Projekt informatyka II Arkanoid w C++

AUTHOR:Alan Maksym s204179

Indeks klas

tc Lista klas

Tutaj znajdują się klasy, struktury, unie i interfejsy wraz z ich krótkimi opisami:

Indeks plików

tc Lista plików

Tutaj znajduje się lista wszystkich plików wraz z ich krótkimi opisami:

Dokumentacja klastc

Dokumentacja struktury BlockData

Struktura pomocnicza przechowująca dane pojedynczego bloku (pozycję i punkty życia) na potrzeby zapisu stanu gry.

tc \l

```
#include <GameState.h>
```

Atrybuty publiczne

- float **x**
  Współrzędna X zapisanego bloku.

- float **y**
  Współrzędna Y zapisanego bloku.

- int **hp**
  Liczba całkowita odpowiadająca punktom życia bloku.

Dokumentacja dla tej struktury została wygenerowana z pliku:

GameState.h

Dokumentacja klasy Game

Główny silnik gry zarządzający pętlą logiczną, aktualizacją fizyki oraz renderowaniem wszystkich obiektów.
tc \l

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

Metody publiczne

- **Game** ()
  Konstruktor inicjalizujący okno, obiekty gry (piłkę, paletkę, kamienie) oraz parametry początkowe.

- void **update** (sf::Time dt)
  Aktualizuje logikę gry (ruch obiektów, kolizje) w oparciu o upływ czasu dt.

- void **render** (sf::RenderTarget &target)
  Wyświetla wszystkie obiekty gry (piłka, paletka, bloki, wynik) na ekranie.

- bool **isGameOver** () const
  Zwraca informację (true/false), czy gra została zakończona.

- void **resetGame** ()
  Przywraca stan gry do wartości początkowych (nowa gra).

- void **saveGame** (const std::string &filename)
  Zapisuje bieżący stan gry do pliku tekstowego.

- bool **loadGame** (const std::string &filename)
  Wczytuje stan gry z pliku, przywracając pozycje obiektów i wynik.

Dokumentacja dla tej klasy została wygenerowana z plików:

Game.h

Game.cpp

Dokumentacja klasy GameState

Klasa odpowiedzialna za system zapisu i odczytu (serializację) kompletnego stanu rozgrywki z pliku
tc \l

```
#include <GameState.h>
```

Metody publiczne

- void **capture** (const **Paletka** &p, const **Pilka** &b, const std::vector< **Stone** > &stones, const **wynik** &currentscore)
  Pobiera aktualne dane z obiektów gry i zapisuje je w wewnętrznych strukturach klasy.

- bool **saveToFile** (const std::string &filename)
  Zapisuje przechwycone dane (pozycje, wynik, stan bloków) do pliku tekstowego.

- bool **loadFromFile** (const std::string &filename)
  Odczytuje dane z pliku tekstowego i zapisuje je w wewnętrznych strukturach.

- void **apply** (**Paletka** &p, **Pilka** &b, std::vector< **Stone** > &stones, sf::Vector2f blockSize, **wynik** &score)
  Przypisuje wczytane dane z powrotem do żywych obiektów gry, aktualizując ich stan.

Dokumentacja dla tej klasy została wygenerowana z plików:

GameState.h

GameState.cpp

Dokumentacja klasy Menu

Klasa obsługująca wyświetlanie menu głównego, nawigację klawiaturą oraz wybór trybów gry.

tc \l

```
#include <Menu.h>
```

Metody publiczne

- **Menu** (float width, float height)
  Konstruktor. Inicjalizuje czcionkę i tworzy napisy opcji menu, rozmieszczając je na ekranie.

- void **przesunG** ()
  Przesuwa zaznaczenie w menu o jedną pozycję w górę.

- void **przesunD** ()
  Przesuwa zaznaczenie w menu o jedną pozycję w dół.

- int **getSelectedItem** ()
  Zwraca indeks aktualnie wybranej opcji menu.

- void **draw** (sf::RenderWindow &window)
  Rysuje elementy menu w podanym oknie.

Dokumentacja dla tej klasy została wygenerowana z plików:

Menu.h

Menu.cpp

Dokumentacja klasy Paletka

Reprezentuje paletkę sterowaną przez gracza, obsługując jej ruch poziomy i ograniczenia ekranowe.
tc \l

```
#include <Paletka.h>
```

Metody publiczne

- **Paletka** (float x_in, float y_in, float szerokosc_in, float wysokosc_in, float predkosc_in)
  Konstruktor ustawiający wymiary, pozycję startową i prędkość paletki.

