

# **Содержание**

[**Введение 4**](#_Toc168362966)

[**1. Постановка задачи. 5**](#_Toc168362967)

[**2. Выбор решения. 6**](#_Toc168362968)

[**3. Отладка и тестирование. 8**](#_Toc168362969)

[**4. Описание программы. 12**](#_Toc168362970)

[**5. Руководство пользователя. 15**](#_Toc168362971)

[**Заключение 19**](#_Toc168362972)

[**Список используемых источников 20**](#_Toc168362973)

[**Приложение А Листинг программы 21**](#_Toc168362974)

Введение

Компьютерные игры давно стали неотъемлемой частью современной культуры и индустрии развлечений. С момента появления первых аркадных игр в 1970-х годах, индустрия видеоигр пережила стремительное развитие, превратившись в одну из крупнейших форм медиа и искусства.

Игра «Тетрис», созданная в 1984 советским программистом Алексеем Пажитновым, стала культурным феноменом благодаря своей простой, но увлекательной механике.

Основная идея игры заключается в манипуляции падающими геометрическими фигурами, называемыми тетромино, с целью создания и уничтожения горизонтальных линий на игровом поле. Несмотря на свою кажущуюся простоту, «Тетрис» предоставляет игрокам увлекательный и сложный игровой опыт, который требует стратегического мышления, быстрых рефлексов и способности планировать на несколько шагов вперед.

Популярность «Тетриса» подтверждается его широким признанием во всем мире и включением в многочисленные платформы — от аркадных автоматов и домашних консолей до современных смартфонов и компьютеров. Игра продолжает привлекать новые поколения игроков и остается символом эпохи цифровых развлечений.

1. Постановка задачи.

Необходимо разработать игру «Тетрис». Реализовать управление с клавиатуры. Игровое поле должно быть 20 строк и 10 столбцов. Лучшие 10 результатов игроков должны выводиться на экран в конце каждой игры и сохраняться в файл.

Многомодульность. Многомодульная программа должна быть организована в несколько файлов или модулей, каждый из которых содержит функции или классы для конкретных задач. Это улучшает читаемость, упрощает поддержку и тестирование, а также способствует повторному использованию и улучшению кода.

Использование сложных типов данных – массивов, структур, файлов. Это необходимо для простой обработки данных в коде.

Режим работы видеосистемы – текстовый/графический. Для начала необходимо определиться с типом интерфейса и с элементами управления, затем необходимо изучить способы их реализации.

Устройство ввода-вывода – клавиатура и мышь. Необходимо осуществлять обработку действий для взаимодействий с интерфейсом или самой игрой.

Интерфейс должен основываться на меню. Должны использоваться инструменты, облегчающие взаимодействие с игрой.

Заставка необходима для того, чтобы запустивший игру пользователь имел достаточную информацию о ней.

2. Выбор решения.

При разработке программы я использовал стандартные правила игры «Тетрис». Алгоритм разработки включает в себя несколько пунктов:

1.Создание игрового поля, которое представляет собой сетку квадратных ячеек. Стандартный размер поля включает в себя 10 столбцов и 20 строк.

2. Создание физики тетромино, проработка их генерации а так же цикличного алгоритма для появления фигур друг за другом.

3. Создание алгоритма для управления тетромино, проработка ускоренного падения, перемещения по горизонтали и поворота фигуры вокруг своей оси.

4. Создание обработки столкновений, исчезновение заполненных строк и определения конца игры, включает в себя проверку на столкновения с фигурами краями поля и проведения связи для обработки появления следующей фигуры. Реализация проигрыша и исчезновения строк, включает в себя проверку полей на определенное заполнение фигурами.

5. Счета и игрового интерфейс позволили связать все пункты создания игры.

Такой алгоритм позволяет создавать функциональные части кода отдельно и с легкостью исправлять ошибки.

При запуске программы открывается окно загрузки. Далее появляется меню, состоящее из 2 функциональных кнопок:

1. «Start» - при нажатии кнопки запускается игра: генерируется фигура в невидимой области поля и начинает падать до низа поля. Фигурой можно управлять с клавиатуры. Кнопка «Up» - отвечает за поворот фигуры вокруг своей оси. Кнопки «Left» и «Right» - отвечают за движение фигур влево и вправо. Кнопка «Down» - позволяет падать фигуре с большей скоростью.

После проигрыша, он зачитывается если фигура стоит вплотную к верхней красной линии, выводится экран проигрыша на нем отражено количество удаленных строк, сообщение о проигрыше – «Game Over» и «High Scores» - и ниже отображается лучшие результаты начиная от самого большого к самому маленькому.

2. В верхнем правом углу находится кнопка «MENU». После ее нажатия пользователя перекинет на начальный экран игры, если игра идет она будет остановлена, а счет сброшен, если кнопка будет нажата после выведения экрана проигрыша счет будет сохранен.

1. Отладка и тестирование.

