Государственное образовательное учреждение высшего профессионального образования

«Московский государственный технический университет имени Н.Э. Баумана»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Системы обработки информации и управления» (ИУ-5)

ДОМАШНЕЕ ЗАДАНИЕ

_____

_____

_____

_____

_____

_____

_____

Группа ИУ5-35Б

Студент _____ **16.12.2024 /А. А. Торопыгин/**

(Подпись, дата) (И.О.Фамилия)

Преподаватель _____ _____ **/Ю. Е. Гапанюк/**

(Подпись, дата) (И.О.Фамилия)

2024

Задание:

Разработать программу на языке Rust.

Реализовать классический вариант игры “2048”:

- Игра запускается в отдельном окне
- Матрица 4x4
- Плитки появляются, двигаются и складываются по обычным правилам
- Есть возможность отменить последнее действие
- Отображается изменяющийся счет
- Рекорд сохраняется при новом запуске
- Есть кнопка перезапуска

Текст программы:
(для работы fltk может быть нужно установить зависимости)

```rust
extern crate rand;

use std::fs::File;
use std::io::{Read, Write};
use button::Button;
use group::Grid;
use frame::Frame;
use app::App;
use window::Window;
use rand::{rngs::ThreadRng, Rng};
use enums::Color;
use fltk::{enums::{self, Font, Key}, prelude::*, *};

// Палитра для плиток
const COLORS: [Color; 12] = [
    Color::from_rgb(238, 228, 218),
    Color::from_rgb(237, 224, 200),
    Color::from_rgb(242, 177, 121),
    Color::from_rgb(245, 149, 99),
    Color::from_rgb(246, 124, 95),
    Color::from_rgb(246, 94, 59),
    Color::from_rgb(237, 207, 114),
    Color::from_rgb(237, 204, 97),
    Color::from_rgb(237, 200, 80),
    Color::from_rgb(237, 197, 63),
    Color::from_rgb(60, 58, 55),
    Color::from_rgb(62, 57, 51),
];


fn main() {
    let a = App::default(); // объект приложения

    let mut win = Window::default(); // объект окна
    win.set_pos(100, 100);
    win.set_size(1600, 900);
    win.set_label("2049");
    win.set_color(Color::White);

    let mut table = Grid::default(); // объект для отображения игрового поля
    table.set_pos(50, 50);
```

```
    table.set_size(800, 800);
    table.show_grid_with_color(true,Color::Black);
    table.set_layout(4, 4);
    table.set_color(Color::BackGround);
    for i in 0..16 {
        let _ = table.set_widget(&mut Frame::default()
        .with_size(200, 200), i/4, i%4);
    } // заполняем поле плитками
    table.end();

    let mut top_score = pull_top_score();
    let mut top_scoreboard = Frame::default(); // объект для отображения
рекорда
    top_scoreboard.set_pos(1100, 100);
    top_scoreboard.set_size(300, 100);
    top_scoreboard.set_label(format!("Top score:\n{}", top_score).as_str());
    top_scoreboard.set_label_size(32);
    top_scoreboard.set_label_font(Font::CourierBold);

    let mut scoreboard = Frame::default(); // объект для отображения счета
    scoreboard.set_pos(1100, 300);
    scoreboard.set_size(300, 100);
    scoreboard.set_label("Your score:\n0");
    scoreboard.set_label_size(32);
    scoreboard.set_label_font(Font::CourierBold);

    let mut rest_btn = Button::default(); //кнопка перезапуска
    rest_btn.set_pos(1150, 550);
    rest_btn.set_size(200, 200);
    rest_btn.set_label("@refresh");
    rest_btn.set_label_size(40);

    win.end();
    win.show();

    let mut _field = Field {     // создаем игровое поле
        table: table,
        matrix: [[0; 4]; 4],
        lst_matrix: [[0; 4]; 4],
        board: scoreboard,
        score: 0,
        lst_score: 0,
        random: rand::thread_rng()
```

```
    };
    _field.drop();
    _field.redraw();

    win.handle({    // обработка событий
        move |_win, ev| {
            match ev {
                enums::Event::KeyUp => {    // если была нажата кнопка