- void **moveLeft** ()
  Przesuwa paletkę w lewo o ustaloną prędkość.

- void **moveRight** ()
  Przesuwa paletkę w prawo o ustaloną prędkość.

- void **clmapToBounds** (float width)
  Sprawdza i koryguje pozycję paletki, aby nie wyjechała poza szerokość ekranu.

- void **draw** (sf::RenderTarget &target)
  Rysuje paletkę na ekranie.

- float **getX** () const
  Zwraca aktualną współrzędną X środka paletki.

- float **getY** () const
  Zwraca aktualną współrzędną Y środka paletki.

- float **getSzerokosc** () const
  Zwraca szerokość paletki.

- float **getWysokosc** () const
  Zwraca wysokość paletki.

- void **setPosition** (float setX, float setY)
  Ustawia paletkę bezpośrednio na podanych współrzędnych (używane przy wczytywaniu gry).

Dokumentacja dla tej klasy została wygenerowana z plików:

Paletka.h

Paletka.cpp

Dokumentacja klasy Pilka

Odpowiada za fizykę piłki, jej ruch, odbicia oraz wykrywanie kolizji z paletką, blokami i ścianami.
tc \l

```
#include <Pilka.h>
```

Metody publiczne

- **Pilka** (float x_in, float y_in, float vx_in, float vy_in, float radius_in)
  Konstruktor inicjalizujący pozycję, prędkość i promień piłki.

- void **move** ()
  Aktualizuje pozycję piłki, dodając do współrzędnych wektor prędkości.

- void **bounceX** ()
  Zmienia znak składowej poziomej prędkości (odbicie od pionowej przeszkody).

- void **bounceY** ()
  Zmienia znak składowej pionowej prędkości (odbicie od poziomej przeszkody).

- void **collideWalls** (float width, float height)
  Wykrywa kolizje z krawędziami ekranu i wywołuje odpowiednie odbicia.

- bool **collidePaddle** (const **Paletka** &p)
  Sprawdza kolizję z paletką i zmienia znnak pionowej prędkości (odbicie od poziomej przeszkody).

- void **collideStone** (**Stone** &s)
  Sprawdza kolizję z blokiem, niszczy go lub zmniejsza punkty życia i odbija piłkę.

- void **draw** (sf::RenderTarget &target)
  Rysuje piłkę na ekranie.

- float **getX** () const
  Zwraca współrzędną X piłki.

- float **getY** () const
  Zwraca współrzędną Y piłki.

- float **getVX** () const
  Zwraca prędkość poziomą piłki.

- float **getVY** () const
  Zwraca prędkość pionową piłki.

- float **getRadius** () const
  Zwraca promień piłki.

- void **setPilka** (float setX, float setY, float setVX, float setVY)
  Ustawia pozycję i prędkość piłki (używane przy wczytywaniu).

Dokumentacja dla tej klasy została wygenerowana z plików:

Pilka.h

pilka.cpp

Dokumentacja klasy Stone

Reprezentuje pojedynczy, zniszczalny blok na planszy, posiadający określoną wytrzymałość i kolor.
tc \l

```
#include <Stone.h>
```

Dziedziczy sf::RectangleShape.

Metody publiczne

- **Stone** (sf::Vector2f startPos, sf::Vector2f rozmiar, int L)
  Konstruktor tworzący blok w danej pozycji, o danym rozmiarze i liczbie życia L.

- void **trafienie** ()
  Zmniejsza życie bloku o 1. Jeśli życie spadnie do 0, blok uznaje się za zniszczony.

- void **aktualizujKolor** ()
  Aktualizuje kolor bloku na podstawie aktualnego poziomu życia (korzystając z tablicy kolorów).

- bool **isDestroyed** () const
  Zwraca true, jeśli blok został zniszczony.

- int **getHP** () const
  Zwraca aktualną liczbę punktów życia bloku.

Dokumentacja dla tej klasy została wygenerowana z plików:

Stone.h

Stone.cpp

Dokumentacja klasy wynik

Zarządza zliczaniem punktów zdobytych przez gracza oraz ich wyświetlaniem w interfejsie graficznym.
tc \l

```
#include <Wynik.h>
```

Metody publiczne

- **wynik** ()
  Konstruktor. Ładuje czcionkę i inicjalizuje licznik punktów na 0.

- void **wynik_update** ()
  Zwiększa aktualny wynik o 1 i aktualizuje wyświetlany tekst.

- void **draw** (sf::RenderTarget &target)
  Wyświetla tekst wyniku na ekranie.

