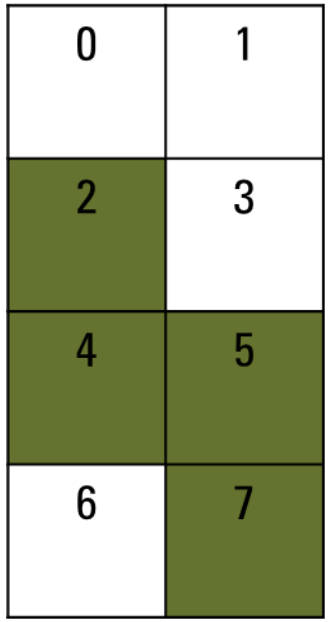
TETRIS

BIBLIOTEKI

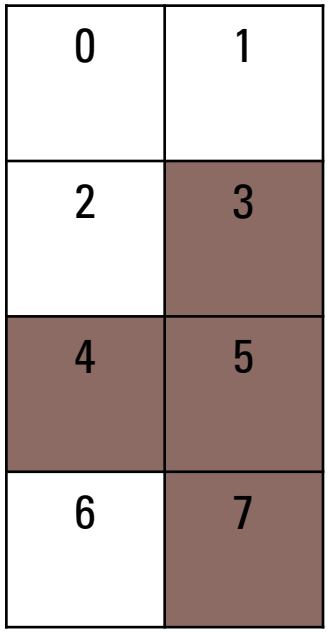
```
#include <SFML/Graphics.hpp>
#include <time.h>
#include <iostream>
#include "menu.h"
```

Rodzaje tetrimino

1



2



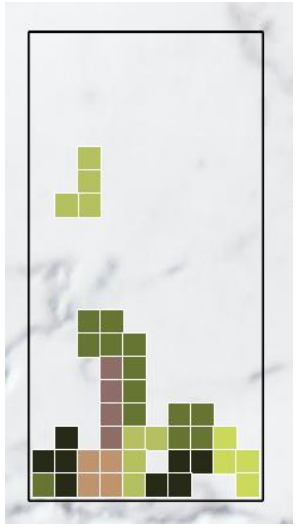

```
int figures[7][4] =
{
    1,3,5,7, // I
    2,4,5,7, // Z
    3,5,4,6, // S
    3,5,4,7, // T
    2,3,5,7, // L
    3,5,7,6, // J
    2,3,4,5, // O
};
```

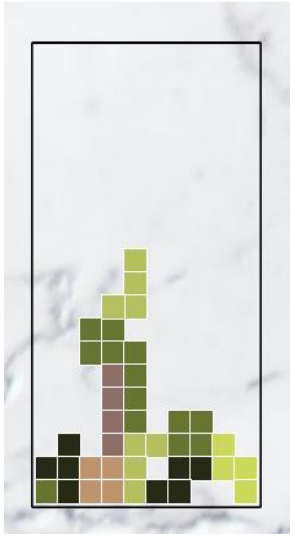
1

2

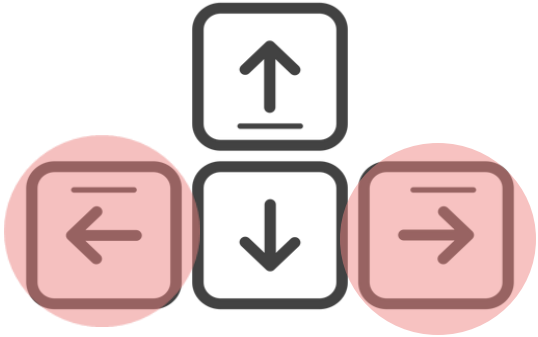
Sterowanie

1. Right , Left - kształt przesunięty o jedną współrzędną w prawo lub w lewo

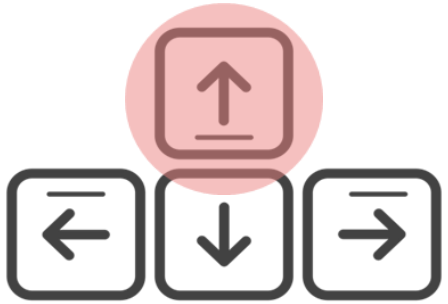




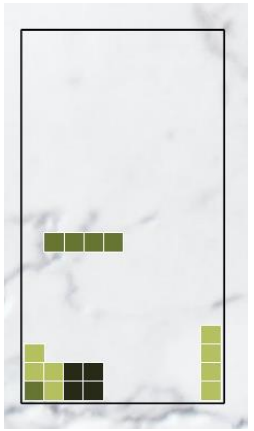
```
}
if (e.type == Event::Closed)
    window.close();
if (e.type == Event::KeyPressed)
    if (e.key.code == Keyboard::Escape) window.close();

if (e.type == Event::KeyPressed)
    if (e.key.code == Keyboard::Up) rotate = true;
    else if (e.key.code == Keyboard::Left) dx = -1;
    else if (e.key.code == Keyboard::Right) dx = 1;


(Keyboard::isKeyPressed(Keyboard::Down)) delay = 0.05;
```



