НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. І. СІКОРСЬКОГО»
ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

Кафедра системного програмування і спеціалізованих систем

Розрахунково-графічна робота
з дисципліни
«Програмування»

«Тетріс»

Виконав студент 1 курсу
ФПМ КВ-22
Міндер Вадим Юрійович
Оцінка:

Завдання

Створити ігрову програму мовою програмування С.

Розробка і реалізація ігрових програм має вестися з врахуванням графічних та звукових можливостей, що надаються конкретним комп'ютером.

Програма мусить коректно розв'язувати поставлену задачу. Логічно відокремлені частини алгоритма реалізувати за допомогою окремих функцій.

Також потрібно передбачити та забезпечити виконання всіх можливих розгалужень алгоритма, тобто програма повинна коректно реагувати на будь-які можливі ситуації (наприклад, виникнення помилкових ситуацій, перевірка файлів на порожність, правильність введених з клавіатури значень і т. д.). Передбачити взаємодію з користувачем (наприклад, можливість виводу правил гри, допомоги), таймер, лічильник числа ходів відповідно до поставленої в конкретному варіанті задачі.

Варіант 20:

«Тетріс». Випадкові фігурки падають згори в прямокутну форму шириною 10 і висотою 20 клітин. У польоті гравець повертати фігурку та рухати її по горизонталі. Також можна «скидати» фігурку, тобто прискорювати її падіння, коли вже вирішено, куди фігурка повинна впасти. Фігурка летить, поки не наткнеться на іншу фігурку або на дно форми. Якщо при цьому заповнився горизонтальний ряд з 10 кліток, він пропадає і все, що вище нього, опускається на одну клітку. Кожна фігура може складатися з 4 квадратів. Гра закінчується, коли нова фігурка не може поміститися в форму. Гравець отримує бали за кожну фігурку.