Для разработки была выбрана среда Microsoft Visual Studio 2022, которая предоставляет все необходимые инструменты для создания и отладки программ. В процессе отладки использовались различные возможности Visual Studio, такие как точки останова, трассировка и анализ переменных. Отладка в Visual Studio 2022 включает исследование и устранение ошибок в коде приложения. Существуют несколько методов отладки, которые помогают эффективно находить и исправлять ошибки. Для начала отладки необходимо открыть проект, выбрать пункт "Отладка" в верхнем меню и выбрать нужный метод отладки. Это позволяет приостановить выполнение программы на заданной точке останова или выполнять её пошагово. Отладка в Visual Studio 2022 является мощным инструментом, упрощающим процесс разработки приложений и обнаружения ошибок в коде.

3.1. Установка точки останова и запуск отладчика.

Точки останова полезны, если вы знаете строку или раздел кода, который нужно подробно изучить во время выполнения программы. Более детальную информацию о различных типах точек останова, таких как условные точки останова и точки останова в функциях, можно найти в документации Visual Studio.

Для отладки необходимо запустить приложение с подключенным отладчиком к процессу приложения. Это можно сделать, нажав клавишу F5 (Отладка > Начать отладку), что является самым распространённым методом. Однако, если у вас ещё не задана ни одна точка останова для проверки кода, сначала их следует установить, а затем начать отладку.

Чтобы установить точку останова в открытом файле в редакторе кода, достаточно кликнуть в поле слева от строки кода. После этого нажмите клавишу F5 (Отладка > Начать отладку), и отладчик запустится, останавливаясь в первой заданной точке останова. Если приложение ещё не запущено, нажатие клавиши F5 запустит отладчик и выполнит остановку в первой заданной точке останова.

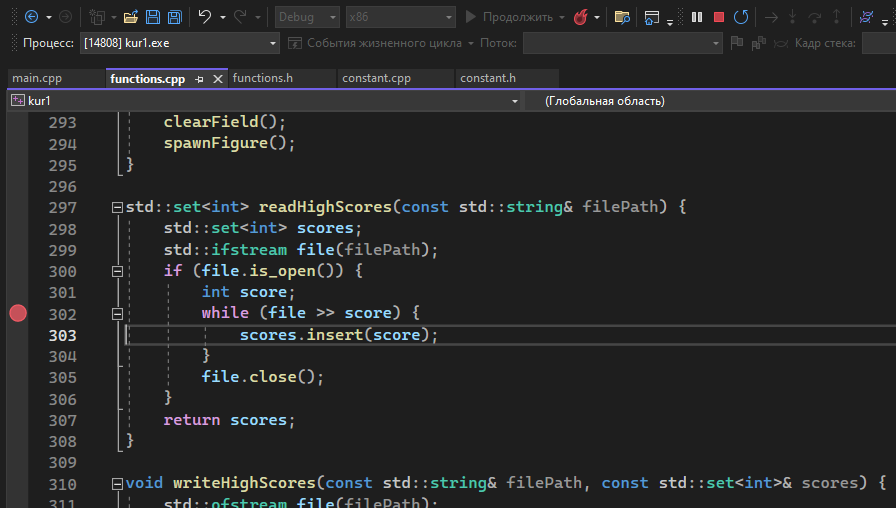


Рисунок 1 – точка остановы и запущенный отладчик

3.2. Переход по коду в отладчике с помощью пошаговых команд.

Для запуска приложения с подключенным отладчиком нажмите клавишу F11 (Отладка > Шаг с заходом). F11 — это команда "Шаг с заходом", которая выполняет приложение, переходя к следующему оператору. При запуске приложения с помощью клавиши F11 отладчик останавливается на первом выполняемом операторе, что позволяет подробно изучить выполнение кода пошагово.