- int **GetScore** () const
  Zwraca aktualną liczbę punktów jako liczbę całkowitą.

- void **SetScore** (int wynik_in)
  Ustawia wynik na konkretną wartość (używane po wczytaniu gry).

Dokumentacja dla tej klasy została wygenerowana z plików:

Wynik.h
Wynik.cpp

Dokumentacja plikówtc

Dokumentacja pliku Game.cpp

tc \l
```
#include "Game.h"
#include <iostream>
```

Game.cpp

Idź do dokumentacji tego pliku.

```
1 #include "Game.h"
2 #include <iostream>
3
4 Game::Game()
5     : m_paletka(320.f, 440.f, 100.f, 20.f, 8.f)
6     , m_pilka(320.f, 300.f, 4.f, 3.f, 8.f)
7     , m_score()
8     , ROZMIAR_BLOKU_X((WIDTH - (ILOSC_KOLUMN - 1) * 2.f) / ILOSC_KOLUMN)
9 {
10     float posX = 2.f;
11     float posY = 50.f;
12     int L;
13
14     for (int i = 0; i < ILOSC_WIERSZY; i++)
15     {
16         for (int j = 0; j < ILOSC_KOLUMN; j++)
17         {
18             L = (i < 1) ? 3 : (i < 3) ? 2 : 1;
19             m_bloki.emplace_back(sf::Vector2f({posX, posY}), sf::Vector2f({ROZMIAR_BLOKU_X,
ROZMIAR_BLOKU_Y}), L);
20             posX = 2.f + posX + ROZMIAR_BLOKU_X;
21         }
22
23         posY = 2.f + posY + ROZMIAR_BLOKU_Y;
24         posX = 2.f;
25     }
26 }
27
28 void Game::update(sf::Time dt)
29 {
30     if (m_gameOver) return;
31
32     m_pilka.move();
33     m_pilka.collideWalls(WIDTH, HEIGHT);
34
35     for (int k = 0; k < m_bloki.size(); k++)
36     {
37         m_pilka.collideStone(m_bloki[k]);
38
39         if (m_bloki[k].isDestroyed() == true)
40         {
41             m_bloki.erase(m_bloki.begin() + k);
42             m_score.wynik_update();
43         }
44
45     }
46
47     if (sf::Keyboard::isKeyPressed(sf::Keyboard::Key::Left)) {
48         m_paletka.moveLeft();
49     }
50     if (sf::Keyboard::isKeyPressed(sf::Keyboard::Key::Right)) {
51         m_paletka.moveRight();
52     }
53     m_paletka.clmapToBounds(WIDTH);
54
55     if (m_pilka.collidePaddle(m_paletka))
56     {
57         std::cout << "HIT PADDLE\n";
58     }
59
60     if (m_pilka.getY() - m_pilka.getRadius() > HEIGHT)
61     {
62         std::cout << "MISS! KONIEC GRY\n";
63         m_gameOver = true;
64     }
```

```
65
66      if (m_deltaClock.getElapsedTime().asSeconds() >= 2)
67      {
68          std::cout << "x=" << m_pilka.getX() << "\t y=" << m_pilka.getY()
69                  << "\t vx=" << m_pilka.getVX() << "\t vy=" << m_pilka.getVY() << std::endl;
70          m_deltaClock.restart();
71      }
72  }
73
74  void Game::render(sf::RenderTarget& target)
75  {
76      for (int l = 0; l < m_bloki.size(); l++)
77      {
78          target.draw(m_bloki[l]);
79      }
80
81      m_paletka.draw(target);
82      m_pilka.draw(target);
83      m_score.draw(target);
84  }
85
86  void Game::resetGame()
87  {
88      m_gameOver = false;
89      m_pilka = Pilka(320.f, 300.f, 4.f, 3.f, 8.f);
90      m_paletka = Paletka(320.f, 440.f, 100.f, 20.f, 8.f);
91      m_score.SetScore(0);
92
93
94      m_bloki.clear();
95
96      float posX = 2.f;
97      float posY = 50.f;
98      int L;
99
100      for (int i = 0; i < ILOSC_WIERSZY; i++)
101      {
102          for (int j = 0; j < ILOSC_KOLUMN; j++)
103          {
104              L = (i < 1) ? 3 : (i < 3) ? 2 : 1;
105              m_bloki.emplace_back(sf::Vector2f({posX, posY}),
sf::Vector2f({ROZMIAR_BLOKU_X, ROZMIAR_BLOKU_Y}), L);
106              posX = 2.f + posX + ROZMIAR_BLOKU_X;
107          }
108
109          posY = 2.f + posY + ROZMIAR_BLOKU_Y;
110          posX = 2.f;
111      }
112  }
113
114  void Game::saveGame(const std::string& filename)
115  {
116      GameState gs;
117      gs.capture(m_paletka, m_pilka, m_bloki, m_score);
118      if(gs.saveToFile(filename)) {
119          std::cout << "Gra zapisana!" << std::endl;
120      } else {
121          std::cout << "Blad zapisu gry!" << std::endl;
122      }
123  }
124
125  bool Game::loadGame(const std::string& filename)
126  {
127      GameState gs;
128      if (gs.loadFromFile(filename)) {
129          gs.apply(m_paletka, m_pilka, m_bloki, sf::Vector2f(ROZMIAR_BLOKU_X,
ROZMIAR_BLOKU_Y), m_score);
130          m_gameOver = false;
131          return true;
132      }
133      return false;
```

```
134 }
```