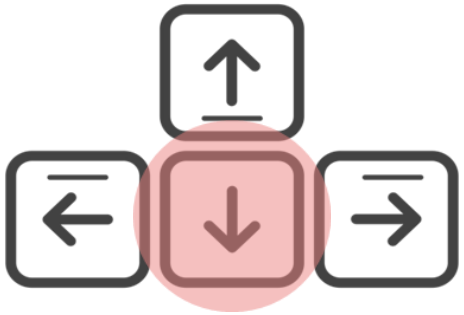
2. Up – obrót kształtu



```
if (rotate)
{
    Point p = a[1]; //srodek obrotu
    for (int i = 0; i < 4; i++)
    {
        int x = a[i].y - p.y; //musimy znać odległość od środka obrotu i później tę odległość przenosimy na współrzędne x albo y
        int y = a[i].x - p.x;
        a[i].x = p.x - x;
        a[i].y = p.y + y;
    }
    if (!check())
        for (int i = 0; i < 4; i++) //dlatego jeśli jest za blisko ścian to się nie wykonuje obrót
            a[i] = b[i];
}
```



2. Down – przyspieszanie



```
bool check()
{
    for (int i = 0; i < 4; i++)
        if (a[i].x < 0 || a[i].x >= N || a[i].y >= M) return 0;
        else if (field[a[i].y][a[i].x]) return 0;



    return 1;
};
```

Funkcja "check"

Sprawdza czy tetrimino nie nachodzą na siebie i czy nie wychodzą z planszy.

Menu - klasa

```cpp
#include "menu.h"



menu::menu(float szerokosc, float wysokosc)
{
    fontTitle.loadFromFile("cz/LexendDeca-Black.ttf");
    font.loadFromFile("cz/LexendDeca-Regular.ttf");

    m_title.setFont(fontTitle);
    m_title.setString("TETRIS");
    m_title.setCharacterSize(40);
    m_title.setFillColor(sf::Color(100, 115, 47, 250));
    m_title.setOutlineColor(sf::Color::White);
    m_title.setOutlineThickness(1);
    m_title.setPosition(sf::Vector2f(szerokosc / 6, wysokosc / 18));

    MENU_T[0].setFont(font);
    MENU_T[0].setFillColor(sf::Color(100, 115, 47, 250));
    MENU_T[0].setString("Play");
    MENU_T[0].setPosition(sf::Vector2f(szerokosc / 3, wysokosc / (MAX_NUMBER_OF_ITEMS + 1) * 1));


    MENU_T[1].setFont(font);
    MENU_T[1].setFillColor(sf::Color::White);
    MENU_T[1].setString("Exit");
    MENU_T[1].setPosition(sf::Vector2f(szerokosc / 3, wysokosc / (MAX_NUMBER_OF_ITEMS+1) * 2));

    Selected = 0;
}
menu::~menu()
```

```cpp
#pragma once
#include ...

#define MAX_NUMBER_OF_ITEMS 3

class menu
{
public:
    menu(float szerokosc, float wysokosc);
    ~menu();

    void draw(sf::RenderWindow &window);
    void MoveUp();
    void MoveDown();
    int Pressed() { return Selected; }


private:
    int Selected;
    sf::Font fontTitle;
    sf::Font font;
    sf::Text MENU_T[MAX_NUMBER_OF_ITEMS];
    sf::Text m_title;

};
```

Menu – metody klasy

- MoveUp,MoveDown- do zmiany kolorów przy poruszaniu się po menu

- Draw-do wyświetlenia menu w oknie

```
34      }
35      menu::~menu()
36      {
37
38      }
39      void menu::draw(sf::RenderWindow &window)
40      {
41          window.draw(m_title);
42
43          for (int i = 0; i < MAX_NUMBER_OF_ITEMS; i++)
44          {
45              window.draw(MENU_T[i]);
46          }
47      }
48      void menu::MoveUp()
49      {
50          if (Selected -1 >= 0)
51          {
52              MENU_T[Selected].setFillColor(sf::Color::White);
53              Selected--;
54              MENU_T[Selected].setFillColor(sf::Color(100, 115, 47, 250));
55          }
56      }
57      void menu::MoveDown()
58      {
59          if (Selected + 1 < MAX_NUMBER_OF_ITEMS)
60          {
61              MENU_T[Selected].setFillColor(sf::Color::White);
62              Selected++;
63              MENU_T[Selected].setFillColor(sf::Color(100, 115, 47, 250));
64          }
65      }
```