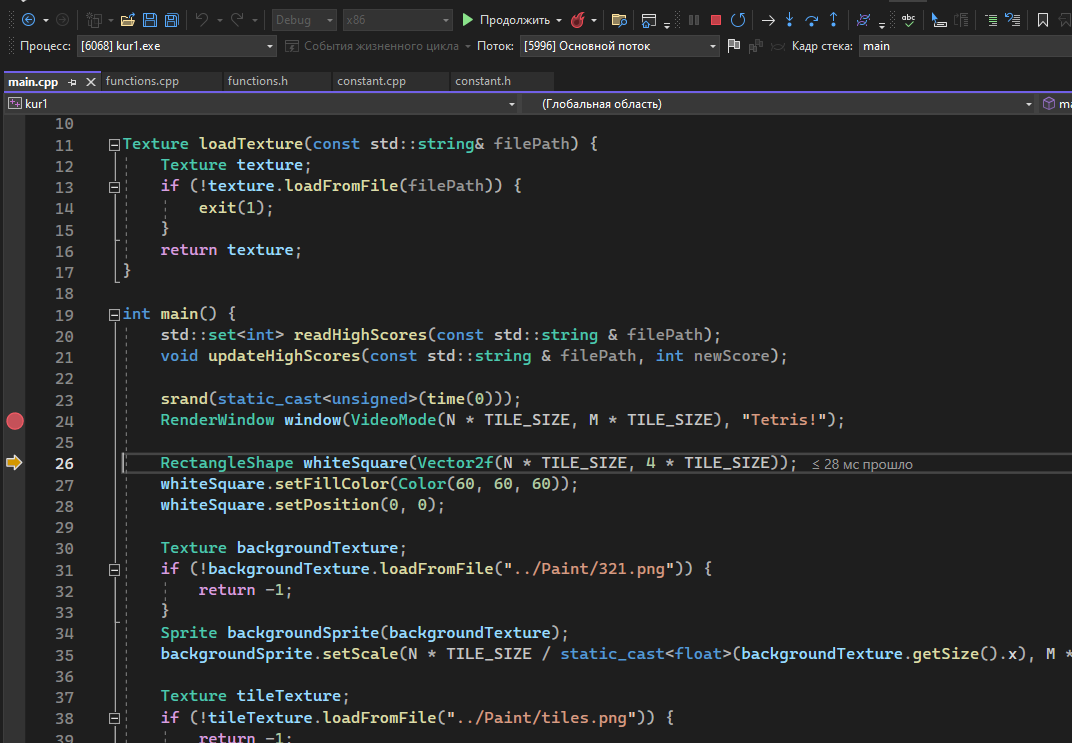


Рисунок 2 – указатель на следующую исполняемую строчку

Желтая стрелка представляет оператор, на котором приостановлен отладчик. В этой же точке приостанавливается выполнение приложения (этот оператор пока не выполнен). Клавишу F11 удобно использовать для более детальной проверки потока выполнения.

3.3. Шаг с обходом по коду для пропуска функций.

В случае, когда вы находитесь на строке кода с вызовом функции или метода, вместо использования клавиши F11, вы можете нажать клавишу F10 (Отладка > Шаг с обходом). Это позволит вам продолжить выполнение программы, минуя вход внутрь функций или методов, и быстро перейти к интересующим вас участкам кода, обходя неактуальные фрагменты. Таким образом, клавиша F10 помогает ускорить процесс отладки, фокусируясь на ключевых частях кода, без необходимости детального просмотра всех вызовов функций.

3.4. Быстрое выполнение до точки в коде с помощью мыши.

Команды «Выполнить до щелчка» - эта команда удобна для быстрого выполнения в видимой области кода вашего приложения и доступна в любом открытом файле.

Во время отладки наведите указатель мыши на строку кода, пока не появится кнопка «Запустить» (выполнить отсюда).