                    let key = app::event_key();
                    match key {     // определяем нажатую кнопку
                        Key::Up => _field.swipe(Direction::Up),
                        Key::Down => _field.swipe(Direction::Down),
                        Key::Right => _field.swipe(Direction::Right),
                        Key::Left => _field.swipe(Direction::Left),
                        Key::BackSpace => _field.revert(),
                        _ => ()
                    }
                    top_score = top_score.max(_field.score); // обновляем
рекорд
                    top_scoreboard.set_label(format!("Top score:\n{}",
top_score).as_str());
                    push_top_score(top_score);
                }
                enums::Event::Push => {     // если было нажатие курсора
                    let coords = app::event_coords(); // определяем
координату
                    if coords.0 >= 1150 && coords.0 <= 1350 &&
                       coords.1 >= 550 && coords.1 <= 750 {
                        _field.restart();
                    } // перезапуск если попали по кнопке
                }
                _ => {}
            }
            true
        }
    });

    a.run().unwrap();
}


// загрузка рекорда из файла
fn pull_top_score() -> i32 {
```

```
    let mut file = match File::open("top.txt") {
        Ok(f) => f,
        Err (_) => File::create("top.txt").unwrap()
    };
    let mut line = String::new();
    _ = file.read_to_string(&mut line);
    match line.parse() {
        Ok(value) => return value,
        Err(_) => return 0
    };
}


// сохранение рекорда в файл
fn push_top_score(score: i32) {
    let mut file = File::create("top.txt").unwrap();
    _ = file.write(score.to_string().as_bytes());
}


// объект игрового поля
struct Field {
    table: Grid,
    matrix: [[i32; 4]; 4],
    lst_matrix: [[i32; 4]; 4],
    board: Frame,
    score: i32,
    lst_score: i32,
    random: ThreadRng
}

// направления движения плиток
enum Direction {
    Up,
    Down,
    Right,
    Left
}

impl Field{
    // появление новой плитки в случайном незанятом месте
    fn drop(&mut self) {
        loop {
```

```
            let cell = self.random.gen_range(0..16);
            if self.matrix[cell/4][cell%4] == 0 {
                if self.random.gen_range(0..10) == 0{
                    self.matrix[cell/4][cell%4] = 4;
                }
                else {
                    self.matrix[cell/4][cell%4] = 2;
                }
                return;
            }
        }
    }

    // отрисовка изменений
    fn redraw(&mut self) {
        for i in 0..16 {
            let mut tile = self.table.child(i as i32).unwrap();
            if self.matrix[i/4][i%4] == 0 {
                tile.set_label("");
                tile.set_frame(enums::FrameType::NoBox);
            }
            else {
                tile.set_label(self.matrix[i/4][i%4].to_string().as_str());
                tile.set_label_font(Font::CourierBold);
                tile.set_label_size(64);
                let color_index = match self.matrix[i/4][i%4] {
                    0 => 0,
                    v => (v as f64).log2() as usize + 1,
                };
                tile.set_frame(enums::FrameType::RoundedBox);
                tile.set_color(COLORS[color_index.min(11)]);
            }
            let _ = self.table.set_widget(&mut tile.clone(), i/4, i%4);
        }

        self.board.set_label(format!("Your score:\n{}",
self.score).as_str());
    }

    // движение и сложение плиток в направлении
    pub fn swipe(&mut self, dir: Direction) {
        self.lst_score = self.score;
        for i in 0..16 {self.lst_matrix[i/4][i%4] = self.matrix[i/4][i%4];}
```

