ФЕДЕРАЛЬНОЕ АГЕНТСТВО СВЯЗИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«СИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ТЕЛЕКОММУНИКАЦИЙ И ИНФОРМАТИКИ»

Кафедра вычислительных систем

КУРСОВАЯ РАБОТА

по дисциплине «Технологии разработки программного обеспечения»

на тему «KeyboardNinja»

Выполнил:
ст. гр. ИП-216
Русецкий А. С.

Проверил:
ст. преподаватель Токмашева Е. И.

Новосибирск, 28.05.2023

Содержание

Введение и постановка задачи

Передо мной и моей командной стояла цель: работая в команде, реализовать законченный продукт.

Задачи, которые были поставлены:

1. Придумать идею реализации данного проекта.
2. Написать рабочую программу.
3. Придумать дизайн программы.
4. Исправить баги и ошибки.
5. Покрытие тестами, включая unit тестирование.
6. Практика использования CI.
7. Разбить проект на модули
8. Полностью подготовить проект к релизу.
9. Итоговая презентация.

Задание на курсовую работу

В качестве темы для курсовой работы нами была выбрана игра «KeyboardNinja», правила игры предельно просты:

Клавиатурный тренажёр представляет собой игру, предназначенную для оценки правильного выбора нужной клавиши на клавиатуре. Во время игры, в игровой области падают буквы. Цель игрока: успеть нажать нужную клавишу с буквой на клавиатуре, пока буква не достигла конца игровой области.

Техническое задание

Цель проекта: Разработать интерактивную игру «KeyboardNinja».

Стадии разработки и задачи на этих стадиях:

1. Разработать дизайн интерфейса игры

- Создать игровое меню
- Определить цветовую схему и общий стиль игры

2. Реализовать игровую логику

- Сделать запуск игры по нажатию кнопки SPACE
- Реализовать падение букв в игровом окне
- Реализовать проверку на правильность нажатой клавиши на клавиатуре

3. Реализовать несколько уровней сложности

- Создать меню для выбора уровня сложности
- Изменить скорость падения буквы на игровом окне в зависимости от уровня сложности

4. Подсчёт и оценка результата

- Добавить счётчик правильно и неправильно нажатых клавиш
- Добавить таймер для игры
- Вывести результаты в отдельное окно после окончания игры

5. Тестирование и отладка

- Протестировать функциональность на всех уровнях сложности
- Отладить игру для устранения ошибок и багов

6. Релиз и поддержка

- Выпустить игру и предоставить ее преподавателю

Описание выполненного проекта

Отдельно опишем содержимое файлов программы.

Файл **difficult.cpp**:

В данном файле содержатся функции:

- «MoveUpDifficult», «MoveDownDifficult» - Переключение кнопок уровня сложности при помощи стрелок на клавиатуре.
- «buttonEasyCondition», «buttonNormalCondition», «buttonHardCondition» - Кнопки «Easy», «Normal», «Hard».

Файл **fileload.cpp**:

Данный файл содержит функции:

- «loadTextureFromFile», «loadFontFromFile» - Проверка загрузки текстур и шрифта.

Файл **game.cpp**:

В данном файле содержатся функции:

- «initFrame» - Отрисовка рамки
- «buttonBack» - Кнопка «Назад»
- «buttonRefresh» - Кнопки «Обновить»
- «startTimer» - Таймер игры
- «selectDifficult» - уровень сложности
- «startPositionCube», «gameKey» - Игровой процесс
- «checkCorrect» - Счётчик правильных и неправильных нажатий

Файл **menu.cpp**:

В данном файле содержатся функции:

- «initText» - Отрисовка текста в меню
- «initButton» - Отрисовка кнопки
- «MoveUp», «MoveDown» - Переключение между кнопками «Начать» и «Выйти» при помощи стрелок на клавиатуре
- «buttonStartCondition», «buttonExitCondition» - Кнопки «Начать» и «Выйти»

Файл **mode.cpp**:

В данном файле содержатся функции:

- «modeMenu» - Стартовое меню
- «modeDifficult» - Меню выбора уровня сложности
- «modeGame» - Игровое окно
- «modeResult» - Окно результатов

Файл **window.cpp**:

В данном файле содержатся функции:

- «windowMenu» - Вывод стартового меню
- «windowDifficult» - Вывод меню выбора уровня сложности
- «windowGame» - Вывод игрового окна
- «windowResult» - Вывод окна результатов

Тестирование проекта

Для данной программы было написано 10 тестов:

1. Проверка загрузки шрифта «`CTEST (ctest, loadFontFromFile)`»
2. Проверка загрузки заднего фона «`CTEST (ctest, loadTextureFromFile»`
3. Проверка переключения клавиши вверх «`CTEST (ctest, MoveUp)`»
4. Проверка переключения клавиши вниз «`CTEST (ctest, MoveDown)`»
5. Проверка переключения клавиши вверх в меню выбора уровня сложности

   «`CTEST (ctest, MoveUpDifficult)`»
6. Проверка переключения клавиши вниз в меню выбора уровня сложности

   «`CTEST (ctest, MoveDownDifficult)`»
7. Проверка выбора уровня сложности

   «`CTEST (ctest, selectDifficult)`»
8. Проверка игрового процесса «`CTEST (ctest, startPositionCube)`»
9. Проверка событий окна результата

   «`CTEST(renderwindow_test, mode_result_test»`
10. Проверка событий окна меню

    «`CTEST(renderwindow_test, mode_menu_test)»`

Результаты работы программы:

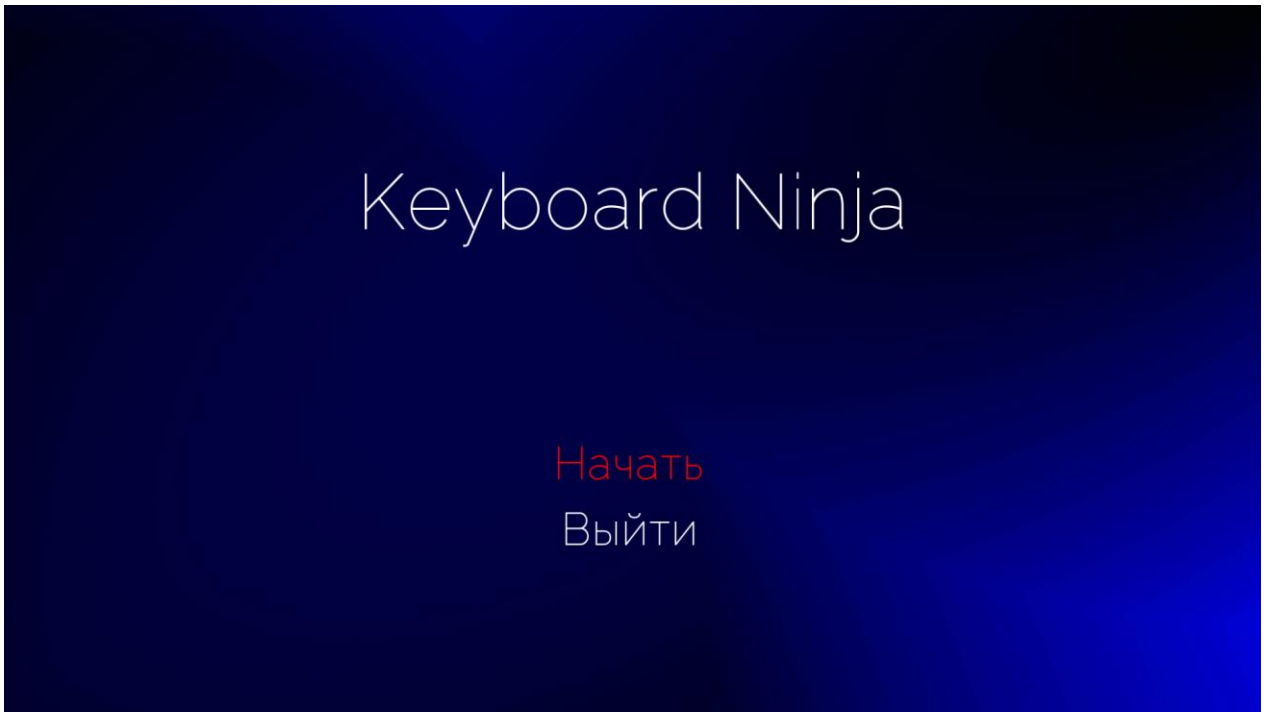

Рис. 1. Главное меню

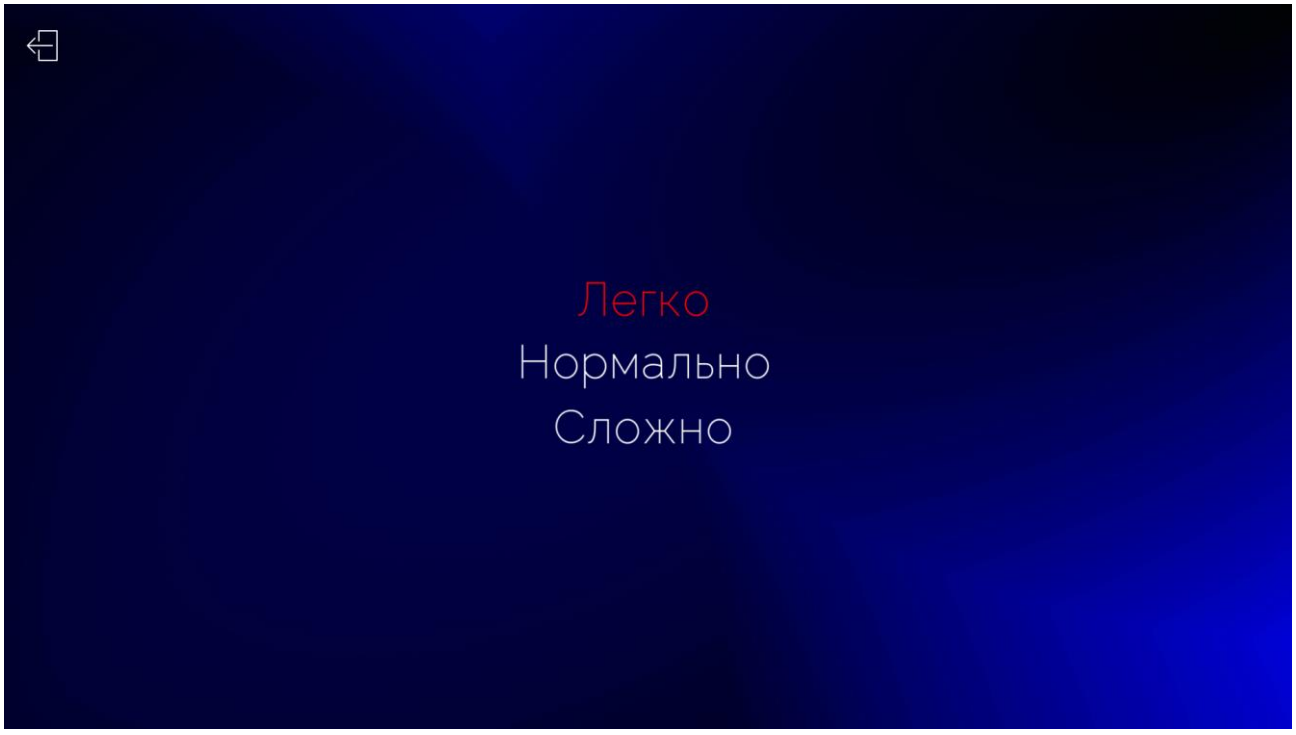

Рис. 2. Меню выбора уровня сложности

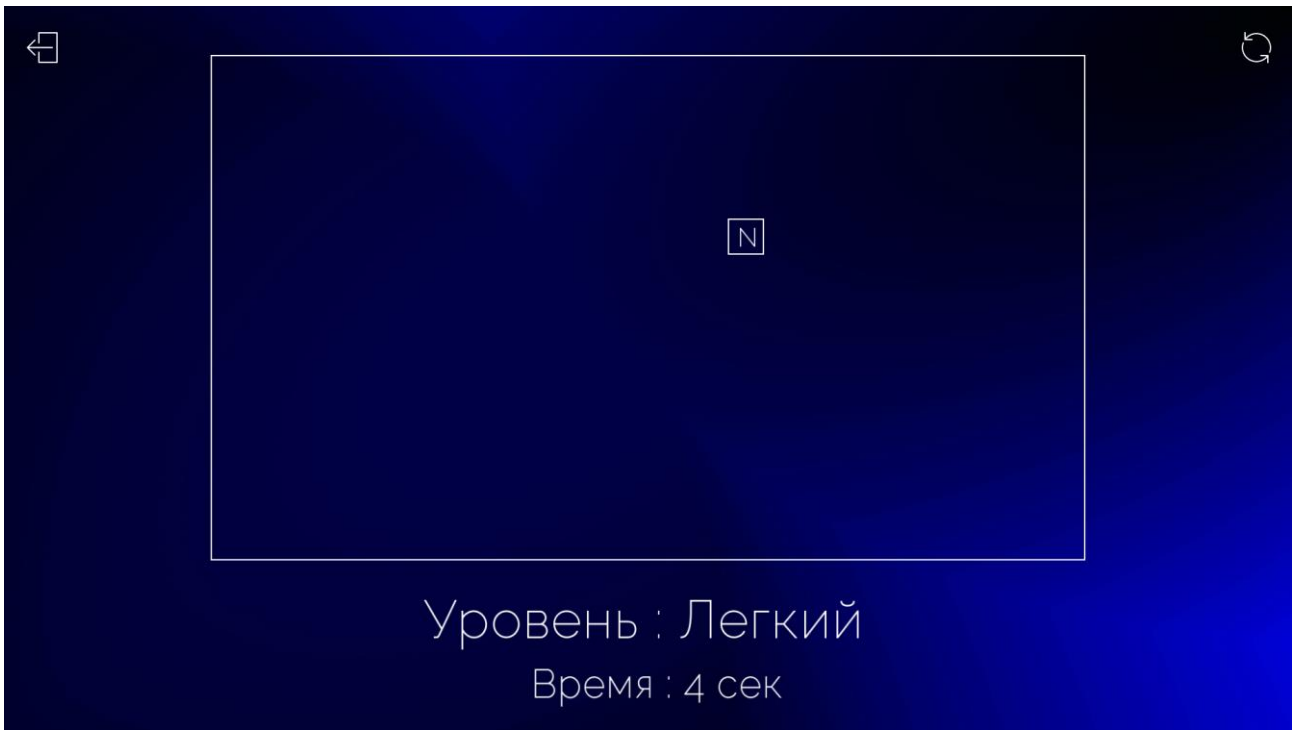

Рис. 3. Игровое меню и процесс

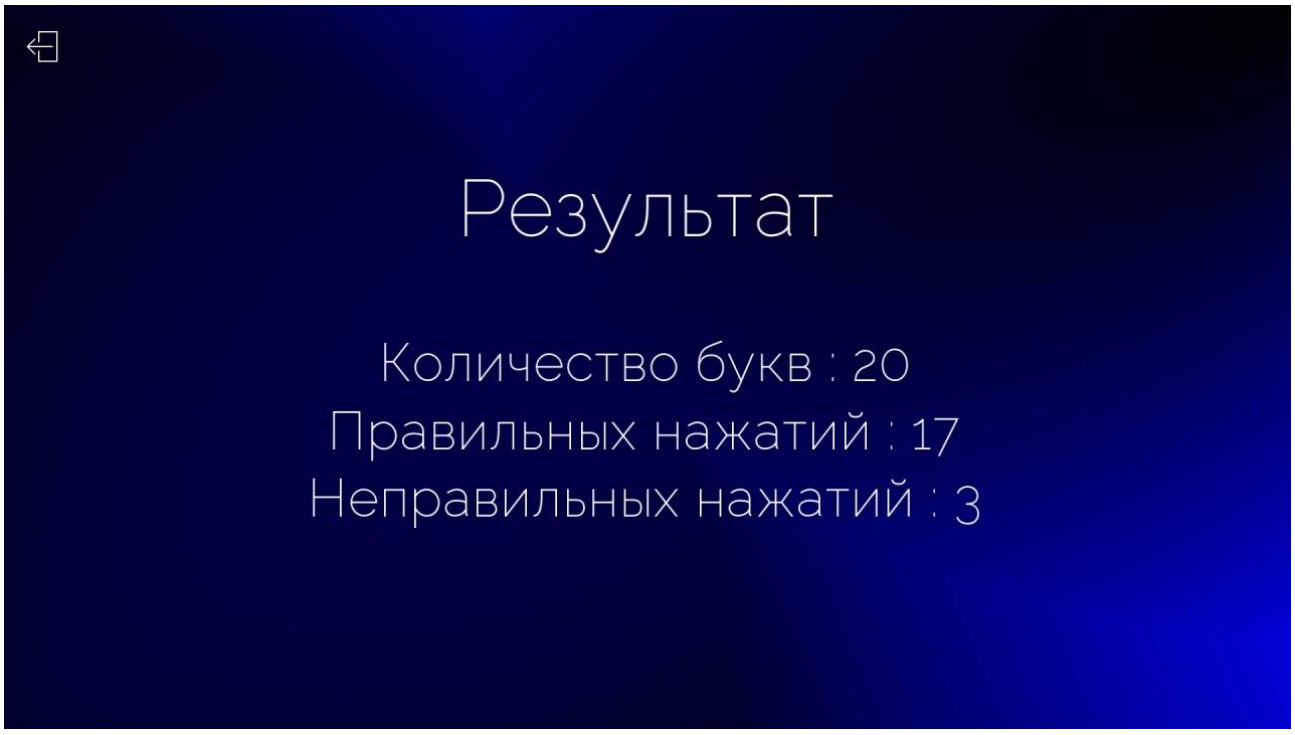

Рис. 4. Результаты

Личный вклад в проект

- Составление ТЗ и презентации продукта
- Разработка интерфейса
- Разработка логики и механики игры
- Участие в разработке игрового процесса
- Помощь в написании MakeFile
- Разбиение проекта на модули
- Частичное написание тестов