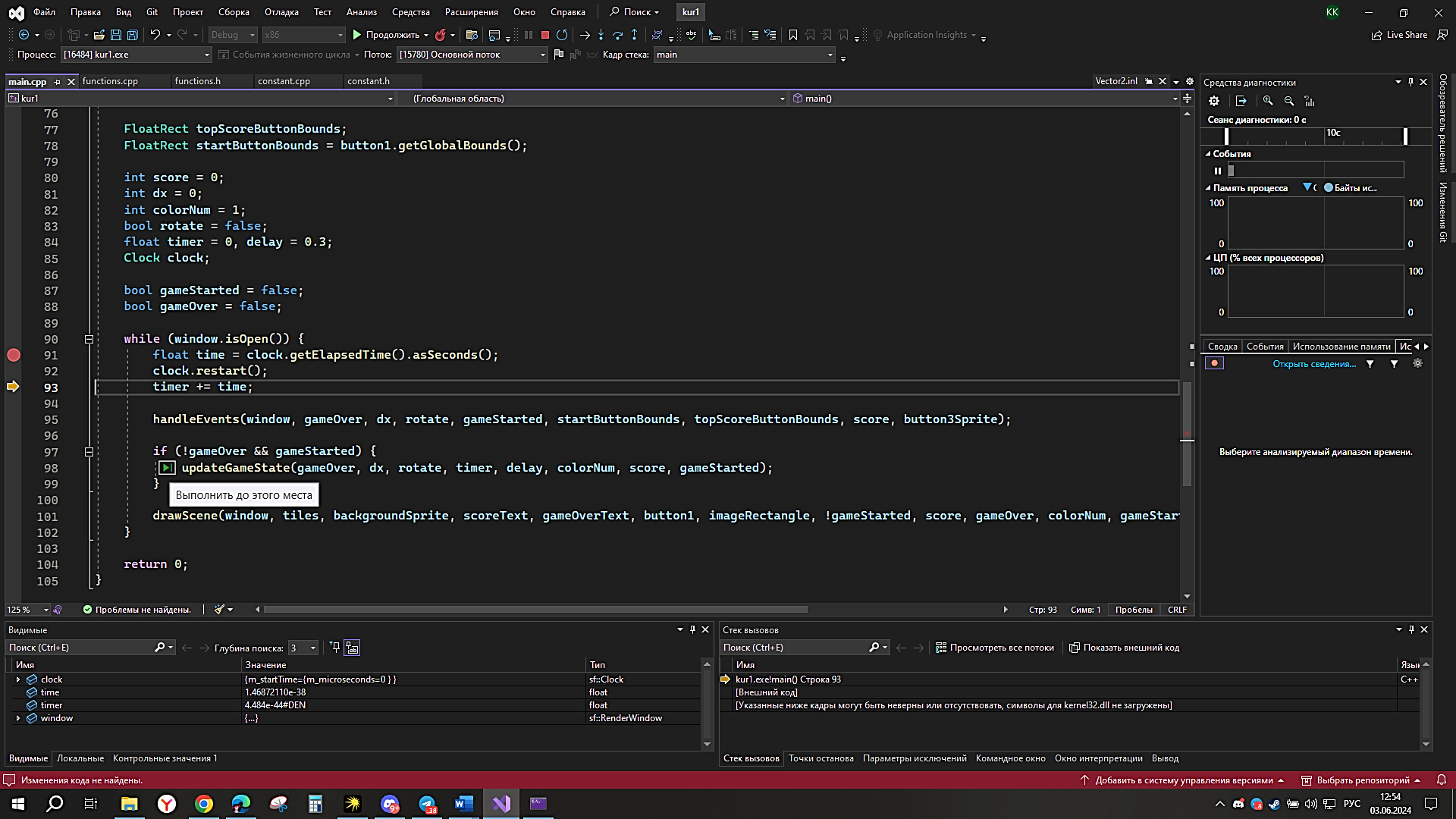


Рисунок 3 – Кнопка «Быстрого выполнения»

Нажмите кнопку «Выполнение до щелчка» (Выполнить до этого места), и отладчик продолжит выполнение до строки кода, на которую вы кликнули.

Тестирование проводилось как во время разработки, так и после ее завершения. Были проверены различные этапы игры. Тщательно протестированы и устранены ошибки при удалении строк, начислении счета и выхода в меню.

4. Описание программы.

Разработанная программа состоит из нескольких модулей:

1. main.cpp – основный файл программы

2. constant.h – заголовочный файл для глобальных переменных

3. constant.cpp – файл, реализующий присвоение значений глобальным переменным

4. fanctions.h – заголовочный файл для функций

5. fanctions.cpp – все функции игры

Приложение Tetris.exe является основным модулем программы. При запуске программы выводиться меню, после чего пользователю необходимо выбрать пункт меню для дальнейшего использования приложения.

Таблица 1.

|  |  |  |
| --- | --- | --- |
| Клавиши, вызывающее событие | Действие пользователя | Действие программы |
| 1. ЛКМ | Выбран пункт «Play» | Запускается игра. |
| 1. ЛКМ | Выбран пункт «MENU» | Возврат в меню. |

Схема программы.

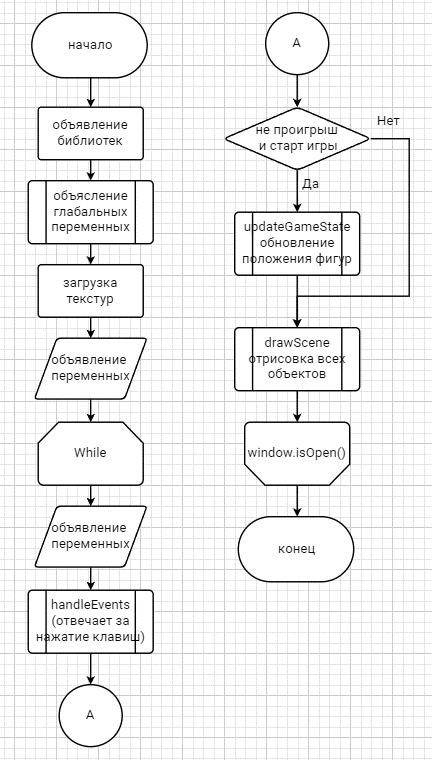


Рисунок 5 – функция main

Функция HandleEvents обрабатывает события пользовательского ввода, такие как нажатия клавиш и щелчки мыши, для управления игрой.

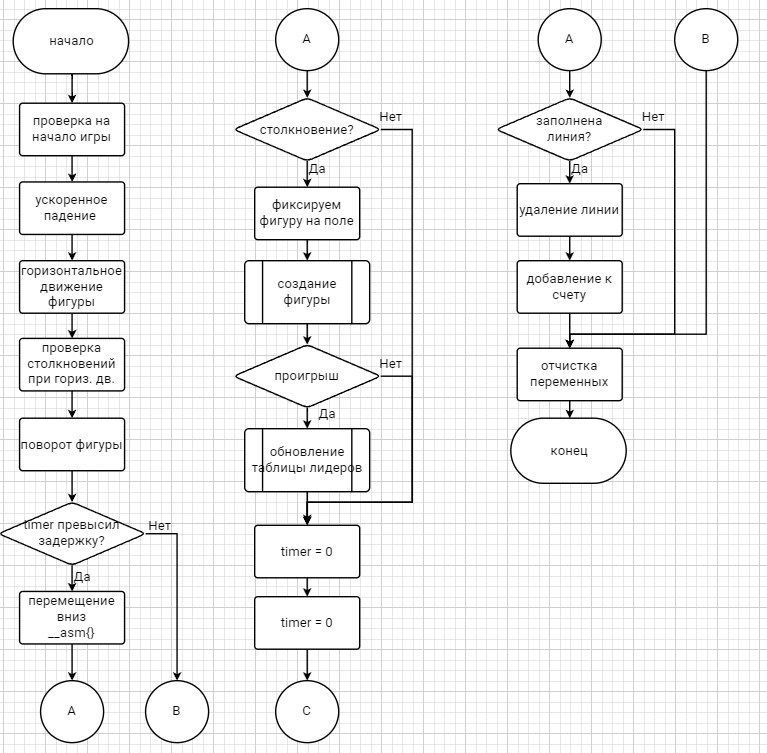


Рисунок 6 – функция updateGameState

Функция drawScene рисует все элементы на экране, включая фон, фигуры, текстовые элементы и кнопки.