Switch do menu

- Tworzymy dwie zmienne bool menu, end
- Ta pętla w kodzie znajduje się w pętli gry, czyli while(window.isOpen()).

```
Event e;
while (window.pollEvent(e))
{

        if (e.type == sf::Event::KeyPressed)
        {
            if (e.key.code == sf::Keyboard::Up)
            {
                Menu.MoveUp();
            }
            if (e.key.code == sf::Keyboard::Down)
            {
                Menu.MoveDown();
            }
            if (e.key.code == sf::Keyboard::Enter)
            {
                switch (Menu.Pressed())
                {
                case 0:
                {
                    //gra
                    menu = false;

                    timer = 0;
                    cleanBoard();
                    score = 0;
                    end = false;
                    break;
                }

                case 2:
                {
                    window.close();
                    break;
                }
                }
            }
```

```
if (timer > delay) //wykonuje kiedy minie delay
{
    for (int i = 0; i < 4; i++)
    {

        b[i] = a[i]; a[i].y += 1;

    }
    timer = 0;
    if (!check())
    {
        for (int i = 0; i < 4; i++)field[b[i].y][b[i].x] = colorNum; //zapisanie do tablicy koloru klocka który jest definiowany niżej

        colorNum = 1 + rand() % 7; //generowanie koloru klocka

        int n = rand() % 7;   // losowanie figury
        for (int i = 0; i < 4; i++)
        {
            a[i].x = figures[n][i] % 2; //ustalam współrzędne nowej figury
            a[i].y = (int)figures[n][i] / 2;
        }

        if (!check())

        {
            end = true;
            timer = -10e10; //cofam sie w czasie

        }
    }
}
```

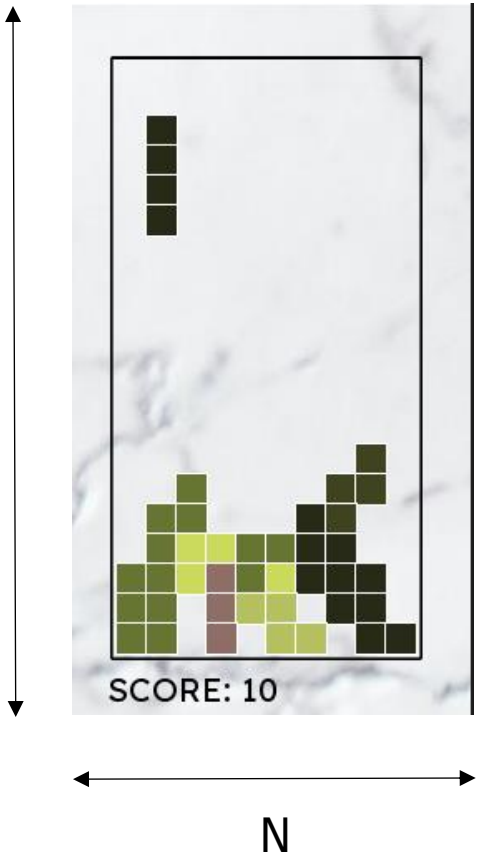
```
float time = clock.getElapsedTime().asSeconds();
clock.restart();
timer += time;
```

Logika gry

Liczenie punktów

```
int k = M - 1;//dlugosc pola, skad spada klocek
for (int i = M - 1; i > 0; i--)
{
    int count = 0;
    for (int j = 0; j < N; j++)
    {
        if (field[i][j]) count++;
        field[k][j] = field[i][j];
    }
    if (count < N)
    {
        k--;
    }
    else { score+=10; }
}
```

Jeśli cały wiersz nie jest zapełniony (count<N) idziemy dalej po pętli, a jeśli zapełniony dodajemy do wyniku 10 pkt i zamieniamy poprzednia linijkę na następna, zapełniona usuwamy.



Renderowanie sceny za pomoca SFML

- Ustalanie kolorów
- konturów
- rozmiarów
- miejsca wyświetlania w oknie rysowanych lub wklejanych kształtów.

```
RectangleShape rectangle;

for (int i = 0; i < M; i++)
    for (int j = 0; j < N; j++) //rysowanie klocków, które spadły
    {

        rectangle.setSize(Vector2f(18, 18));

        rectangle.setOutlineColor(Color::White);
        rectangle.setOutlineThickness(1);
        if (field[i][j] == 0) continue; //przezroczystość
        rectangle.setFillColor(colors[field[i][j]]);
        rectangle.setPosition(j * 18, i * 18);
        rectangle.move(28, 31);
        window.draw(rectangle);

    }

for (int i = 0; i < 4; i++)
{

    rectangle.setSize(Vector2f(18, 18));
    rectangle.setFillColor(colors[colorNum]);
    rectangle.setOutlineColor(Color::White);
    rectangle.setOutlineThickness(1);
    rectangle.setPosition(a[i].x * 18, a[i].y * 18);
    rectangle.move(28, 31);
    window.draw(rectangle);

}
```

Czyszczenie pola i gameover

Jeśli nasza zmienna bool end=true wyświetlamy komunikat gameover.

```
if (end)
{
    rectangle.setSize(Vector2f(260, 430));
    rectangle.setFillColor(Color(0, 0, 0, 200));
    rectangle.setPosition(0, 0);
    window.draw(rectangle);
    window.draw(gameover);
}
```

```
void cleanBoard()
{
    for (int i = 0; i < M; i++)
    {
        for (int j = 0; j < N; j++)
        {
            field[i][j] = 0;
        }
    }
}
```

Przed rozpoczęciem nowej gry musimy wyczyścić plansze, prostą funkcją void cleanBoard().