Приложение. Текст программы

main.cpp :

```
#include <iostream>
#include <difficult.h>
#include <game.h>
#include <menu.h>
#include <mode.h>
#include <window.h>
#include <fileLoad.h>

using namespace sf;

int main() {

    //Окно
    RenderWindow window;
    window.create(VideoMode::getDesktopMode(), "KeyBoardNinja",
Style::Fullscreen);
    window.setFramerateLimit(60);


////////////////////////////////////////////////////////////////////////////

    //Фон меню
    RectangleShape background(Vector2f(1920, 1080));
    Texture screen;
    loadTextureFromFile(screen, "Resource/Pictures/Background/3.jpg");
    background.setTexture(&screen);


////////////////////////////////////////////////////////////////////////////

    //Шрифт
    Font font;
    loadFontFromFile(font, "Resource/Font/Raleway/static/Raleway-Thin.ttf");


////////////////////////////////////////////////////////////////////////////

    //Заголовок
    Text title;
    initText(title, font, 120, L"Keyboard Ninja", 960, 300, Color::White);


////////////////////////////////////////////////////////////////////////////

    //Кнопки

    Text buttonMenu[2];
    initText(buttonMenu[0], font, 70, L"Начать", 960, 700, Color::Red);
    initText(buttonMenu[1], font, 70, L"Выйти", 960, 800, Color::White);

    int numberButton = 0;       //0 - Start; 1 - Exit

    Text buttonDifficult[3];
    initText(buttonDifficult[0], font, 70, L"Легко", 960, 440, Color::Red);
    initText(buttonDifficult[1], font, 70, L"Нормально", 960, 540,
Color::White);
```

```
    initText(buttonDifficult[2], font, 70, L"Сложно", 960, 630, Color::White);

    int numberDifficult = 0;    //0 - Easy; 1 - Normal; 2 - Hard

    RectangleShape logOutButton;    //Кнопка назад
    Texture logOutTexture;
    loadTextureFromFile(logOutTexture, "Resource/Pictures/Button/logout.png");
    initButton(logOutButton, 25, 30, logOutTexture);

    RectangleShape refreshButton;   //Кнопка перезапуска
    Texture refreshTexture;
    loadTextureFromFile(refreshTexture, "Resource/Pictures/Button/refresh.png");
    initButton(refreshButton, 1835, 30, refreshTexture);


////////////////////////////////////////////////////////////////////////////////

    //Текст
    Text typeGamelvl[3];
    initText(typeGamelvl[0], font, 80, L"Уровень : Легкий", 960, 920,
Color::White);
    initText(typeGamelvl[1], font, 80, L"Уровень : Нормальный", 960, 920,
Color::White);
    initText(typeGamelvl[2], font, 80, L"Уровень : Сложный", 960, 920,
Color::White);

    Text noticeMessage;
    String noticeMessage_str = L"Отсчет времени начнется после нажатия SPACE";
    initText(noticeMessage, font, 50, noticeMessage_str, 960, 1020,
Color::White);
    int flagStart = 0;          //0 - !Start; 1 - Start

    Text timeMessage;
    timeMessage.setFont(font);
    timeMessage.setStyle(Text::Bold);
    timeMessage.setCharacterSize(60);
    timeMessage.setFillColor(Color::White);


////////////////////////////////////////////////////////////////////////////////

    //Рамки
    RectangleShape board;

    initFrame(board, 1300, 750, 960, 450);



////////////////////////////////////////////////////////////////////////////////

    //Таймер

    Clock clock;
    int timer;


////////////////////////////////////////////////////////////////////////////////

    //Кубик

    RectangleShape cube;
```

```
    float xSize = 50, ySize = 50;
    float xPos = 935, yPos = 100;
    initFrame(cube, xSize, ySize, xPos, yPos);


////////////////////////////////////////////////////////////////////////////////

    //Буквы

    std::string letters[26];

    int z = 0;
    for (int i = 'A'; i <= 'Z'; i++) {
        letters[z] = i;
        z++;
    }

    Text letter;
    initText(letter, font, 40, "A", 935 + 10, 100 + 10, Color::White);

    int numberLetter;
    int flagCorrect = 0;


////////////////////////////////////////////////////////////////////////////////

    //Результат

    Text titleResult;
    initText(titleResult, font, 120, L"Результат", 960, 300, Color::White);
    Text textSumLetters;
    Text textCorrectTypes;
    Text textIncorrectTypes;

    int sumLetters = 0;
    int correctTypes = 0;
    int incorrectTypes = 0;



////////////////////////////////////////////////////////////////////////////////

    int checkMode = 0;          //0 - Menu; 1 - Difficult; 2 - Game; 4 - Result

    while (window.isOpen()) {
        Event ev;
        while (window.pollEvent(ev)) {
            modeMenu(window, ev, buttonMenu, numberButton, checkMode);
            modeDifficult(window, ev, buttonDifficult, logOutButton,
numberDificult, checkMode, sumLetters, correctTypes, incorrectTypes);
            modeGame(window, ev, logOutButton, refreshButton, clock, cube,
letter, checkMode,
            flagStart, sumLetters, numberLetter, correctTypes, flagCorrect,
incorrectTypes);
            modeResult(window, ev, logOutButton, checkMode, sumLetters,
correctTypes, incorrectTypes);
        }

        window.clear();
        window.draw(background);
```

```
        switch (checkMode) {
            case 0 :
                windowMenu(window, title, buttonMenu[0], buttonMenu[1]);
                break;
            case 1 :
                windowDifficult(window, logOutButton, buttonDifficult[0],
buttonDifficult[1], buttonDifficult[2]);
                break;
            case 2 :
                windowGame(window, board, typeGamelvl, noticeMessage,
timeMessage, logOutButton, refreshButton, clock, timer, cube, letter, letters,
flagStart, checkMode, numberDifficult, flagCorrect, correctTypes,
incorrectTypes, sumLetters, numberLetter);
                break;
            case 3:
                windowResult(window, font, titleResult, textSumLetters,
textCorrectTypes, textIncorrectTypes, logOutButton, sumLetters, correctTypes,
incorrectTypes);
                break;
        }

        window.display();
    }

    return 0;
}
```

difficult.h :

```
#pragma once

#include <iostream>
#include <SFML/Graphics.hpp>

using namespace sf;

bool MoveUpDifficult(Text button[], int& number);

bool MoveDownDifficult(Text button[], int& number);

void buttonEasyCondition(Text buttonDifficult[], int& numberDifficult, int&
checkMode);

void buttonNormalCondition(Text buttonDifficult[], int& numberDifficult, int&
checkMode);

void buttonHardCondition(Text buttonDifficult[], int& numberDifficult, int&
checkMode);
```

difficult.cpp :

```
#include <difficult.h>

bool MoveUpDifficult(Text button[], int& number) {
    if (number - 1 >= -1) {
        button[number].setFillColor(Color::White);
        number--;
        if (number == -1)
            number = 2;
        button[number].setFillColor(Color::Red);
        return true;
    }
    else return false;
}

bool MoveDownDifficult(Text button[], int& number) {
    if (number + 1 >= 1) {
        button[number].setFillColor(Color::White);
        number++;
        if (number == 3)
            number = 0;
        button[number].setFillColor(Color::Red);
        return true;
    }
    else return false;
}

void buttonEasyCondition(Text buttonDifficult[], int& numberDifficult, int&
checkMode) {
    if (Mouse::getPosition().x >= 960 - 5 -
buttonDifficult[0].getGlobalBounds().width / 2 &&
        Mouse::getPosition().y >= 440 -
buttonDifficult[0].getGlobalBounds().height / 2 &&
        Mouse::getPosition().x <= 960 +
buttonDifficult[0].getGlobalBounds().width / 2 &&
        Mouse::getPosition().y <= 440 +
buttonDifficult[0].getGlobalBounds().height / 2) {
        numberDifficult = 0;
        buttonDifficult[0].setFillColor(Color::Red);
        buttonDifficult[1].setFillColor(Color::White);
        buttonDifficult[2].setFillColor(Color::White);
        if (Mouse::isButtonPressed(Mouse::Left)) {
            checkMode = 2;
        }
    }
}

void buttonNormalCondition(Text buttonDifficult[], int& numberDifficult, int&
checkMode) {
    if (Mouse::getPosition().x >= 960 - 5 -
buttonDifficult[1].getGlobalBounds().width / 2 &&
        Mouse::getPosition().y >= 540 -
buttonDifficult[1].getGlobalBounds().height / 2 &&
        Mouse::getPosition().x <= 960 +
buttonDifficult[1].getGlobalBounds().width / 2 &&
        Mouse::getPosition().y <= 540 +
buttonDifficult[1].getGlobalBounds().height / 2) {
        numberDifficult = 1;
```

```
        buttonDifficult[1].setFillColor(Color::Red);
        buttonDifficult[0].setFillColor(Color::White);
        buttonDifficult[2].setFillColor(Color::White);
        if (Mouse::isButtonPressed(Mouse::Left)) {
            checkMode = 2;
        }
    }
}

void buttonHardCondition(Text buttonDifficult[], int& numberDifficult, int&
checkMode) {
    if (Mouse::getPosition().x >= 960 - 5 -
buttonDifficult[2].getGlobalBounds().width / 2 &&
        Mouse::getPosition().y >= 630 -
buttonDifficult[2].getGlobalBounds().height / 2 &&
        Mouse::getPosition().x <= 960 +
buttonDifficult[2].getGlobalBounds().width / 2 &&
        Mouse::getPosition().y <= 630 +
buttonDifficult[2].getGlobalBounds().height / 2) {
        numberDifficult = 2;
        buttonDifficult[2].setFillColor(Color::Red);
        buttonDifficult[1].setFillColor(Color::White);
        buttonDifficult[0].setFillColor(Color::White);
        if (Mouse::isButtonPressed(Mouse::Left)) {
            checkMode = 2;
        }
    }
}
```

fileLoad.h :