Dokumentacja pliku Game.h

tc \l#include <SFML/Graphics.hpp>
#include "Pilka.cpp"
#include "Paletka.cpp"
#include "Stone.cpp"
#include "GameState.cpp"
#include "Wynik.cpp"
#include <vector>

Komponenty

class **Game**

Game.h

Idź do dokumentacji tego pliku.

```
1 #ifndef GAME_H
2 #define GAME_H
3
4 #include <SFML/Graphics.hpp>
5 #include "Pilka.cpp"
6 #include "Paletka.cpp"
7 #include "Stone.cpp"
8 #include "GameState.cpp"
9 #include "Wynik.cpp"
10
11 #include <vector>
12
13 class Game
14 {
15 private:
16     Paletka m_paletka;
17     Pilka m_pilka;
18     wynik m_score;
19     std::vector<Stone> m_bloki;
20     sf::Clock m_deltaClock;
21
22
23     const float WIDTH = 640.f;
24     const float HEIGHT = 480.f;
25     const int ILOSC_KOLUMN = 6;
26     const int ILOSC_WIERSZY = 7;
27     const float ROZMIAR_BLOKU_Y = 25.f;
28     float ROZMIAR_BLOKU_X;
29
30 public:
31     Game();
32     void update(sf::Time dt);
33     void render(sf::RenderTarget& target);
34
35     bool isGameOver() const { return m_gameOver; }
36     void resetGame();
37
38     void saveGame(const std::string& filename);
39     bool loadGame(const std::string& filename);
40
41 private:
42     bool m_gameOver = false;
43 };
44
45 #endif
```

Dokumentacja pliku GameState.cpp

tc \l#include "GameState.h"

GameState.cpp

Idź do dokumentacji tego pliku.

```
1 #include "GameState.h"
2
3
4 void GameState::capture(const Paletka& p, const Pilka& b, const std::vector<Stone>& stones,
const wynik& currentscore) {
5    paddlePosition = {p.getX(), p.getY()};
6    ballPosition = {b.getX(), b.getY()};
7    ballVelocity = {b.getVX(), b.getVY()};
8    saved_score = currentscore.GetScore();
9    blocks.clear();
10    for (const auto& stone : stones)
11    {
12       if (!stone.isDestroyed())
13       {
14          blocks.push_back({stone.getPosition().x, stone.getPosition().y,
stone.getHP()});
15       }
16    }
17 }
18
19 bool GameState::saveToFile(const std::string& filename) {
20    std::ofstream file(filename);
21    if (!file.is_open()) return false;
22    file << "SCORE " << saved_score << "\n";
23    file << "PADDLE " << paddlePosition.x << " " << paddlePosition.y << "\n";
24    file << "BALL " << ballPosition.x << " " << ballPosition.y << " " << ballVelocity.x <<
" " << ballVelocity.y << "\n";
25    file << "BLOCKS_COUNT " << blocks.size() << "\n";
26    for (const auto& block : blocks) {
27       file << block.x << " " << block.y << " " << block.hp << "\n";
28    }
29    file.close();
30    return true;
31 }
32
33 bool GameState::loadFromFile(const std::string& filename) {
34    std::ifstream file(filename);
35    if (!file.is_open()) return false;
36    std::string label;
37
38    if (!(file >> label >> saved_score)) return false;
39    if (!(file >> label >> paddlePosition.x >> paddlePosition.y)) return false;
40
41    if (!(file >> label >> ballPosition.x >> ballPosition.y >> ballVelocity.x >>
ballVelocity.y)) return false;
42    int blocksCount;
43    if (!(file >> label >> blocksCount)) return false;
44
45    blocks.clear();
46    for (int i = 0; i < blocksCount; ++i) {
47       float x, y;
48       int hp;
49       file >> x >> y >> hp;
50       blocks.push_back({x, y, hp});
51    }
52
53    file.close();
54    return true;
55 }
56
57 void GameState::apply(Paletka& p, Pilka& b, std::vector<Stone>& stones, sf::Vector2f
blockSize, wynik& score) {
58    p.setPosition(paddlePosition.x, paddlePosition.y);
59
60    b.setPilka(ballPosition.x, ballPosition.y, ballVelocity.x, ballVelocity.y);
```

```
61
62    score.SetScore(saved_score);
63    stones.clear();
64    for (const auto& data : blocks)
65    {
66        stones.emplace_back(sf::Vector2f(data.x, data.y), blockSize, data.hp);
67    }
68 }
```

