ball.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:
```
#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);
    int getHP() const { return life; }
};
```

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;
    sf::Vector2f blockSize;

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

    sf::Vector2f getBlockSize() const { return blockSize; }
    Paddle& getPaddle() { return m_paletka; }
    Ball& getBall() { return m_pilka; }
    std::vector<Brick>& getBlocks() { return m_bloki; }


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

main.cpp:

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

enum class GameStateFlag { Menu, Playing, Exiting };

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

    Menu menu(window.getSize().x, window.getSize().y);
    //Menu menu(800, 600);
    Game game;
    GameState saveState;

    GameStateFlag currentState = GameStateFlag::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 && event.type == sf::Event::KeyPressed)
            {
                // 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)
                {
                    int choice = menu.getSelectedItem();

                    if (choice == 0) {
                        currentState = GameState::Playing;
                    }
                    else if (choice == 1)
                    {
                        if (gs.loadFromFile("save.txt"))
                        {
                            gs.apply(game.getPaddle(), game.getBall(), game.getBlocks(), game.getBlockSize());
                            currentState = GameState::Playing;
                            std::cout << "Gra wczytana!\n";
                        }
                        else
                        {
                            std::cout << "Brak pliku save.txt!\n";
                        }
                    }
                    else if (choice == 2) {
                        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;
    menu[0].setFont(font);
    menu[0].setFillColor(sf::Color::Red);
    menu[0].setString("Nowa gra");
    menu[0].setCharacterSize(40);
    menu[0].setPosition(sf::Vector2f(width / 2.f - 100, height / 3.f));

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

    menu[2].setFont(font);
    menu[2].setFillColor(sf::Color::White);
    menu[2].setString("Wyjscie");
    menu[2].setCharacterSize(40);
    menu[2].setPosition(sf::Vector2f(width / 2.f - 100, height / 3.f + 160));
}

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 > 0)
    {
        menu[selectedIndex].setFillColor(sf::Color::White);
        selectedIndex--;
        menu[selectedIndex].setFillColor(sf::Color::Red);
    }
}

void Menu::MoveDown()
{
    if (selectedIndex < MAX_NUMBER_OF_ITEMS - 1)
    {
        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 3

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);

   sf::Vector2f getPosition() const { return m_shape.getPosition(); }
   void setPosition(sf::Vector2f pos) { m_shape.setPosition(pos); }


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

ScoreManager.cpp:

```
#include "ScoreManager.h"
#include <fstream>

void GameState::capture(const Paddle& p, const Ball& b, const std::vector<Brick>& stones)
{
        paddlePosition = p.getPosition();
        ballPosition = b.getPosition();
        ballVelocity = b.getVelocity();

        blocks.clear();
        for (const auto& blk : stones)
        {
                blocks.push_back(
                        {
                                blk.getPosition().x,
                                blk.getPosition().y,
                                blk.getHP()
                        });
        }
}

bool GameState::saveToFile(const std::string& filename)
```

```
{
	std::ofstream file(filename);
	if (!file.is_open())
		return false;

	file << "PADDLE " << paddlePosition.x << " " << paddlePosition.y << "\n";
	file << "BALL " << ballPosition.x << " " << ballPosition.y << " "
		<< ballVelocity.x << " " << ballVelocity.y << "\n";

	file << "BLOCKS_COUNT " << blocks.size() << "\n";

	for (auto& b : blocks)
		file << b.x << " " << b.y << " " << b.hp << "\n";

	return true;
}

bool GameState::loadFromFile(const std::string& filename)
{
	std::ifstream file(filename);
	if (!file.is_open()) return false;

	std::string label;

	file >> label >> paddlePosition.x >> paddlePosition.y;
	file >> label >> ballPosition.x >> ballPosition.y >> ballVelocity.x >> ballVelocity.y;

	int count = 0;
	file >> label >> count;

	blocks.clear();
	for (int i = 0; i < count; i++)
	{
		float x, y;
		int hp;
		file >> x >> y >> hp;
		blocks.push_back({ x, y, hp });
	}

	return true;
}

void GameState::apply(Paddle& p, Ball& b, std::vector<Brick>& stones, sf::Vector2f blockSize)
{
	p.setPosition(paddlePosition);
	b.setState(ballPosition, ballVelocity);

	stones.clear();
	for (auto& bd : blocks)
		stones.emplace_back(sf::Vector2f(bd.x, bd.y), blockSize, bd.hp);
}
```

ScoreManager.h:

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

struct BlockData
{
	float x, y;
	int hp;
};

class GameState
{
private:
	sf::Vector2f paddlePosition;
	sf::Vector2f ballPosition;
	sf::Vector2f ballVelocity;

public:
	std::vector<BlockData> blocks;

	GameState() = default;

	void capture(const Paddle& p, const Ball& b, const std::vector<Brick>& stones);
	bool saveToFile(const std::string& filename);
	bool loadFromFile(const std::string& filename);
	void apply(Paddle& p, Ball& b, std::vector<Brick>& stones, sf::Vector2f blockSize);
};
```