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

using namespace sf;

bool loadTextureFromFile(Texture& texture, std::string path);

bool loadFontFromFile(Font& font, std::string path);
```

fileLoad.cpp :

```
#include <fileLoad.h>

bool loadTextureFromFile(Texture& texture, std::string path) {
    if (!texture.loadFromFile(path)) return false;
    return true;
}

bool loadFontFromFile(Font& font, std::string path) {
    if (!font.loadFromFile(path)) return false;
    return true;
}
```

game.h :

```
#pragma once

#include <chrono>
#include <thread>
#include <time.h>
#include <iostream>
#include <stdlib.h>
#include <SFML/Graphics.hpp>

using namespace sf;

void initFrame(RectangleShape& board, float xSize, float ySize, float xPos,
float yPos);

void buttonBack(RectangleShape logOutButton, RenderWindow& window, int&
checkMode);

void buttonRefresh(RectangleShape refreshButton, RenderWindow& window,
RectangleShape& cube, Text& letter, int& sumLetters, int& correctTypes, int&
incorrectTypes, int& flagStart);

void startTimer(Text &timeMessage, Clock& clock, RectangleShape& cube, Text&
letter, int& checkMode, int& flagStart);

bool selectDifficult(int& numberDifficult, float& speedFall);

void startPositionCube(float& xPos, RectangleShape& cube, int& numberLetter,
Text& letter, std::string *letters);

void gameKey(RenderWindow& window, RectangleShape& cube, Text& letter,
std::string *letters, int& numberDifficult, int& flagCorrect, int& correctTypes,
int& incorrectTypes, int& sumLetters, int& numberLetter);

void checkCorrect(Event& ev, int& numberLetter, int& correctTypes, int&
flagCorrect, int& sumLetters, int& incorrectTypes);
```

game.cpp :

```
#include <game.h>

void initFrame(RectangleShape& board, float xSize, float ySize, float xPos,
float yPos) {
    board.setSize(Vector2f(xSize, ySize));      //Размер рамки
    board.setFillColor(Color(0, 0, 0, 0));      //Внутренняя часть рамки
прозрачная
    board.setOutlineThickness(2);               //Толщина рамки
    board.setOutlineColor(Color::White);        //Цвет рамки
    xPos = xPos - board.getSize().x / 2;
    yPos = yPos - board.getSize().y / 2;
    board.setPosition(xPos, yPos);              //Позиция рамки
}

void buttonBack(RectangleShape logOutButton, RenderWindow& window, int&
checkMode) {
    if (Mouse::getPosition().x >= 25 &&
        Mouse::getPosition().y >= 30  &&
```

```
        Mouse::getPosition().x <= 25 + logOutButton.getSize().x &&
        Mouse::getPosition().y <= 30 + logOutButton.getSize().y) {
        if (Mouse::isButtonPressed(Mouse::Left)) {
            checkMode = 0;
        }
    }
}

void buttonRefresh(RectangleShape refreshButton, RenderWindow& window,
RectangleShape& cube, Text& letter, int& sumLetters, int& correctTypes, int&
incorrectTypes, int& flagStart) {
    if (Mouse::getPosition().x >= 1835 &&
        Mouse::getPosition().y >= 30  &&
        Mouse::getPosition().x <= 1835 + refreshButton.getSize().x &&
        Mouse::getPosition().y <= 30 + refreshButton.getSize().y) {
        if (Mouse::isButtonPressed(Mouse::Left)) {
            sumLetters = 0;
            correctTypes = 0;
            incorrectTypes = 0;
            flagStart = 0;
            cube.setPosition(945, 110);
            letter.setPosition(945 + 12, 110);
        }
    }
}

void startTimer(Text &timeMessage, Clock& clock, RectangleShape& cube, Text&
letter, int& checkMode, int& flagStart) {
    int timer = clock.getElapsedTime().asSeconds();
    String timerStr = L"Время : " + std::to_string(25 - timer) + L" сек";
    float xPos = 980 - 10 - timeMessage.getGlobalBounds().width / 2;
//Выравнивание по X
    float yPos = 1020 - 20 - timeMessage.getGlobalBounds().height / 2;
//Выравнивание по Y
    timeMessage.setString(timerStr);
    timeMessage.setPosition(xPos, yPos);   //Позиция текста

    if (timer >= 25)  {
        clock.restart();
        cube.setPosition(945, 110);
        letter.setPosition(945 + 12, 110);
        checkMode = 3;
        flagStart = 0;
    }
}

bool selectDifficult(int& numberDifficult, float& speedFall) {
    switch (numberDifficult) {
        case 0:
            speedFall = 4;
            return true;
            break;
        case 1:
            speedFall = 8;
            return true;
            break;
        case 2:
            speedFall = 12;
            return true;
            break;
        default:
```

```
            return false;
            break;
    }
}

void startPositionCube(float& xPos, RectangleShape& cube, int& numberLetter,
Text& letter, std::string *letters) {
    cube.setPosition(xPos, 75);
    numberLetter = rand() % (25 + 1);
    letter.setString(letters[numberLetter]);
    letter.setPosition(xPos + 12, 75);
}

void gameKey(RenderWindow& window, RectangleShape& cube, Text& letter,
std::string *letters, int& numberDifficult, int& flagCorrect, int& correctTypes,
int& incorrectTypes, int& sumLetters, int& numberLetter) {
    srand(time(NULL));
    float speedFall = 0;
    selectDifficult(numberDifficult, speedFall);
    float xPos = rand() % 1220 + 320;
    if (flagCorrect == 0) {
        if (cube.getPosition().y <= 768) {
            cube.move(0, speedFall);
            letter.move(0, speedFall);
        } else if (cube.getPosition().y >= 768) {
            startPositionCube(xPos, cube, numberLetter, letter, letters);
            incorrectTypes++;
            sumLetters++;
            flagCorrect = 0;
        }
    } else if (flagCorrect != 0) {
            startPositionCube(xPos, cube, numberLetter, letter, letters);
            flagCorrect = 0;
        }
    window.draw(letter);
    window.draw(cube);
}

void checkCorrect(Event& ev, int& numberLetter, int& correctTypes, int&
flagCorrect, int& sumLetters, int& incorrectTypes) {
    if (numberLetter == ev.key.code) {
        correctTypes++;
        flagCorrect = 1;
        sumLetters++;
    }
    else if (numberLetter != ev.key.code && ev.key.code >= 0 && ev.key.code <
26) {
        incorrectTypes++;
        sumLetters++;
    }
}
```

menu.h :

