (speed > MAX_SPEED) {
        p->dx = (p->dx / speed) * MAX_SPEED;
        p->dy = (p->dy / speed) * MAX_SPEED;
    }
}
```

Funkcja **limitSpeed()**:

1. Ogranicza prędkość cząsteczki, aby nie przekraczała **MAX_SPEED**

```
void handleCollision(Particle *a, Particle *b) {
    float dx = b->x - a->x;
    float dy = b->y - a->y;
    float distance = sqrtf(dx * dx + dy * dy);

    float overlap = (2 * PARTICLE_RADIUS - distance) / 2.0f;
    float unitDx = dx / distance;
    float unitDy = dy / distance;

    a->x -= unitDx * overlap;
    a->y -= unitDy * overlap;
    b->x += unitDx * overlap;
    b->y += unitDy * overlap;

    float tempDx = a->dx;
    float tempDy = a->dy;
    a->dx = b->dx;
    a->dy = b->dy;
    b->dx = tempDx;
    b->dy = tempDy;

    changeColor(a);
    changeColor(b);

    limitSpeed(a);
    limitSpeed(b);
}
```

Funkcja **handleCollision()** obsługuje zderzenia:

1. Oblicza różnice pozycji i dystans między cząsteczkami
2. Zmienia kierunek cząsteczki na zewnątrz aby się rozdzieliły
3. Zamienia ich prędkości symulując odbicie
4. Zmienia ich kolor
5. Ogranicza prędkość aby symulacja była jak najbardziej realna

FUNKCJA MAIN

```
if (SDL_Init(SDL_INIT_VIDEO) < 0) { ... }
SDL_Window *window = SDL_CreateWindow(...);
SDL_Renderer *renderer = SDL_CreateRenderer(...);
```

Inicjalizacja SDL:

1. Inicjalizuje biblioteke SDL i tworzy okno o wymiarach 1000x800 pikseli oraz redner (do rysowania)

```
Particle particles[NUM_PARTICLES];
for (int i = 0; i < NUM_PARTICLES; i++) {
    particles[i].x = rand() % (WINDOW_WIDTH - 2 * PARTICLE_RADIUS) + PARTICLE_RADIUS;
    particles[i].y = rand() % (WINDOW_HEIGHT - 2 * PARTICLE_RADIUS) + PARTICLE_RADIUS;
    particles[i].dx = (rand() % 2 == 0 ? 1 : -1) * (1 + rand() % 2);
    particles[i].dy = (rand() % 2 == 0 ? 1 : -1) * (1 + rand() % 2);
    particles[i].r = rand() % 200;
    particles[i].g = rand() % 200;
    particles[i].b = rand() % 200;
    limitSpeed(&particles[i]);
}
```

Tworzenie cząsteczek:

1. Generuje podaną liczbę (**NUM_PARTICLES**) cząsteczek
2. Przypisuje im wartości takie jak:
    - Pozycja na ekranie (x, y)
    - Prędkość początkowa (dx, dy)
    - Losowy kolor
    - Ograniczenie prędkości w celu zachowania rzeczywistości symulacji

```
while (running) {
    // Obsługa zdarzeń
    while (SDL_PollEvent(&event)) {
        if (event.type == SDL_QUIT) {
            running = 0;
        }
    }

    // Aktualizacja pozycji cząsteczek
    for (int i = 0; i < NUM_PARTICLES; i++) {
        particles[i].x += particles[i].dx;
        particles[i].y += particles[i].dy;

        // Odbicie od lewej i prawej ściany
        if (particles[i].x - PARTICLE_RADIUS < 0) {
            particles[i].x = PARTICLE_RADIUS; // Korekta pozycji
            particles[i].dx = -particles[i].dx;
        } else if (particles[i].x + PARTICLE_RADIUS > WINDOW_WIDTH) {
            particles[i].x = WINDOW_WIDTH - PARTICLE_RADIUS; // Korekta pozycji
            particles[i].dx = -particles[i].dx;
        }

        // Odbicie od górnej i dolnej ściany
        if (particles[i].y - PARTICLE_RADIUS < 0) {
            particles[i].y = PARTICLE_RADIUS; // Korekta pozycji
            particles[i].dy = -particles[i].dy;
        } else if (particles[i].y + PARTICLE_RADIUS > WINDOW_HEIGHT) {
            particles[i].y = WINDOW_HEIGHT - PARTICLE_RADIUS; // Korekta pozycji
            particles[i].dy = -particles[i].dy;
        }
    }


    // Obsługa kolizji między cząsteczkami
    for (int i = 0; i < NUM_PARTICLES; i++) {
        for (int j = i + 1; j < NUM_PARTICLES; j++) {
            if (checkCollision(&particles[i], &particles[j])) {
                handleCollision(&particles[i], &particles[j]);
            }
        }
    }

    // Wyczyść ekran
    SDL_SetRenderDrawColor(renderer, 0, 0, 0, 255); // Czarny
    SDL_RenderClear(renderer);

    // Rysowanie cząsteczek
    for (int i = 0; i < NUM_PARTICLES; i++) {
```

```
            SDL_SetRenderDrawColor(renderer, particles[i].r, particles[i].g, particles[i].b, 255);
            drawCircle(renderer, (int)particles[i].x, (int)particles[i].y, PARTICLE_RADIUS);
        }

        // Wyświetlenie nowej klatki
        SDL_RenderPresent(renderer);

        // Opóźnienie
        SDL_Delay(16); // Około 60 FPS
    }
```

Główna pętla programu:

1. Obsługa zdarzeń:
    - Sprawdza, czy użytkowanik chce zamknąć okno
2. Aktualizacja pozycji:
    - Przesuwa cząsteczki na podstawie ich prędkości
    - Odbija cząsteczki od krawędzi okna
3. Obsługa kolizji:
    - Sprawdza, czy cząsteki zderzyły się i obsługuje kolizjie
4. Rysowanie:
    - Czyści ekran, rysuje każdą cząsteczke jako kółko i wyświetla wynik
5. Opóźnienie:
    - **SDL_Delay(16)** sprawia, że program działa z prędkością około 60 klatek na sekunde

CZYSZCZENIE ZASOBÓW

```
SDL_DestroyRenderer(renderer);
SDL_DestroyWindow(window);
SDL_Quit();
```

Usuwa zasoby SDL przed zakończeniem programu (co zapobiega niepotrzebnym błędom)