Dokumentacja pliku GameState.h

tc \l#include <SFML/Graphics.hpp>
#include <vector>
#include <fstream>
#include <string>
#include <iostream>
#include "Paletka.h"
#include "Pilka.h"
#include "Stone.h"
#include "Wynik.h"

Komponenty

- struct **BlockData**class **GameState**

GameState.h

Idź do dokumentacji tego pliku.

```
1 #ifndef GAMESTATE_H
2 #define GAMESTATE_H
3
4 #include <SFML/Graphics.hpp>
5 #include <vector>
6 #include <fstream>
7 #include <string>
8 #include <iostream>
9 #include "Paletka.h"
10 #include "Pilka.h"
11 #include "Stone.h"
12 #include "Wynik.h"
13
14
15 struct BlockData {
16    float x, y;
17    int hp;
18 };
19
20 class GameState {
21 private:
22    int saved_score;
23    sf::Vector2f paddlePosition;
24    sf::Vector2f ballPosition;
25    sf::Vector2f ballVelocity;
26    std::vector<BlockData> blocks;
27
28 public:
29    void capture(const Paletka& p, const Pilka& b, const std::vector<Stone>& stones, const
wynik& currentscore);
30    bool saveToFile(const std::string& filename);
31    bool loadFromFile(const std::string& filename);
32    void apply(Paletka& p, Pilka& b, std::vector<Stone>& stones, sf::Vector2f blockSize,
wynik& score);
33 };
34
35 #endif
```

Dokumentacja pliku main.cpp

tc \l#include <SFML/Graphics.hpp>
#include <iostream>
#include "Menu.cpp"
#include "Game.cpp"

Wyliczenia

- enum class **StanGry** { **Menu**, **Playing**, **Scores**, **Exiting** }

Funkcje

- void **myDelay** (int opoznienie)
- int **main** ()

Dokumentacja typów wyliczanych

enum class StanGry [strong]

Wartości wyliczeń:

Menu	
Playing	
Scores	
Exiting	

Dokumentacja funkcji

int main ()

void myDelay (int opoznienie)

Dokumentacja pliku Menu.cpp

tc \l`#include "Menu.h"`
`#include <iostream>`

Menu.cpp

Idź do dokumentacji tego pliku.

```
1 #include "Menu.h"
2 #include <iostream>
3
4 Menu::Menu(float width, float height)
5 {
6    if (!font.openFromFile("arial.ttf"))
7    {
8       std::cout << "Nie mozna zaladowac czcionki!" << std::endl;
9       return;
10    }
11
12    sf::Text t(font);
13
14    t.setFillColor(sf::Color::Cyan);
15    t.setStyle(sf::Text::Bold);
16    t.setString("Nowa gra");
17    t.setPosition(sf::Vector2f(width / 3, height / (MAX_LICZBA_POZIOMOW + 1) * 1));
18    menu.push_back(t);
19
20    t.setFillColor(sf::Color::White);
21    t.setStyle(sf::Text::Regular);
22    t.setString("Wczytaj gre");
23    t.setPosition(sf::Vector2f(width / 3, height / (MAX_LICZBA_POZIOMOW + 1) * 2));
24    menu.push_back(t);
25
26    t.setFillColor(sf::Color::White);
27    t.setString("Wyjscie");
28    t.setPosition(sf::Vector2f(width / 3, height / (MAX_LICZBA_POZIOMOW + 1) * 3));
29    menu.push_back(t);
30 }
31
32 void Menu::draw(sf::RenderWindow &window)
33 {
34    for (int i = 0; i < MAX_LICZBA_POZIOMOW; i++)
35    {
36       window.draw(menu[i]);
37    }
38 }
39
40 void Menu::przesunG()
41 {
42    if (selectedItem >= 0 && selectedItem < MAX_LICZBA_POZIOMOW)
43    {
44       menu[selectedItem].setFillColor(sf::Color::White);
45       menu[selectedItem].setStyle(sf::Text::Regular);
46       selectedItem--;
47       if (selectedItem < 0)
48          selectedItem = MAX_LICZBA_POZIOMOW - 1;
49       menu[selectedItem].setFillColor(sf::Color::Cyan);
50       menu[selectedItem].setStyle(sf::Text::Bold);
51    }
52 }
53
54 void Menu::przesunD()
55 {
56    if (selectedItem >= 0 && selectedItem < MAX_LICZBA_POZIOMOW)
57    {
58       menu[selectedItem].setFillColor(sf::Color::White);
59       menu[selectedItem].setStyle(sf::Text::Regular);
60       selectedItem++;
61       if (selectedItem >= MAX_LICZBA_POZIOMOW)
62          selectedItem = 0;
63       menu[selectedItem].setFillColor(sf::Color::Cyan);
64       menu[selectedItem].setStyle(sf::Text::Bold);
65    }
```

```
66 }
```