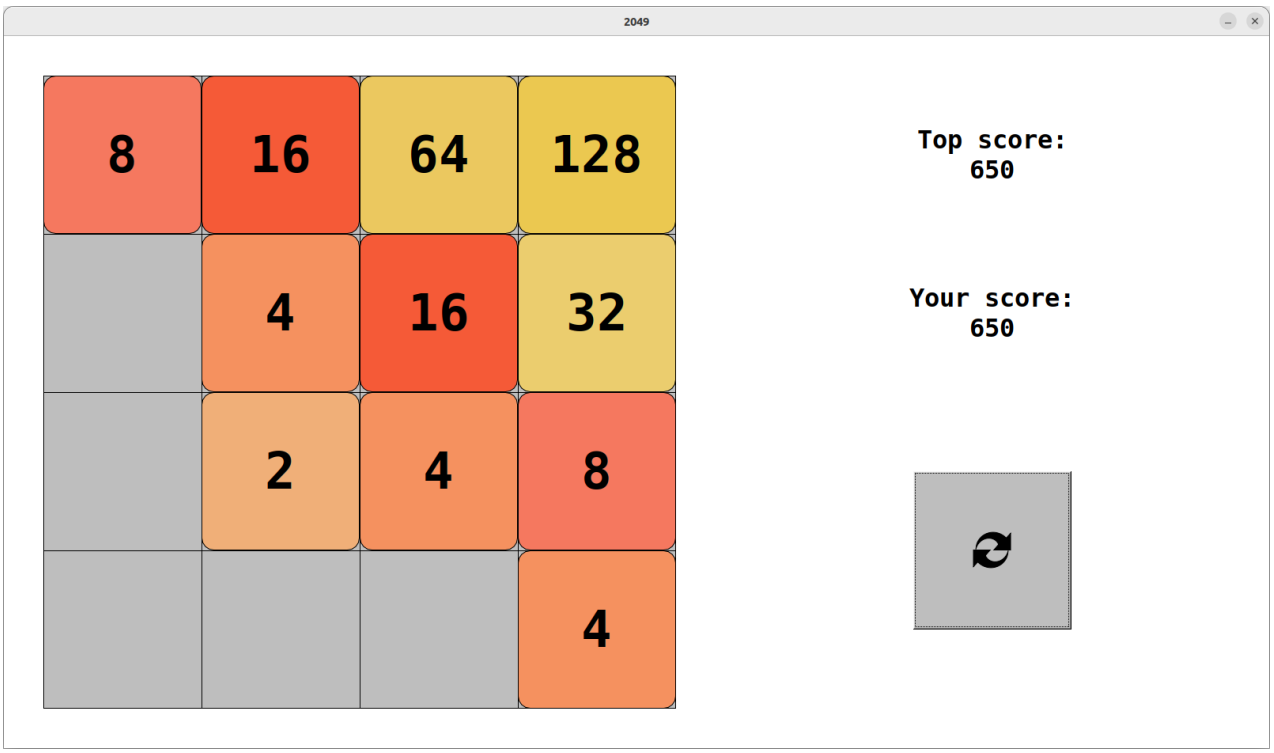
```
let mut is_changed = false;
for i in 0..4 {
    let mut temp = [0; 4];
    let mut ind = 0;
    for j in 0..4 {
        let val = match dir {
            Direction::Up => self.matrix[j][i],
            Direction::Down => self.matrix[3-j][i],
            Direction::Right => self.matrix[i][3-j],
            Direction::Left => self.matrix[i][j]
        };
        if val == 0 {continue;}
        if temp[ind] == 0 {
            temp[ind] = val;
        }
        else if temp[ind] == val {
            temp[ind] += val;
            self.score += val;
            ind += 1;
        }
        else {
            ind += 1;
            temp[ind] = val;
        }
    }
    for j in 0..4 {
        match dir {
            Direction::Up => {
                if self.matrix[j][i] != temp[j] {
                    self.matrix[j][i] = temp[j];
                    is_changed = true;
                }
            },
            Direction::Down => {
                if self.matrix[3-j][i] != temp[j] {
                    self.matrix[3-j][i] = temp[j];
                    is_changed = true;
                }
            },
            Direction::Right => {
                if self.matrix[i][3-j] != temp[j] {
                    self.matrix[i][3-j] = temp[j];
```

```
                            is_changed = true;
                        }
                    },
                    Direction::Left => {
                        if self.matrix[i][j] != temp[j] {
                            self.matrix[i][j] = temp[j];
                            is_changed = true;
                        }
                    }
                };
            }
        }
        if is_changed {
            self.drop();
            self.redraw();
        }
    }

    // отмена последнего действия
    pub fn revert(&mut self) {
        self.score = self.lst_score;
        for i in 0..16 {self.matrix[i/4][i%4] = self.lst_matrix[i/4][i%4];}
        self.redraw();
    }

    // перезапуск игры
    pub fn restart(&mut self) {
        self.matrix = [[0; 4]; 4];
        self.lst_matrix = [[0; 4]; 4];
        self.score = 0;
        self.lst_score = 0;
        self.drop();
        self.redraw();
    }
}
```

Результат:


2049
8
16
64
128
4
16
32
2
4
8
4
Top score:
650
Your score:
650