Текст програми

```
#include <SFML/Graphics.hpp>
#include <ctime>




class Button{
    sf::RectangleShape button;
    sf::Text text;
public:
    Button(){


    }
    Button(std::string str, sf::Color txtColor, sf::Font &font, sf::Vector2f size,
sf::Color bgColor, int fontSize) {
        text.setString(str);
        text.setColor(txtColor);
        text.setFont(font);
        text.setCharacterSize(fontSize);
        button.setSize(size);
        button.setFillColor(bgColor);
    }



    void setPositionBtn(sf::Vector2f pos){
        button.setPosition(pos);


        float xPos = (pos.x + (button.getLocalBounds().width -
text.getLocalBounds().width)/2.0);
```

```
        float yPos = (pos.y + (button.getLocalBounds().height -
text.getLocalBounds().height)/2.0) - 6;
        text.setPosition({xPos, yPos});
    }

    void drawTo(sf::RenderWindow &window){
        window.draw(button);
        window.draw(text);
    }

    bool isMouseOnBtn(sf::RenderWindow &window){
        float mouseX = sf::Mouse::getPosition(window).x;
        float mouseY = sf::Mouse::getPosition(window).y;
        float btnX1 = button.getPosition().x;
        float btnX2 = button.getPosition().x + button.getLocalBounds().width;
        float btnY1 = button.getPosition().y;
        float btnY2 = button.getPosition().y + button.getLocalBounds().height;

        if(mouseX < btnX2 && mouseX > btnX1 && mouseY < btnY2 && mouseY
> btnY1)
            return true;
        else return false;
    }



};

const int wWidth = 300;
const int wHeight = 600;
```

```
const int cellSize = 18;
const int M = 20;
const int N = 10;
const int fieldX = (wWidth - (N*cellSize))/2.0;
const int fieldY = 200;

int field[M][N] = { 0 };


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
};

struct Point
{
	int x, y;
} a[4], b[4];


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;
}

bool endGame()
{
        for (int i = 0; i < N; i++)
        {
                if (field[1][i])
                {
                        return true;
                }
        }
}

int main()
{
    srand(time(0));
    int score = 0;
    int dx = 0;
    bool rotate = false;
    int colorNum = 1;
    bool beginGame = true;
    bool gameOver = false;
    int n;
```

```
int scene = 0;









sf::RenderWindow window(sf::VideoMode(wWidth, wHeight), "Tetris");

auto icon = sf::Image{};
icon.loadFromFile("tetris.png");
window.setIcon(icon.getSize().x, icon.getSize().y, icon.getPixelsPtr());

sf::Texture texture;
texture.loadFromFile("tetris.png");

sf::Sprite imageTetris;
imageTetris.setTexture(texture);
imageTetris.setPosition({50,10});
imageTetris.setScale({0.2,0.2});

sf::Font font;
font.loadFromFile("font.ttf");

sf::RectangleShape rect({cellSize - 1,cellSize - 1});
rect.setOutlineThickness(1);
rect.setOutlineColor({0,0,0});
rect.setFillColor({255,0,0});
```

```
sf::RectangleShape rect3({cellSize- 1,cellSize - 1});
rect3.setOutlineThickness(1);
rect3.setOutlineColor({0,0,0});

sf::RectangleShape rect2({250,180});
rect2.setFillColor({255,200,200});
rect2.setPosition({25,10});

sf::Text textGameOver("GAME OVER!", font, 39);
textGameOver.setPosition({30,10});
textGameOver.setColor({0,0,0});

sf::Text textScore("SCORE: 0", font, 18);
textScore.setPosition({30,160});
textScore.setColor({0,0,0});


sf::RectangleShape fieldBorder({N*cellSize, M*cellSize});
fieldBorder.setPosition({fieldX, fieldY});
fieldBorder.setOutlineThickness(2);
fieldBorder.setOutlineColor({0,0,0});

Button btnStart("PLAY", {0,0,0},font,{200,50},{200,255,200},24);
Button btnRestart("RESTART", {0,0,0},font,{230,50},{200,255,200},24);
Button btnExit("EXIT", {0,0,0},font,{200,50},{255,200,200},24);
btnStart.setPositionBtn({50,250});
btnExit.setPositionBtn({50,320});
btnRestart.setPositionBtn({35, 100});
```

```
    sf::Color colorRect[10] =
{{255,0,0},{0,255,0},{0,0,255},{255,0,255},{255,255,0},{255,150,50},{100,255,
255}};


    float timer = 0;
    float delay = 0.3;


    clock_t clockStart;



    while (window.isOpen())
    {
        clockStart = clock();
        sf::Event event;
        while (window.pollEvent(event))
        {
            if (event.type == sf::Event::Resized){
                sf::FloatRect view(0, 0, event.size.width, event.size.height);
                window.setView(sf::View(view));
            }


            switch (event.type){
                case sf::Event::Closed:
                    window.close();
                break;
                case sf::Event::KeyPressed:
                    if(!gameOver && scene == 1){
                        switch (event.key.code){
                            case sf::Keyboard::Up:
```

```
                    rotate = true;
                break;
                case sf::Keyboard::Left:
                    dx = -1;
                break;
                case sf::Keyboard::Right:
                    dx = 1;
                break;
                case sf::Keyboard::Down:
                    delay = 0.05;
                break;

            }
        }
    break;
    case sf::Event::MouseButtonPressed:{
        if(btnStart.isMouseOnBtn(window)&& scene == 0) scene = 1;
        if(btnExit.isMouseOnBtn(window)&& scene == 0) window.close();
        if(btnRestart.isMouseOnBtn(window) && gameOver){
;           gameOver = false;
            score = 0;
            for(int i=0;i<M;i++)
                for (int j = 0; j < N; j++)
                {
                    field[i][j] = 0;
                }

        }
```

```
            }


        }

    }


if(!gameOver && scene == 1){
    for (int i = 0; i < 4; i++) {
        b[i] = a[i];
        a[i].x += dx;
    }

    if (!check()) {
        for (int i = 0; i < 4; i++)
        a[i] = b[i];
    }

    if (rotate)
    {
        Point p = a[1];
        for (int i = 0; i < 4; i++)
        {
            int x = a[i].y - p.y;
            int y = a[i].x - p.x;
            a[i].x = p.x - x;
```