```
#pragma once

#include <iostream>
#include <SFML/Graphics.hpp>

using namespace sf;

void initText(Text& text, Font& font, int size, String str, float xPos, float
yPos,  Color textColor);

void initButton(RectangleShape& button, float xPos, float yPos, Texture& t);

bool MoveUp(Text button[], int& number);

bool MoveDown(Text button[], int& number);

void buttonStartCondition(Text buttonMenu[], int& numberButton, int& checkMode);

void buttonExitCondition(Text buttonMenu[], RenderWindow& window, int&
numberButton, int& checkMode);
```

menu.cpp :

```
#include <menu.h>

void initText(Text& text, Font& font, int size, String str, float xPos, float
yPos,  Color textColor) {
    text.setFont(font);              //Шрифт
    text.setStyle(Text::Bold);       //Толщина шрифта
    text.setCharacterSize(size);     //Размер шрифта
    text.setString(str);             //Текст

    xPos = xPos - 10 - text.getGlobalBounds().width / 2;     //Выравнивание по X
    yPos = yPos - 20 - text.getGlobalBounds().height / 2;    //Выравнивание по Y

    text.setPosition(xPos, yPos);    //Позиция текста
    text.setFillColor(textColor);    //Цвет текста

}

void initButton(RectangleShape& button, float xPos, float yPos, Texture& t) {
    button.setSize(Vector2f(64, 64));
    t.setSmooth(true);
    button.setTexture(&t);
    button.setPosition(xPos, yPos);
}

bool MoveUp(Text button[], int& number) {
    if (number - 1 >= -1) {
        button[number].setFillColor(Color::White);
        number--;
        if (number == -1)
            number = 1;
        button[number].setFillColor(Color::Red);
        return true;
```

```
    }
    else return false;
}

bool MoveDown(Text button[], int& number) {
    if (number + 1 >= 1) {
        button[number].setFillColor(Color::White);
        number++;
        if (number == 2)
            number = 0;
        button[number].setFillColor(Color::Red);
        return true;
    }
    else return false;
}

void buttonStartCondition(Text buttonMenu[], int& numberButton, int& checkMode)
{
    if (Mouse::getPosition().x >= 960 - 5 -
buttonMenu[0].getGlobalBounds().width / 2 &&
        Mouse::getPosition().y >= 700 - buttonMenu[0].getGlobalBounds().height /
2 &&
        Mouse::getPosition().x <= 960 + buttonMenu[0].getGlobalBounds().width /
2 &&
        Mouse::getPosition().y <= 700 + buttonMenu[0].getGlobalBounds().height /
2) {
        numberButton = 0;
        buttonMenu[0].setFillColor(Color::Red);
        buttonMenu[1].setFillColor(Color::White);
        if (Mouse::isButtonPressed(Mouse::Left)) {
            checkMode = 1;
        }
    }
}

void buttonExitCondition(Text buttonMenu[], RenderWindow& window, int&
numberButton, int& checkMode) {
    if (Mouse::getPosition().x >= 960 - 5 -
buttonMenu[1].getGlobalBounds().width / 2 &&
        Mouse::getPosition().y >= 800 - buttonMenu[1].getGlobalBounds().height /
2 &&
        Mouse::getPosition().x <= 960 + buttonMenu[1].getGlobalBounds().width /
2 &&
        Mouse::getPosition().y <= 800 + buttonMenu[1].getGlobalBounds().height /
2) {
        numberButton = 1;
        buttonMenu[1].setFillColor(Color::Red);
        buttonMenu[0].setFillColor(Color::White);
        if (Mouse::isButtonPressed(Mouse::Left)) {
            window.close();
        }
    }
}
```

mode.h :

```
#pragma once

#include <iostream>
#include <SFML/Graphics.hpp>
#include <menu.h>
#include <difficult.h>
#include <game.h>

using namespace sf;

void modeMenu(RenderWindow& window, Event& ev, Text buttonMenu[], int&
numberButton, int& checkMode);

void modeDifficult(RenderWindow &window, Event &ev, Text buttonDifficult[],
RectangleShape& logOutButton, int& numberDifficult, int& checkMode, int&
sumLetters, int& correctTypes, int& incorrectTypes);

void modeGame(RenderWindow &window, Event &ev, RectangleShape& logOutButton,
RectangleShape& refreshButton, Clock& clock, RectangleShape& cube, Text& letter,
int& checkMode, int& flagStart, int& sumLetters, int& numberLetter, int&
correctTypes, int& flagCorrect, int& incorrectTypes);

void modeResult(RenderWindow& window, Event& ev, RectangleShape& logOutButton,
int& checkMode, int& sumLetters, int& correctTypes, int& incorrectTypes);
```

mode.cpp :

```
#include <mode.h>

void modeMenu(RenderWindow& window, Event &ev, Text buttonMenu[], int&
numberButton, int& checkMode) {
    if (checkMode == 0) {
        buttonStartCondition(buttonMenu, numberButton, checkMode);
        buttonExitCondition(buttonMenu, window, numberButton, checkMode);
        switch (ev.type) {
            case Event::KeyPressed :
                switch (ev.key.code) {
                    case Keyboard::Escape :
                        window.close();
                    case Keyboard::Up :
                        MoveUp(buttonMenu, numberButton);
                        break;
                    case Keyboard::Down :
                        MoveDown(buttonMenu, numberButton);
                        break;
                    case Keyboard::Enter :
                        if (numberButton == 0) {
                            checkMode = 1;
                            ev.key.code = Keyboard::Unknown;
                        }
                        else if (numberButton == 1)
                            window.close();
                }
        }
    }
```

```
}

void modeDifficult(RenderWindow &window, Event &ev, Text buttonDifficult[],
RectangleShape& logOutButton,
                   int& numberDifficult, int& checkMode, int& sumLetters, int&
correctTypes, int& incorrectTypes) {
    if (checkMode == 1) {
        buttonBack(logOutButton, window, checkMode);
        buttonEasyCondition(buttonDifficult, numberDifficult, checkMode);
        buttonNormalCondition(buttonDifficult, numberDifficult, checkMode);
        buttonHardCondition(buttonDifficult, numberDifficult, checkMode);
        sumLetters = 0;
        correctTypes = 0;
        incorrectTypes = 0;
        switch (ev.type) {
            case Event::KeyPressed :
                switch (ev.key.code) {
                    case Keyboard::Escape :
                        checkMode = 0;
                        numberDifficult = 0;
                        break;
                    case Keyboard::Up :
                        MoveUpDifficult(buttonDifficult, numberDifficult);
                        break;
                    case Keyboard::Down :
                        MoveDownDifficult(buttonDifficult, numberDifficult);
                        break;
                    case Keyboard::Enter :
                        checkMode = 2;
                }
        }
    }
}

