ball.cpp:
```cpp
#include "ball.h"

Ball::Ball(sf::Vector2f startPos, float radius, sf::Vector2f startVel)
{
    velocity = startVel;
    m_shape.setPosition(startPos);
    m_shape.setRadius(radius);
    m_shape.setOrigin(radius, radius);
    m_shape.setFillColor(sf::Color::Green);
}

void Ball::draw(sf::RenderTarget& window)
{
    window.draw(m_shape);
}

sf::FloatRect Ball::getGlobalBounds() const
{
    return m_shape.getGlobalBounds();
}

void Ball::ruch(sf::Time dt, sf::Vector2f win, Paddle& pdl)
{
    m_shape.move(velocity * dt.asSeconds());

    float x = m_shape.getPosition().x;
    float y = m_shape.getPosition().y;
    float r = m_shape.getRadius();

    if (x - r <= 0 || x + r >= win.x)
        velocity.x = -velocity.x;

    if (y - r <= 0)
        velocity.y = -velocity.y;

    if (m_shape.getGlobalBounds().intersects(pdl.getGlobalBounds()))
        velocity.y = -velocity.y;

    if (y + r >= win.y)
    {
        velocity = { 0, 0 };
    }
}
```

ball.h:
```cpp
#pragma once
#include <SFML/Graphics.hpp>
```

```
#include "paddle.h"

class Ball
{
private:
    sf::CircleShape m_shape;
    sf::Vector2f velocity;

public:
    Ball(sf::Vector2f startPos, float radius, sf::Vector2f startVel);
    void ruch(sf::Time dt, sf::Vector2f win, Paddle& pdl);
    void draw(sf::RenderTarget& window);
    void odbijY() { velocity.y = -velocity.y; }
    sf::FloatRect getGlobalBounds() const;
};
```

brick.h:

```
#pragma once
#include <SFML/Graphics.hpp>
#include <array>

class Brick : public sf::RectangleShape
{
private:
    int life;
    bool destroyed;
    static const std::array<sf::Color, 4> LUT;

public:
    Brick(sf::Vector2f pos, sf::Vector2f size, int hp);
    void trafienie();
    bool czyZniszczony() const;
    void draw(sf::RenderTarget& window);
};
```

brick.cpp:

```
#include "brick.h"

const std::array<sf::Color, 4> Brick::LUT =
{
    sf::Color::Transparent,
    sf::Color::Yellow,
    sf::Color::Magenta,
    sf::Color::Red
};

Brick::Brick(sf::Vector2f pos, sf::Vector2f size, int hp)
{
    life = hp;
    destroyed = false;

    setPosition(pos);
    setSize(size);
    setFillColor(LUT[life]);
    setOutlineColor(sf::Color::White);
    setOutlineThickness(1.f);
}

void Brick::trafienie()
{
    if (destroyed) return;

    life--;
    setFillColor(LUT[life]);

    if (life <= 0)
        destroyed = true;
}

bool Brick::czyZniszczony() const
{
    return destroyed;
}

void Brick::draw(sf::RenderTarget& window)
{
    if (!destroyed)
        window.draw(*this);
}
```

game.h:

```
#pragma once
#include <SFML/Graphics.hpp>
#include <vector>
#include "paddle.h"
#include "ball.h"
#include "brick.h"

class Game
```

```
{
private:
    Paddle m_paletka;
    Ball m_pilka;
    std::vector<Brick> m_bloki;

public:
    Game();
    void update(sf::Time dt, sf::Vector2u windowSize);
    void render(sf::RenderTarget& target);

private:
    void loadLevel(sf::Vector2u windowSize);
};
```

game.cpp:

```
#include "Game.h"
#include <iostream>

Game::Game()
    : m_paletka({ 400.f, 550.f }, { 200.f, 20.f }, { 300.f, 0.f }),
    m_pilka({ 400.f, 300.f }, 20.f, { 200.f, 200.f })
{
    loadLevel({ 800, 600 }); // Domyślnie ładuje poziom
}

void Game::loadLevel(sf::Vector2u windowSize)
{
    m_bloki.clear();

    const int COLS = 12;
    const int ROWS = 4;
    float blockW = (windowSize.x - (COLS - 1) * 2.f) / COLS;
    float blockH = 20.f;

    for (int y = 0; y < ROWS; y++) {
        for (int x = 0; x < COLS; x++) {

            float px = x * (blockW + 2.f);
            float py = y * (blockH + 2.f) + 60.f;

            int life = (y == 0 ? 2 : 1);

            m_bloki.emplace_back(sf::Vector2f(px, py),
                sf::Vector2f(blockW, blockH),
                life);
        }
    }
}

void Game::update(sf::Time dt, sf::Vector2u windowSize)
{
    m_paletka.ruch(dt, sf::Vector2f(windowSize.x, windowSize.y));
    m_pilka.ruch(dt, sf::Vector2f(windowSize.x, windowSize.y), m_paletka);
    // kolizje pilki z blokami
    for (auto& blk : m_bloki) {
        if (!blk.czyZniszczony() &&
            m_pilka.getGlobalBounds().intersects(blk.getGlobalBounds()))
        {
            blk.trafienie();
            m_pilka.odbijY();
        }
    }

    // usuwanie blokow od konca
    for (int i = (int)m_bloki.size() - 1; i >= 0; i--) {
        if (m_bloki[i].czyZniszczony())
            m_bloki.erase(m_bloki.begin() + i);
    }
}

void Game::render(sf::RenderTarget& target)
{
    m_paletka.draw(target);
    m_pilka.draw(target);

    for (auto& blk : m_bloki)
        blk.draw(target);
}
```