Функция checkTopRows проверяет, заполнены ли верхние строки поля, чтобы определить, закончилась ли игра.

Функция spawnFigure создает новую фигуру, выбирая случайную из возможных и устанавливает ее начальные координаты.

Функция checkCollision проверяет столкновение текущей фигуры a с границами поля или другими фигурами.

Функция clearField очищает игровое поле, заполняя массив field нулями.

5. Руководство пользователя.

Программа Tetris.exe это классическая игра Тетрис, где вам нужно размещать падающие фигуры так, чтобы они заполняли горизонтальные линии. Как только линия заполняется, она исчезает, принося вам очки. Игра заканчивается, когда новые фигуры не могут появиться из-за того, что поле заполнено.

В главном меню находится основная кнопка «Start». Она отвечает за начало самой игры.

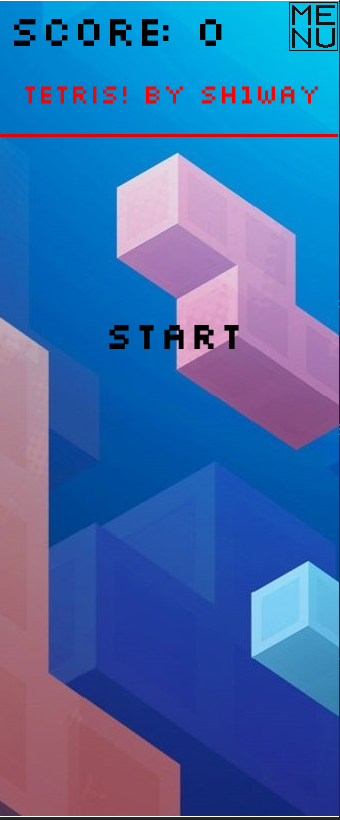


Рисунок 7 – меню

После начала самой игры будет доступно управлять падающими фигурами с помощью клавиш:

Стрелка вверх: Поворот фигуры.

Стрелка вправо: Движение фигуры вправо.

Стрелка влево: Движение фигуры влево.

Стрелка вниз: Быстрое падение фигуры.

В левом верхнем углу пишется счет, 1 очко = 1 удаленной строке.

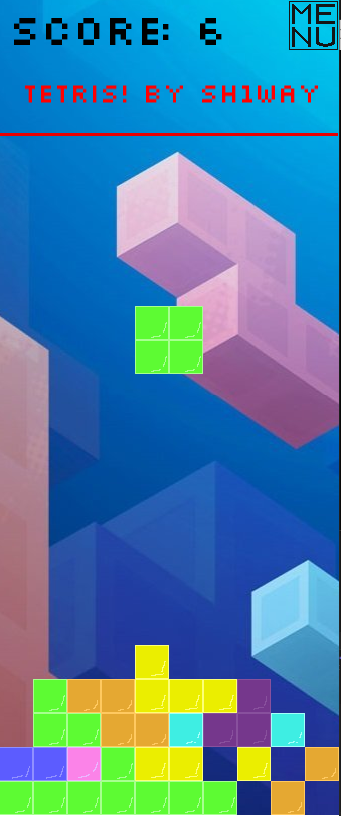


Рисунок 8 – игровое поле

Проигрыш происходит в случае, если в статичном положении находится около красной линии.

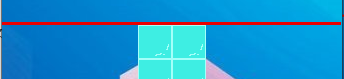


Рисунок 9 – условие проигрыша

Экран проигрыша состоит из нескольких частей:

Cообщение о проигрыше

Итоговый счет

Лучшие попытки

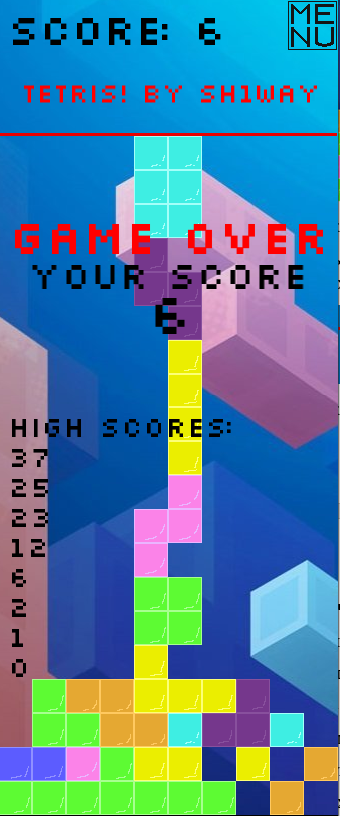


Рисунок 10 – экран проигрыша

Также в любом состоянии игры отображается кнопка «MENU». Она позволяет выйти в меню. У нее есть главная особенность она отчищает поле от поставленных блоков и сбрасывает счет. Важно отметить, если кнопку нажать на экране проигрыша, то счет будет сохранен, если он попадает в таблицу c лучшим счетом.