void modeGame(RenderWindow &window, Event &ev, RectangleShape& logOutButton,
RectangleShape& refreshButton,
              Clock& clock, RectangleShape& cube, Text& letter, int& checkMode,
int& flagStart, int& sumLetters,
              int& numberLetter, int& correctTypes, int&flagCorrect, int&
incorrectTypes) {
    if (checkMode == 2) {
        buttonBack(logOutButton, window, checkMode);
        buttonRefresh(refreshButton, window, cube, letter, sumLetters,
correctTypes, incorrectTypes, flagStart);
        switch (ev.type) {
            case Event::KeyPressed :
                switch (ev.key.code) {
                    case Keyboard::Space :
                        flagStart = 1;
                        clock.restart();
                        break;
                    case Keyboard::Escape :
                        checkMode = 1;
                        flagStart = 0;
                        sumLetters = 0;
                        correctTypes = 0;
                        incorrectTypes = 0;
                        cube.setPosition(945, 110);
                        letter.setPosition(945 + 12, 110);
                        break;
                    default :
```

```
                        checkCorrect(ev, numberLetter, correctTypes,
flagCorrect, sumLetters, incorrectTypes);
                        break;
                }
        }
    }
}

void modeResult(RenderWindow& window, Event& ev, RectangleShape& logOutButton,
int& checkMode, int& sumLetters, int& correctTypes, int& incorrectTypes) {
    if (checkMode == 3) {
        buttonBack(logOutButton, window, checkMode);
        switch (ev.type) {
            case Event::KeyPressed :
                switch (ev.key.code) {
                    case Keyboard::Escape :
                        sumLetters = 0;
                        correctTypes = 0;
                        incorrectTypes = 0;
                        checkMode = 0;
                        break;
                }
        }
    }
}
```

window.h :

```
#pragma once

#include <iostream>
#include <stdlib.h>
#include <SFML/Graphics.hpp>
#include <game.h>

using namespace sf;

void windowMenu(RenderWindow& window, Text& title, Text& button1, Text&
button2);

void windowDifficult (RenderWindow& window, RectangleShape& logOutButton, Text&
button1, Text& button2, Text& button3);

void windowGame(RenderWindow& window, RectangleShape& board, Text typeGamelvl[],
Text& noticeMessage, Text& timeMessage, RectangleShape& logOutButton,
RectangleShape& refreshButton, Clock& clock, int& timer, RectangleShape& cube,
Text& letter, std::string *letters, int& flagStart, int& checkMode, int&
numberDifficult, int& flagCorrect, int& correctTypes, int& incorrectTypes, int&
sumLetters, int& numberLetter);

void windowResult(RenderWindow& window, Font& font, Text& titleResult, Text&
textSumLetters, Text& textCorrectTypes, Text& textIncorrectTypes,
RectangleShape& logOutButton, int& sumLetters, int& correctTypes, int&
incorrectTypes);
```

window.cpp :

```
#include <window.h>
#include <menu.h>

void windowMenu(RenderWindow& window, Text& title, Text& button1, Text& button2)
{
    window.draw(title);     //Заголовок
    window.draw(button1);   //Кнопка "Начать"
    window.draw(button2);   //Кнопка "Выход"
}

void windowDifficult (RenderWindow& window, RectangleShape& logOutButton, Text&
button1, Text& button2, Text& button3) {
    window.draw(logOutButton);
    window.draw(button1);   //Кнопка "Легко"
    window.draw(button2);   //Кнопка "Нормально"
    window.draw(button3);   //Кнопка "Сложно"
}

void windowGame(RenderWindow& window, RectangleShape& board, Text typeGamelvl[],
Text& noticeMessage, Text& timeMessage, RectangleShape& logOutButton,
RectangleShape& refreshButton, Clock& clock, int& timer,RectangleShape& cube,
Text& letter, std::string *letters, int& flagStart, int& checkMode, int&
numberDifficult, int& flagCorrect, int& correctTypes, int& incorrectTypes, int&
sumLetters, int& numberLetter) {
    window.draw(board);
    window.draw(typeGamelvl[numberDifficult]);
    if (flagStart == 0) window.draw(noticeMessage);
    else if (flagStart == 1)  {
        startTimer(timeMessage, clock, cube, letter, checkMode, flagStart);
        gameKey(window, cube, letter, letters, numberDifficult, flagCorrect,
correctTypes, incorrectTypes, sumLetters, numberLetter);
        window.draw(timeMessage);
    }
    window.draw(logOutButton);
    window.draw(refreshButton);
}

void windowResult(RenderWindow& window, Font& font, Text& titleResult, Text&
textSumLetters, Text& textCorrectTypes, Text& textIncorrectTypes,
RectangleShape& logOutButton, int& sumLetters, int& correctTypes, int&
incorrectTypes) {
    initText(textSumLetters, font, 80, L"Количество букв : " +
std::to_string(sumLetters), 960, 540, Color::White);
    initText(textCorrectTypes, font, 80, L"Правильных нажатий : " +
std::to_string(correctTypes), 960, 640, Color::White);
    initText(textIncorrectTypes, font, 80, L"Неправильных нажатий : " +
std::to_string(incorrectTypes), 960, 740, Color::White);
    window.draw(titleResult);
    window.draw(textSumLetters);
    window.draw(textCorrectTypes);
    window.draw(textIncorrectTypes);
    window.draw(logOutButton);
}
```

parser_test.cpp :

```
#include <ctest.h>
#include <fileLoad.h>
#include <difficult.h>
#include <menu.h>
#include <game.h>

//проверка загрузки шрифта
CTEST (ctest, loadFontFromFile) {
    Font fontTrue;
    bool checkFontTrue = loadFontFromFile(fontTrue,
"Resource/Font/Raleway/static/Raleway-Thin.ttf") ;
    ASSERT_TRUE(checkFontTrue);


    Font fontFalse;
    bool checkFontFalse = loadFontFromFile(fontFalse,
"Resource/Font/Raleway/static/Ralewy-Thin.ttf") ;
    ASSERT_FALSE(checkFontFalse);
}

//проверка загрузки заднего фона
CTEST (ctest, loadTextureFromFile) {
    Texture screenTrue;
    bool checkScreenTrue = loadTextureFromFile(screenTrue,
"Resource/Pictures/Background/3.jpg") ;
    ASSERT_TRUE(checkScreenTrue);


    Texture screenFalse;
    bool checkScreenFalse = loadTextureFromFile(screenFalse,
"Resource/Pictures/Background/3333.jpg") ;
    ASSERT_FALSE(checkScreenFalse);
}

//проверка переключения клавиш
CTEST (ctest, MoveUp) {
    Text button[2];
    int numberTrue = 0;
    bool MoveUpTrue = MoveUp(button, numberTrue);
    ASSERT_TRUE(MoveUpTrue);

    int numberFalse = -4;
    bool MoveUpFalse = MoveUp(button, numberFalse);
    ASSERT_FALSE(MoveUpFalse);
}