Dokumentacja pliku Menu.h

tc \l#include <SFML/Graphics.hpp>
#include <vector>

Komponenty

class Menu Definicje

- #define **MAX_LICZBA_POZIOMOW** 3

Dokumentacja definicji

#define MAX_LICZBA_POZIOMOW 3

Menu.h

Idź do dokumentacji tego pliku.

```
1 #ifndef MENU_H
2 #define MENU_H
3
4 #include <SFML/Graphics.hpp>
5 #include <vector>
6
7 #define MAX_LICZBA_POZIOMOW 3
8
9 class Menu
10 {
11 private:
12    sf::Font font;
13    std::vector<sf::Text> menu;
14    int selectedItem = 0;
15
16 public:
17    Menu(float width, float height);
18    ~Menu() {};
19    void przesunG();
20    void przesunD();
21    int getSelectedItem() { return selectedItem; }
22    void draw(sf::RenderWindow &window);
23 };
24
25 #endif
```

Dokumentacja pliku Paletka.cpp

tc \l#include "Paletka.h"

Paletka.cpp

Idź do dokumentacji tego pliku.

```
1 #include "Paletka.h"
2
3 Paletka::Paletka(float x_in, float y_in, float szerokosc_in, float wysokosc_in, float
predkosc_in)
4 {
5    x = x_in;
6    y = y_in;
7    szerokosc = szerokosc_in;
8    wysokosc = wysokosc_in;
9    predkosc = predkosc_in;
10
11    shape.setSize({szerokosc,wysokosc});
12    shape.setOrigin({szerokosc/2,wysokosc/2});
13    shape.setPosition({x,y});
14    shape.setFillColor(sf::Color::White);
15 }
16
17 void Paletka::moveLeft()
18 {
19    x = x - predkosc;
20    shape.setPosition({x,y});
21 }
22
23 void Paletka::moveRight()
24 {
25    x = x + predkosc;
26    shape.setPosition({x,y});
27 }
28
29 void Paletka::clmapToBounds(float width)
30 {
31    if (x-szerokosc/2 < 0)
32    {
33       x= szerokosc/2;
34       shape.setPosition({x,y});
35    }
36
37    if (x+szerokosc/2 > width)
38    {
39       x= width-szerokosc/2;
40       shape.setPosition({x,y});
41    }
42 }
43
44 void Paletka::draw(sf::RenderTarget& target)
45 {
46    target.draw(shape);
47 }
48
49 void Paletka::setPosition(float setX, float setY)
50 {
51    x = setX;
52    y = setY;
53    shape.setPosition({x, y});
54 }
```

Dokumentacja pliku Paletka.h

tc \l`#include <SFML/Graphics.hpp>`

Komponenty

class **Paletka**

Paletka.h

Idź do dokumentacji tego pliku.

```
1 #ifndef PALETKA_H
2 #define PALETKA_H
3
4 #include <SFML/Graphics.hpp>
5
6 class Paletka
7 {
8 private:
9    float x;
10    float y;
11    float szerokosc;
12    float wysokosc;
13    float predkosc;
14    sf::RectangleShape shape;
15
16 public:
17    Paletka(float x_in, float y_in, float szerokosc_in, float wysokosc_in, float
predkosc_in);
18    void moveLeft();
19    void moveRight();
20    void clmapToBounds(float width);
21    void draw(sf::RenderTarget& target);
22
23    float getX() const {return x;}
24    float getY() const {return y;}
25    float getSzerokosc() const {return szerokosc;}
26    float getWysokosc() const {return wysokosc;}
27
28    void setPosition(float setX, float setY);
29 };
30
31 #endif
```