Рисунок 11 – кнопка меню

Заключение

Для выполнения данной курсовой работы были получены ценные навыки разработки многомодульных программ. В процессе работы была освоена работа с файлами, изучены функции работы с консолью, а также получены основные навыки отладки и тестирования программ. Программирование осуществлялось в среде Visual Studio 2022 на языках C++ и Assembler.

В рамках курсовой работы была написана программа для игры в тетрис. В дальнейшем её можно улучшить, добавив анимацию, визуальные и звуковые эффекты. Это повысит удобство использования, визуальную привлекательность и общее восприятие игры пользователями.

Данные улучшения могут включать:

Добавление плавных анимаций для падения и перемещения фигур, а также для удаления заполненных линий.

Добавление звукового сопровождения для действий в игре (например, падение фигуры, заполнение линии, конец игры).

Таким образом, разработка программы для игры в тетрис позволила не только применить на практике изученные навыки, но и заложить основу для дальнейшего улучшения проекта, ориентированного на создание более привлекательного и удобного продукта для пользователей.

Список используемых источников

1. Артур Морейра, Ян Халлер, Хенрик Фогелиус Ханссон. Разработка игр на языке SFML. — Издательство Packt, 2013 – 296р.

2. Аблязов Р. З. Программирование на ассемблере на платформе x86-64. – М.: ДМК Пресс, 2011 – 304 с.: ил.

Приложение А  
Листинг программы

**Файл main.cpp**

#include <SFML/Graphics.hpp>

#include <SFML/Window.hpp>

#include <ctime>

#include <string>

#include <vector>

#include <fstream>

#include <set>

#include "functions.h"

#include "constant.h"

Texture loadTexture(const std::string& filePath) {

Texture texture;

if (!texture.loadFromFile(filePath)) {

exit(1);

}

return texture;

}

int main() {

std::set<int> readHighScores(const std::string & filePath);

void updateHighScores(const std::string & filePath, int newScore);

srand(static\_cast<unsigned>(time(0)));

RenderWindow window(VideoMode(N \* TILE\_SIZE, M \* TILE\_SIZE), "Tetris!");

RectangleShape whiteSquare(Vector2f(N \* TILE\_SIZE, 4 \* TILE\_SIZE));

whiteSquare.setFillColor(Color(60, 60, 60));

whiteSquare.setPosition(0, 0);

Texture backgroundTexture;

if (!backgroundTexture.loadFromFile("../Paint/321.png")) {

return -1;

}

Sprite backgroundSprite(backgroundTexture);

backgroundSprite.setScale(N \* TILE\_SIZE / static\_cast<float>(backgroundTexture.getSize().x), M \* TILE\_SIZE / static\_cast<float>(backgroundTexture.getSize().y));

Texture tileTexture;

if (!tileTexture.loadFromFile("../Paint/tiles.png")) {

return -1;

}

Sprite tiles(tileTexture);

Texture imageTexture;

if (!imageTexture.loadFromFile("../Paint/321kub.png")) {

return -1;

}

Sprite imageRectangle(imageTexture);

Font font;

if (!font.loadFromFile("../Paint/demo2.ttf")) {

return -1;

}

Texture button3Texture;

if (!button3Texture.loadFromFile("../Paint/menu3.png")) {

return -1;

}

Text scoreText("", font, 45);

scoreText.setFillColor(Color::Black);

scoreText.setPosition(10, 0);

Text gameOverText("Game Over", font, 50);

gameOverText.setFillColor(Color::Red);

gameOverText.setStyle(Text::Bold);

gameOverText.setPosition((N \* TILE\_SIZE - gameOverText.getLocalBounds().width) / 2 - 5, M \* TILE\_SIZE / 4);

Text button1("Start", font, 40);

button1.setFillColor(Color::Black);

button1.setStyle(Text::Bold);

button1.setPosition((N \* TILE\_SIZE - button1.getLocalBounds().width) / 2, M \* TILE\_SIZE / 2 - 100);

Sprite button3Sprite(button3Texture);

button3Sprite.setScale(51.0f / button3Texture.getSize().x, 51.0f / button3Texture.getSize().y);

button3Sprite.setPosition(N \* TILE\_SIZE - 51, 0);

FloatRect topScoreButtonBounds;

FloatRect startButtonBounds = button1.getGlobalBounds();

int score = 0;

int dx = 0;

int colorNum = 1;

bool rotate = false;

float timer = 0, delay = 0.3;

Clock clock;

bool gameStarted = false;

bool gameOver = false;

while (window.isOpen()) {

float time = clock.getElapsedTime().asSeconds();

clock.restart();

timer += time;

handleEvents(window, gameOver, dx, rotate, gameStarted, startButtonBounds, topScoreButtonBounds, score, button3Sprite);

if (!gameOver && gameStarted) {

updateGameState(gameOver, dx, rotate, timer, delay, colorNum, score, gameStarted);

}

drawScene(window, tiles, backgroundSprite, scoreText, gameOverText, button1, imageRectangle, !gameStarted, score, gameOver, colorNum, gameStarted, button3Sprite, font);

}

return 0;