main.cpp:

```
#include <SFML/Graphics.hpp>
#include "Menu.h"
#include "Game.h"

enum class GameState { Menu, Playing, Exiting };

int main()
{
    sf::RenderWindow window(sf::VideoMode(800, 600), "Arkanoid");
    window.setFramerateLimit(60);

    Menu menu(window.getSize().x, window.getSize().y);
    Game game;

    GameState currentState = GameState::Menu;

    sf::Clock deltaClock;

    while (window.isOpen())
    {
        sf::Event event;

        while (window.pollEvent(event))
        {
            if (event.type == sf::Event::Closed)
                window.close();
            if (currentState == GameState::Menu)
            {
                if (event.type == sf::Event::KeyPressed)
                {
                    if (event.key.code == sf::Keyboard::Up)
                        menu.MoveUp();

                    else if (event.key.code == sf::Keyboard::Down)
                        menu.MoveDown();

                    else if (event.key.code == sf::Keyboard::Enter)
                    {
                        if (menu.getSelectedItem() == 0)
                            currentState = GameState::Playing;
                        else if (menu.getSelectedItem() == 1)
                            window.close();
                    }
                }
            }
        }

        sf::Time dt = deltaClock.restart();

        if (currentState == GameState::Playing)
            game.update(dt, window.getSize());

        window.clear();

        if (currentState == GameState::Menu)
            menu.draw(window);
        else if (currentState == GameState::Playing)
            game.render(window);

        window.display();
    }

    return 0;
}
```

menu.cpp:

```
#include "Menu.h"

Menu::Menu(float width, float height)
{
    if (!font.loadFromFile("arial.ttf"))
    {
        std::cout << "Błąd: nie można wczytać czcionki arial.ttf!\n";
    }

    selectedIndex = 0;

    // --- OPCJA 1: NOWA GRA ---
    menu[0].setFont(font);
    menu[0].setFillColor(sf::Color::Red);     // zaznaczona
    menu[0].setString("Nowa gra");
    menu[0].setCharacterSize(40);
    menu[0].setPosition(sf::Vector2f(width / 2.f - 100, height / 3.f));

    // --- OPCJA 2: WYJSCIE ---
    menu[1].setFont(font);
    menu[1].setFillColor(sf::Color::White);
    menu[1].setString("Wyjscie");
    menu[1].setCharacterSize(40);
    menu[1].setPosition(sf::Vector2f(width / 2.f - 100, height / 3.f + 80));
}

void Menu::draw(sf::RenderWindow& window)
{
    for (int i = 0; i < MAX_NUMBER_OF_ITEMS; i++)
        window.draw(menu[i]);
}

void Menu::MoveUp()
{
    if (selectedIndex - 1 >= 0)
    {
        menu[selectedIndex].setFillColor(sf::Color::White);
        selectedIndex--;
        menu[selectedIndex].setFillColor(sf::Color::Red);
    }
}

void Menu::MoveDown()
{
    if (selectedIndex + 1 < MAX_NUMBER_OF_ITEMS)
    {
        menu[selectedIndex].setFillColor(sf::Color::White);
        selectedIndex++;
        menu[selectedIndex].setFillColor(sf::Color::Red);
    }
}
```

menu.h:

```
#pragma once
#include <SFML/Graphics.hpp>
#include <iostream>

#define MAX_NUMBER_OF_ITEMS 2   // "Nowa gra", "Wyjście"

class Menu
{
private:
    int selectedIndex;
    sf::Font font;
    sf::Text menu[MAX_NUMBER_OF_ITEMS];

public:
    Menu(float width, float height);

    void draw(sf::RenderWindow& window);
    void MoveUp();
    void MoveDown();
    int getSelectedItem() const { return selectedIndex; }
};
```

paddle.cpp:

```
#include "paddle.h"

Paddle::Paddle(sf::Vector2f startPos, sf::Vector2f size, sf::Vector2f startVel)
{
    velocity = startVel;
    m_shape.setPosition(startPos);
    m_shape.setSize(size);
    m_shape.setFillColor(sf::Color::Cyan);
    m_shape.setOrigin(size.x / 2.f, size.y / 2.f);
}

void Paddle::ruch(sf::Time dt, sf::Vector2f windowSize)
{
    float dx = 0;

    if (sf::Keyboard::isKeyPressed(sf::Keyboard::A))
        dx -= velocity.x * dt.asSeconds();

    if (sf::Keyboard::isKeyPressed(sf::Keyboard::D))
        dx += velocity.x * dt.asSeconds();

    sf::Vector2f pos = m_shape.getPosition();
    pos.x += dx;

    float half = m_shape.getSize().x / 2.f;

    if (pos.x - half < 0)
        pos.x = half;
    if (pos.x + half > windowSize.x)
        pos.x = windowSize.x - half;

    m_shape.setPosition(pos);
}

void Paddle::draw(sf::RenderTarget& window)
{
    window.draw(m_shape);
}

sf::FloatRect Paddle::getGlobalBounds() const
{
    return m_shape.getGlobalBounds();
}
```

paddle.h:

```
#pragma once
#include <SFML/Graphics.hpp>

class Paddle
{
private:
    sf::RectangleShape m_shape;
    sf::Vector2f velocity;

public:
    Paddle(sf::Vector2f startPos, sf::Vector2f size, sf::Vector2f startVel);
    void ruch(sf::Time dt, sf::Vector2f windowSize);
    void draw(sf::RenderTarget& window);
    sf::FloatRect getGlobalBounds() const;
};
```