Dokumentacja pliku pilka.cpp

tc \l#include "Pilka.h"
#include <cmath>
#include <iostream>

pilka.cpp

Idź do dokumentacji tego pliku.

```
1 #include "Pilka.h"
2 #include <cmath>
3 #include <iostream>
4
5 Pilka::Pilka(float x_in, float y_in, float vx_in, float vy_in, float radius_in)
6 {
7    x = x_in;
8    y = y_in;
9    vx = vx_in;
10    vy = vy_in;
11    radius = radius_in;
12
13    shape.setRadius(radius);
14    shape.setPosition({x,y});
15    shape.setOrigin({radius,radius});
16    shape.setFillColor(sf::Color::Magenta);
17 }
18
19 void Pilka::move()
20 {
21    x += vx;
22    y += vy;
23    shape.setPosition({x,y});
24 }
25
26 void Pilka::bounceX()
27 {
28    vx = -vx;
29 }
30
31 void Pilka::bounceY()
32 {
33    vy = -vy;
34 }
35
36 void Pilka::collideWalls(float width, float height)
37 {
38    if (x-radius <= 0 || x+radius >= width)
39       {
40          bounceX();
41       }
42
43    if (y-radius <= 0)
44       {
45          bounceY();
46       }
47 }
48
49 bool Pilka::collidePaddle(const Paletka& p)
50 {
51
52    if (x>=p.getX()-p.getSzerokosc()/2 && x<=p.getX()+p.getSzerokosc()/2 &&
53    y+ radius>= p.getY()-p.getWysokosc()/2 && y-radius < p.getY()-p.getWysokosc()/2)
54       {
55          vy=-std::abs(vy);
56          y = p.getY()-p.getWysokosc()/2 - radius;
57          shape.setPosition({x,y});
58          return true;
59       }
60    return false;
61 }
62
63 void Pilka::collideStone(Stone& s)
64 {
65    if (s.isDestroyed()== true)
```

```
66      {
67          return;
68      }
69
70      if (x-radius <= s.getPosition().x + s.getSize().x && x+radius >= s.getPosition().x &&
y-radius <= s.getPosition().y+s.getSize().y
71       && y + radius >= s.getPosition().y )
72      {
73          s.trafienie();
74
75          // Priorytet: jeśli kolizja z boku, odbij w X, w przeciwnym razie w Y
76          if (x + radius >= s.getPosition().x && x < s.getPosition().x ||
77          x - radius <= s.getPosition().x + s.getSize().x && x > s.getPosition().x +
s.getSize().x) {
78              bounceX();
79          }
80          else if (y + radius >= s.getPosition().y && y < s.getPosition().y ||
81          y - radius <= s.getPosition().y + s.getSize().y && y > s.getPosition().y +
s.getSize().y) {
82              bounceY();
83          }
84      }
85  }
86
87  void Pilka::draw(sf::RenderTarget& target)
88  {
89      target.draw(shape);
90  }
91
92  void Pilka::setPilka(float setX, float setY, float setVX, float setVY)
93  {
94      x = setX;
95      y = setY;
96      vx = setVX;
97      vy = setVY;
98      shape.setPosition({x, y});
99  }
```

Dokumentacja pliku Pilka.h

tc \l#include <SFML/Graphics.hpp>
#include "Paletka.h"
#include "Stone.h"

Komponenty

class **Pilka**

Pilka.h

Idź do dokumentacji tego pliku.

```
1 #ifndef PILKA_H
2 #define PILKA_H
3
4 #include <SFML/Graphics.hpp>
5 #include "Paletka.h"
6 #include "Stone.h"
7
8 class Pilka
9 {
10 private:
11    float x;
12    float y;
13    float vx;
14    float vy;
15    float radius;
16    sf::CircleShape shape;
17
18 public:
19    Pilka(float x_in, float y_in, float vx_in, float vy_in, float radius_in);
20    void move();
21    void bounceX();
22    void bounceY();
23    void collideWalls(float width, float height);
24    bool collidePaddle(const Paletka& p);
25    void collideStone(Stone& s);
26    void draw(sf::RenderTarget& target);
27
28    float getX() const {return x;}
29    float getY() const {return y;}
30    float getVX() const {return vx;}
31    float getVY() const {return vy;}
32    float getRadius() const {return radius;}
33
34    void setPilka(float setX, float setY, float setVX, float setVY);
35 };
36
37 #endif
```