}

**Файл functions.h**

#pragma once

#include <SFML/Graphics.hpp>

#include <SFML/Window.hpp>

#include <string>

#include <vector>

using namespace sf;

void clearField();

bool checkCollision();

void spawnFigure();

bool checkTopRows();

void handleEvents(RenderWindow& window, bool& gameOver, int& dx, bool& rotate, bool& gameStarted, const FloatRect& startButtonBounds, const FloatRect& topScoreButtonBounds, int& score, Sprite& button3Sprite);

void updateGameState(bool& gameOver, int& dx, bool& rotate, float& timer, float delay, int& colorNum, int& score, bool gameStarted);

void drawScene(RenderWindow& window, Sprite& tiles, Sprite& backgroundSprite, Text& scoreText, Text& gameOverText, Text& button1, Sprite& imageRectangle, bool startVisible, int score, bool gameOver, int colorNum, bool gameStarted, Sprite& button3Sprite, Font& font);

bool isButtonPressed(RenderWindow& window, Text& button);

void restartGame(bool& gameStarted, bool& gameOver, int& score);

std::set<int> readHighScores(const std::string& filePath);

void writeHighScores(const std::string& filePath, const std::set<int>& scores);

void updateHighScores(const std::string& filePath, int newScore);

**Файл functions.cpp**

#include <SFML/Graphics.hpp>

#include <SFML/Window.hpp>

#include <ctime>

#include <string>

#include <vector>

#include <fstream>

#include <set>

#include "functions.h"

#include "constant.h"

void clearField() {

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

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

field[i][j] = 0;

}

}

}

void drawImageRectangle(RenderWindow& window, Sprite& sprite, const Vector2f& position, const Vector2f& size) {

sprite.setPosition(position);

sprite.setScale(size.x / sprite.getLocalBounds().width, size.y / sprite.getLocalBounds().height);

window.draw(sprite);

}

bool checkCollision() {

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

if (a[i].x < 0 || a[i].x >= N || a[i].y >= M || field[a[i].y][a[i].x]) {

return false;

}

}

return true;

}

void spawnFigure() {

int n = rand() % 7;

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

a[i].x = figures[n][i] % 2 + N / 2 - 1;

a[i].y = figures[n][i] / 2 + 1;

}

}

bool checkTopRows() {

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

if (field[4][j] != 0) {

return true;

}

}

return false;

}

void handleEvents(RenderWindow& window, bool& gameOver, int& dx, bool& rotate, bool& gameStarted, const FloatRect& startButtonBounds, const FloatRect& topScoreButtonBounds, int& score, Sprite& button3Sprite) {

Event event;

while (window.pollEvent(event)) {

if (event.type == Event::Closed)

window.close();

if (event.type == Event::Closed) {

window.close();

}

if (event.type == Event::KeyPressed && !gameOver) {

if (event.key.code == Keyboard::Up) {

rotate = true;

}

else if (event.key.code == Keyboard::Right) {

dx = 1;

}

else if (event.key.code == Keyboard::Left) {

dx = -1;

}

}

if (!gameStarted && event.type == Event::MouseButtonPressed && event.mouseButton.button == Mouse::Left) {

Vector2i mousePos = Mouse::getPosition(window);

if (startButtonBounds.contains(static\_cast<float>(mousePos.x), static\_cast<float>(mousePos.y))) {

gameStarted = true;

spawnFigure();

}

if (topScoreButtonBounds.contains(static\_cast<float>(mousePos.x), static\_cast<float>(mousePos.y))) {

gameStarted = false;

gameOver = true;

}

}

if (event.type == Event::MouseButtonPressed && event.mouseButton.button == Mouse::Left) {

Vector2i mousePos = Mouse::getPosition(window);

if (button3Sprite.getGlobalBounds().contains(static\_cast<float>(mousePos.x), static\_cast<float>(mousePos.y))) {

gameStarted = false;

gameOver = false;

score = 0;

clearField();

spawnFigure();

}

}

}

}

void updateGameState(bool& gameOver, int& dx, bool& rotate, float& timer, float delay, int& colorNum, int& score, bool gameStarted) {

if (!gameStarted) {

return;

}

if (Keyboard::isKeyPressed(Keyboard::Down)) {

delay = 0.05;

}

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

b[i] = a[i];

a[i].x += dx;

}

if (!checkCollision()) {

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 (!checkCollision()) {

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

a[i] = b[i];

}

}

}

if (timer > delay) {

\_\_asm {

push eax

push ecx

push edi

push esi

mov ecx, 0

lea esi, a

lea edi, b

loop\_start :

mov eax, [esi + ecx \* 8]

mov[edi + ecx \* 8], eax

mov eax, [esi + ecx \* 8 + 4]

mov[edi + ecx \* 8 + 4], eax

add eax, 1

mov[esi + ecx \* 8 + 4], eax

inc ecx

cmp ecx, 4

jl loop\_start

pop esi

pop edi

pop ecx

pop eax

}

if (!checkCollision()) {

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

field[b[i].y][b[i].x] = colorNum;

}

colorNum = 1 + rand() % 7;

spawnFigure();

if (!checkCollision() || checkTopRows()) {

gameOver = true;

updateHighScores("scores.txt", score);