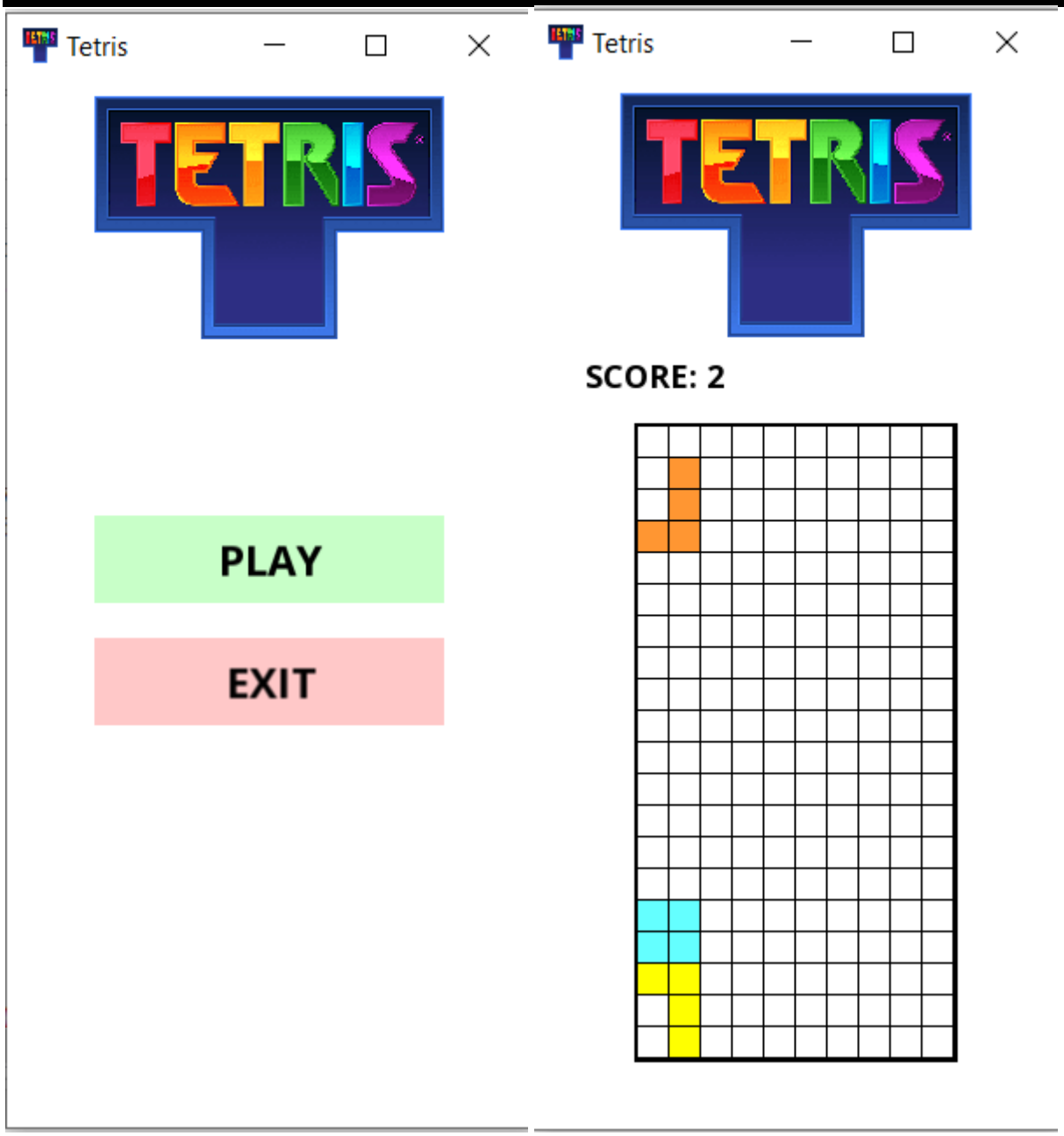
```
            a[i].y = p.y + y;
        }

        if (!check()) {
            for (int i = 0; i < 4; i++)
            a[i] = b[i];
        }
    }



            if (timer > delay)
    {
        for (int i = 0; i < 4; i++) { b[i] = a[i]; a[i].y += 1; }
        if (!check())
        {
            delay = 0.3;
            score++;
            textScore.setString("SCORE: "+std::to_string(score));
            for (int i = 0; i < 4; i++) field[b[i].y][b[i].x] = n + 1;
            n = rand() % 7;
            for (int i = 0; i < 4; i++)
            {
                a[i].x = figures[n][i] % 2;
                a[i].y = figures[n][i] / 2;
            }

        }
        timer = 0;
```

```
}

if (beginGame)
        {
                beginGame = false;
                n = rand() % 7;
                for (int i = 0; i < 4; i++)
                {
                        a[i].x = figures[n][i] % 2;
                        a[i].y = figures[n][i] / 2;
                }
        }

dx = 0;
rotate = 0;


int k = M - 1;
        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--;
        }
```

```
            if(endGame()){
        gameOver = true;
            }
}



            window.clear(sf::Color::White);
            window.draw(imageTetris);

            switch(scene){
               case 0:
                  btnStart.drawTo(window);
                  btnExit.drawTo(window);

        break;
        case 1:
           window.draw(fieldBorder);
           for(int i=0;i<M;i++)
              for (int j = 0; j < N; j++)
              {
                 rect3.setPosition(fieldX+j*cellSize,i*cellSize+fieldY);
                 window.draw(rect3);
              }


           for(int i=0;i<M;i++)
              for (int j = 0; j < N; j++)
              {
```

```
            if (field[i][j] == 0) continue;
            rect.setFillColor(colorRect[field[i][j]-1]);
            rect.setPosition(j * cellSize+ fieldX, i * cellSize+ fieldY);
            window.draw(rect);
        }

    for (int i = 0; i < 4; i++)
    {
        rect.setFillColor(colorRect[n]);
        rect.setPosition(a[i].x * cellSize + fieldX, a[i].y * cellSize+ fieldY);

        window.draw(rect);
    }
    if(gameOver){
        window.draw(rect2);
        window.draw(textGameOver);
        btnRestart.drawTo(window);
    }
    window.draw(textScore);

  break;
  case 2:

  break;



        }

window.display();
```

```
        timer += static_cast<float>(clock() - clockStart)/CLOCKS_PER_SEC;
    }

    return 0;
}
```



Tetris

GAME OVER!

RESTART

SCORE: 7