Dokumentacja pliku Stone.cpp

tc \l`#include "Stone.h"`

Stone.cpp

Idź do dokumentacji tego pliku.

```
1 #include "Stone.h"
2
3 const std::array<sf::Color, 4> Stone::m_colorLUT = {
4    sf::Color::Transparent,
5    sf::Color::Red,
6    sf::Color::Yellow,
7    sf::Color::Blue,
8 };
9
10 Stone::Stone(sf::Vector2f startPos, sf::Vector2f rozmiar, int L)
11 {
12    Stone::RectangleShape::setPosition(startPos);
13    Stone::RectangleShape::setSize(rozmiar);
14    m_punktyZycia = L;
15    m_jestZniszczony = false;
16    Stone::RectangleShape::setOutlineThickness(2.f);
17    aktualizujKolor();
18 }
19
20 void Stone::trafienie()
21 {
22    if (m_jestZniszczony == true)
23    {
24       return;
25    }
26
27    m_punktyZycia= m_punktyZycia - 1;
28
29    if (m_punktyZycia <= 0)
30    {
31       m_jestZniszczony = true;
32    }
33
34    aktualizujKolor();
35
36 }
37
38 void Stone::aktualizujKolor()
39 {
40    Stone::RectangleShape::setFillColor(m_colorLUT[m_punktyZycia]);
41    if (m_jestZniszczony == true)
42    {
43       Stone::RectangleShape::setOutlineColor(sf::Color::Transparent);
44    }
45
46 }
```

Dokumentacja pliku Stone.h

tc \l#include <SFML/Graphics.hpp>
#include <array>

Komponenty

class **Stone**

Stone.h

Idź do dokumentacji tego pliku.

```
1 #ifndef STONE_H
2 #define STONE_H
3
4 #include <SFML/Graphics.hpp>
5 #include <array>
6
7 class Stone : public sf::RectangleShape
8 {
9 private:
10    int m_punktyZycia;
11    bool m_jestZniszczony;
12    static const std::array<sf::Color, 4> m_colorLUT;
13
14 public:
15    Stone(sf::Vector2f startPos, sf::Vector2f rozmiar, int L);
16    void trafienie();
17    void aktualizujKolor();
18    bool isDestroyed() const{return m_jestZniszczony;}
19
20    int getHP() const { return m_punktyZycia; }
21 };
22
23 #endif
```

Dokumentacja pliku Wynik.cpp

tc \l#include "Wynik.h"

Wynik.cpp

Idź do dokumentacji tego pliku.

```
1 #include "Wynik.h"
2
3 wynik::wynik() : t(font)
4 {   if (!font.openFromFile("arial.ttf"))
5    {
6       std::cout << "Nie mozna zaladowac czcionki!" << std::endl;
7       return;
8    }
9    score = 0;
10
11    t.setFont(font);
12    t.setFillColor(sf::Color::White);
13    t.setStyle(sf::Text::Bold);
14    t.setPosition(sf::Vector2f(5.f, 5.f));
15    t.setString("Wynik: 0");
16
17 }
18
19 void wynik::wynik_update()
20 {
21    score ++;
22    std::string str_score = std::to_string(score);
23    t.setString("Wynik: " + str_score);
24
25
26 }
27
28 void wynik::draw(sf::RenderTarget& target)
29 {
30    target.draw(t);
31 }
32
33 void wynik::SetScore(int wynik_in)
34 {
35    score = wynik_in;
36    t.setString("Wynik: " + std::to_string(score));
37 }
```

Dokumentacja pliku Wynik.h

tc \l`#include <SFML/Graphics.hpp>`
`#include <vector>`
`#include <iostream>`
`#include <string>`

Komponenty

class **wynik**

Wynik.h

Idź do dokumentacji tego pliku.

```
1 #ifndef WYNIK_H
2 #define WYNIK_H
3
4 #include <SFML/Graphics.hpp>
5 #include <vector>
6 #include <iostream>
7 #include <string>
8
9
10 class wynik
11 {
12 private:
13    int score;
14    sf::Font font;
15    sf::Text t;
16 public:
17    wynik();
18    void wynik_update();
19    void draw(sf::RenderTarget& target);
20    int GetScore()const {return score;};
21    void SetScore (int wynik_in);
22 };
23
24
25
26 #endif
```