}

}

timer = 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 (count == N) {

score += 1;

}

}

dx = 0;

rotate = false;

}

void drawScene(RenderWindow& window, Sprite& tiles, Sprite& backgroundSprite, Text& scoreText, Text& gameOverText, Text& button1, Sprite& imageRectangle, bool startVisible, int score, bool gameOver, int colorNum, bool gameStarted, Sprite& button3Sprite, Font& font) {

window.clear(Color::Black);

window.draw(backgroundSprite);

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

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

if (field[i][j] == 0) {

continue;

}

tiles.setTextureRect(IntRect(field[i][j] \* TILE\_SIZE, 0, TILE\_SIZE, TILE\_SIZE));

tiles.setPosition(j \* TILE\_SIZE, i \* TILE\_SIZE);

window.draw(tiles);

}

}

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

if (!gameOver && gameStarted) {

tiles.setTextureRect(IntRect(colorNum \* TILE\_SIZE, 0, TILE\_SIZE, TILE\_SIZE));

tiles.setPosition(a[i].x \* TILE\_SIZE, a[i].y \* TILE\_SIZE);

window.draw(tiles);

}

}

drawImageRectangle(window, imageRectangle, Vector2f(0, 0), Vector2f(N \* TILE\_SIZE - 1, 4 \* TILE\_SIZE));

if (gameOver) {

window.draw(gameOverText);

Text yourScoreText("Your score", font, 40);

yourScoreText.setFillColor(Color::Black);

yourScoreText.setPosition((N \* TILE\_SIZE - yourScoreText.getLocalBounds().width) / 2 - 5, M \* TILE\_SIZE / 4 + 45);

window.draw(yourScoreText);

Text scoreNumberText(std::to\_string(score), font, 60);

scoreNumberText.setFillColor(Color::Black);

scoreNumberText.setPosition((N \* TILE\_SIZE - scoreNumberText.getLocalBounds().width) / 2 - 5, M \* TILE\_SIZE / 4 + 70);

window.draw(scoreNumberText);

std::set<int> highScores = readHighScores("scores.txt");

Text highScoresText("High Scores:", font, 30);

highScoresText.setFillColor(Color::Black);

highScoresText.setPosition(10, M \* TILE\_SIZE / 2);

window.draw(highScoresText);

int y\_offset = 30;

for (auto it = highScores.rbegin(); it != highScores.rend(); ++it) {

Text scoreText(std::to\_string(\*it), font, 30);

scoreText.setFillColor(Color::Black);

scoreText.setPosition(10, M \* TILE\_SIZE / 2 + y\_offset);

window.draw(scoreText);

y\_offset += 30;

}

}

scoreText.setString("Score: " + std::to\_string(score));

window.draw(scoreText);

Text ByMeText("Tetris! by sh1way", font, 28);

ByMeText.setFillColor(Color::Red);

FloatRect rect = imageRectangle.getGlobalBounds();

ByMeText.setPosition(rect.left + (rect.width - ByMeText.getLocalBounds().width) / 2, rect.top + (rect.height - ByMeText.getLocalBounds().height) / 2 + 15);

window.draw(ByMeText);

if (startVisible) {

window.draw(button1);

}

window.draw(button3Sprite);

window.display();

}

bool isButtonPressed(RenderWindow& window, Text& button) {

Vector2i mousePos = Mouse::getPosition(window);

if (Mouse::isButtonPressed(Mouse::Left)) {

FloatRect bounds = button.getGlobalBounds();

if (bounds.contains(static\_cast<float>(mousePos.x), static\_cast<float>(mousePos.y))) {

return true;

}

}

return false;

}

void restartGame(bool& gameStarted, bool& gameOver, int& score) {

gameStarted = false;

gameOver = false;

score = 0;

clearField();

spawnFigure();

}

std::set<int> readHighScores(const std::string& filePath) {

std::set<int> scores;

std::ifstream file(filePath);

if (file.is\_open()) {

int score;

while (file >> score) {

scores.insert(score);

}

file.close();

}

return scores;

}

void writeHighScores(const std::string& filePath, const std::set<int>& scores) {

std::ofstream file(filePath);

if (file.is\_open()) {

for (auto it = scores.rbegin(); it != scores.rend(); ++it) {

file << \*it << std::endl;

}

file.close();

}

}

void updateHighScores(const std::string& filePath, int newScore) {

std::set<int> scores = readHighScores(filePath);

scores.insert(newScore);

writeHighScores(filePath, scores);

}

**Файл constant.h**

#pragma once

extern const int M;

extern const int N;

extern const int TILE\_SIZE;

extern int field[24][10];

struct Point {

int x, y;

};

extern int figures[7][4];

extern Point a[4], b[4];

**Файл constant.cpp**

#include "constant.h"

const int M = 24;

const int N = 10;

const int TILE\_SIZE = 34;

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

int figures[7][4] = {

1, 3, 5, 7,

2, 4, 5, 7,

3, 5, 4, 6,

3, 5, 4, 7,

2, 3, 5, 7,

3, 5, 7, 6,

2, 3, 4, 5

};

Point a[4], b[4];