//проверка переключения клавиш
CTEST (ctest, MoveDown) {
    Text button[2];
    int numberTrue = 1;
    bool MoveUpTrue = MoveUp(button, numberTrue);
    ASSERT_TRUE(MoveUpTrue);

    int numberFalse = -1;
    bool MoveUpFalse = MoveUp(button, numberFalse);
    ASSERT_FALSE(MoveUpFalse);
```

```
}


//проверка переключения клавиш
CTEST (ctest, MoveUpDifficult) {
    Text button[3];
    int numberTrue = 0;
    bool MoveUpDifficultTrue = MoveUpDifficult(button, numberTrue);
    ASSERT_TRUE(MoveUpDifficultTrue);

    int numberFalse = -5;
    bool MoveUpDifficultFalse = MoveUpDifficult(button, numberFalse);
    ASSERT_FALSE(MoveUpDifficultFalse);
}

//проверка переключения клавиш
CTEST (ctest, MoveDownDifficult) {
    Text button[3];
    int numberTrue = 0;
    bool MoveDownDifficultTrue = MoveDownDifficult(button, numberTrue);
    ASSERT_TRUE(MoveDownDifficultTrue);

    int numberFalse = -5;
    bool MoveDownDifficultFalse = MoveDownDifficult(button, numberFalse);
    ASSERT_FALSE(MoveDownDifficultFalse);
}

//проверка выбора уровня сложности
CTEST (ctest, selectDifficult) {
    int numberDifficultTrue = 0;
    float speedFall;
    bool selectDifficultTrue = selectDifficult(numberDifficultTrue, speedFall);
    ASSERT_TRUE(selectDifficultTrue );

    int numberDifficultFalse = -4;
    bool selectDifficultFalse = selectDifficult(numberDifficultFalse,
speedFall);
    ASSERT_FALSE(selectDifficultFalse );
}

//проверка игрового процесса
CTEST (ctest, startPositionCube) {
    float xPos = 600;
    RectangleShape cube;
    int numberLetter;
    Text letter;
    std::string letters[26];
    startPositionCube(xPos, cube, numberLetter, letter, letters);
    ASSERT_TRUE(cube.getPosition().x == xPos);
}

//проверка событий окна результата
CTEST(renderwindow_test, mode_result_test) {
    RenderWindow window(VideoMode(800, 600), "Mode Result Test");

    RectangleShape logOutButton(sf::Vector2f(100, 50));
    int checkMode = 3;
    int sumLetters = 10;
    int correctTypes = 5;
    int incorrectTypes = 2;
```

```
    Event ev;
    ev.type = Event::KeyPressed;
    ev.key.code = Keyboard::Escape;
    modeResult(window, ev, logOutButton, checkMode, sumLetters, correctTypes,
incorrectTypes);

    ASSERT_EQUAL(0, checkMode);
    ASSERT_EQUAL(0, sumLetters);
    ASSERT_EQUAL(0, correctTypes);
    ASSERT_EQUAL(0, incorrectTypes);
}

//проверка событий окна меню
CTEST(renderwindow_test, mode_menu_test) {
    RenderWindow window(VideoMode(800, 600), "Mode Menu Test");
    Font font;
    Text buttonMenu[2];
    buttonMenu[0].setFont(font);
    buttonMenu[0].setString("Start");
    buttonMenu[1].setFont(font);
    buttonMenu[1].setString("Exit");
    int numberButton = 0;
    int checkMode = 0;
    Event ev;
    ev.type = Event::KeyPressed;

    ev.key.code = Keyboard::Up;
    modeMenu(window, ev, buttonMenu, numberButton, checkMode);
    ASSERT_EQUAL(1, numberButton);

    ev.key.code = Keyboard::Down;
    modeMenu(window, ev, buttonMenu, numberButton, checkMode);
    ASSERT_EQUAL(0, numberButton);

    ev.key.code = Keyboard::Enter;
    numberButton = 1;
    modeMenu(window, ev, buttonMenu, numberButton, checkMode);
    ASSERT_EQUAL(0, window.isOpen());
}
```

Makefile :

```
APP_NAME = main
LIB_NAME = lib
TEST_NAME = testmain
    TESTFLAGS = -I thirdparty
SFML_FLAGS = -lsfml-graphics -lsfml-system -lsfml-window
CFLAGS = -I src/lib
DEPSFLAGS = -MMD
CC = g++
BIN_DIR = bin
OBJ_DIR = obj
SRC_DIR = src
TEST_DIR = test

APP_PATH = $(BIN_DIR)/$(APP_NAME)
LIB_PATH = $(OBJ_DIR)/$(SRC_DIR)/$(LIB_NAME)/$(LIB_NAME).a
TEST_PATH = $(BIN_DIR)/$(TEST_NAME)
APP_SOURCES = $(wildcard $(SRC_DIR)/$(APP_NAME)/*.cpp)
APP_OBJECTS = $(patsubst %.cpp, $(OBJ_DIR)/%.o, $(APP_SOURCES))
LIB_SOURCES = $(wildcard $(SRC_DIR)/$(LIB_NAME)/*.cpp)
LIB_OBJECTS = $(patsubst %.cpp, $(OBJ_DIR)/%.o, $(LIB_SOURCES))
TEST_SOURCES = $(wildcard $(TEST_DIR)/*.cpp)
TEST_OBJECTS = $(patsubst %.cpp, $(OBJ_DIR)/%.o, $(TEST_SOURCES))

DEPS = $(APP_OBJECTS:.o=.d) $(LIB_OBJECTS:.o=.d)

all: $(APP_PATH)

-include $(DEPS)

$(APP_PATH): $(APP_OBJECTS) $(LIB_PATH)
    $(CC) $(CFLAGS) -o $@ $^ $(SFML_FLAGS)

$(LIB_PATH): $(LIB_OBJECTS)
    ar rcs $@ $^

$(OBJ_DIR)/%.o: %.cpp
    $(CC) $(CFLAGS) $(DEPSFLAGS) -c -o $@ $<

test: $(LIB_PATH) $(TEST_PATH)
    $(TEST_PATH)

$(TEST_PATH): $(TEST_OBJECTS) $(LIB_PATH)
    $(CC) $(TESTFLAGS) $(CFLAGS) -o $@ $^ $(SFML_FLAGS)

$(OBJ_DIR)/test/main.o: test/main.cpp
    $(CC) $(TESTFLAGS) $(CFLAGS) $(DEPSFLAGS) -c -o $@ $<

$(OBJ_DIR)/test/parser_test.o: test/parser_test.cpp
    $(CC) $(TESTFLAGS) $(CFLAGS) $(DEPSFLAGS) -c -o $@ $<

run: $(APP_PATH)
    ./bin/main

clean:
    $(RM) $(APP_PATH) $(TEST_PATH) $(OBJ_DIR)/*/*/*.[aod]
$(OBJ_DIR)/test/*.[aod]
```