Министерство цифрового развития, связи и массовых телекоммуникаций Российской Федерации

«Сибирский государственный университет телекоммуникаций и информатики»

Кафедра прикладной математики и кибернетики

КУРСОВАЯ РАБОТА

по дисциплине «Объектно-ориентированное программирование»

Тема: «Тетрис»

Выполнил: студент 2 курса группы ИП-311 Николаев М.Ю

Проверил: старший преподаватель кафедры ПМиК

Дементьева Кристина Игоревна

Оглавление

[1. Постановка задачи 3](#_Toc30598306)

[2. Технологии ООП 3](#_Технологии_ООП)

[2.1 Задание в соответсвии с вариантом………………………………………….3](#_Задание_в_соответствии)

[3. Структура классов…………………………………………………………………...4](#_Структура_классов)

[4. Описание алгоритма основной программы…………………………………..........7](#_Описание_алгоритма_основной)

[5. Примеры работы программы…………………………………………………..........8](#_Примеры_работы_программы)

[6. Заключение………………….………………………………………………............10](#_Заключение)

[7. Используемые источники………………………………………………….............11](#_Используемые_источники)

[*Приложение* Листинг …………………………………………………......................12](#_Приложение_Листинг)

# Постановка задачи

Программа должна быть написана с использованием **объектно-ориентированных технологий**. **Описания** объектов и методов **необходимо оформить в отдельном модуле**.

# Технологии ООП

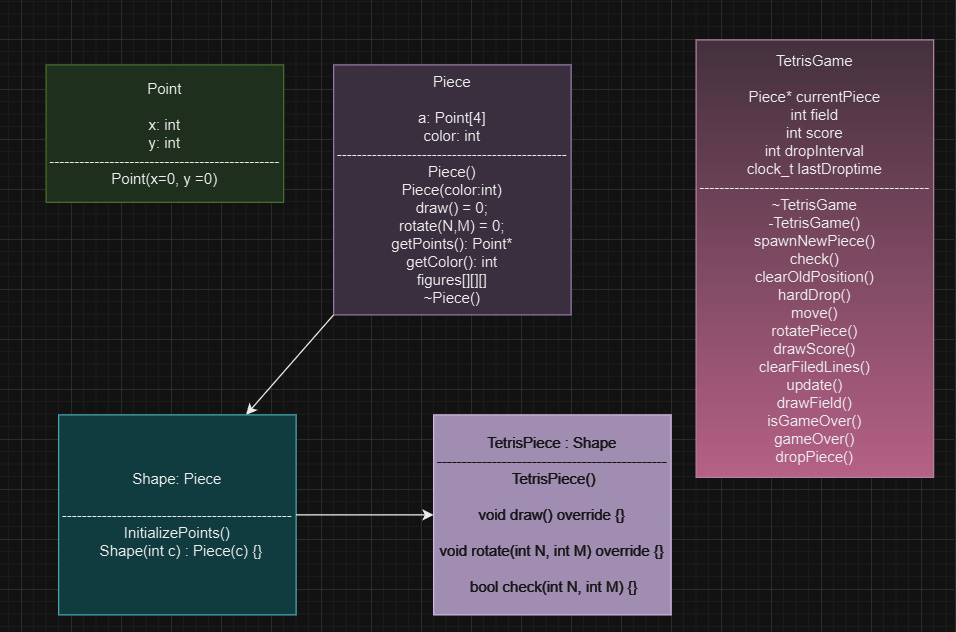
Работа содержит такие технологии ООП, как:

* Инкапсуляция (все поля данных не доступны из внешних функций)
* Наследование (реализовано 3 класса, один из которых - абстрактный)
* Полиморфизм
* Конструкторы, Перегрузка конструкторов
* Списки инициализации

## Задание в соответствии с вариантом

Реализовать игру «Тетрис» (графический режим)

# Структура классов



Так выглядит иерархия классов в проекте.

**Класс Point** - класс, представляющий точку на игровом поле с координатами x и y.

**Public**

**Поля:**

int x – координата по оси X.

int y – координата по оси Y.

**Методы:**

Point(int x = 0, int y = 0) – конструктор, инициализирующий координаты точки.

**Класс Piece** - абстрактный класс, представляющий тетромино (фигуру) в игре. Служит базовым классом для различных типов фигур.

**Protected**

**Поля:**

Point a[4] – массив из 4х точкек, представляющий координаты фигур.

int color – цвет фигуры.

**Public**

**Методы:**

Piece() – конструктор, инициализирующий фигуру.

Piece(int colorValue) – конструктор, задающий цвет фигуры.

virtual void draw() = 0 – виртуальный метод для отрисовки фигуры.

virtual void rotate(int N, int M) = 0 – виртуальный метод для поворота фигуры.

Point\* getPoints() – метод, возвращающий массив точек.

int getColor() – метод, возвращающий цвет фигуры.

virtual ~Piece() – деструктор.

**Класс Shape : public Piece** - это производный класс от Piece, который отвечает за инициализацию точек для конкретной фигуры.

**Public Методы:**

Shape(int c) – конструктор, который вызывает конструктор базового класса Piece.

**Protected Методы:**

void initializePoints(const int figures[7][4][2], int figureType) – инициализирует точки в зависимости от типа фигуры.

**Класс TetrisPiece : public Shape** - это класс, представляющий конкретные фигуры тетриса, реализующий методы отрисовки и вращения.

**Public Методы:**

TetrisPiece(int figureType, int c) – конструктор, который инициализирует точки фигуры на основе типа.

void draw() override – реализует отрисовку фигуры на экране.

void rotate(int N, int M) override – реализует поворот фигуры с учетом границ игрового поля.

bool check(int N, int M) – проверяет, подходит ли фигура после вращения внутри границ поля.

**Класс TetrisGame** - класс, отвечающий за логику игры Тетрис, включая управление игровыми фигурами и подсчет очков.

**Private Поля:**

Piece\* currentPiece – указатель на текущую фигуру.

int field[M][N] – игровое поле, представляющее состояние ячеек.

static int totalScore – общий счет игрока.

int dropInterval – интервал между падениями фигур.

clock\_t lastDropTime – время последнего падения фигуры.

**Public Методы:**

static void resetScore() – сбрасывает счет.

static int getScore() – возвращает текущее количество очков.

TetrisGame() – конструктор, инициализирующий игровое поле и создающий новую фигуру.

~TetrisGame() – деструктор, очищающий память.

void spawnNewPiece() – генерирует новую фигуру.

bool check() – проверяет допустимость текущего положения фигуры.

void clearOldPosition() – очищает предыдущую позицию фигуры на экране.

void hardDrop() – заставляет фигуру падать до основания или до столкновения.

void move(int dx) – перемещает фигуру по оси X.

void rotatePiece() – поворачивает фигуру.

void drawScore() – отображает текущий счет на экране.

void clearFilledLines() – очищает полные линии и обновляет счет.

void update() – обновляет игровое поле и отображает фигуру.

void drawField() – отрисовывает игровое поле.

bool isGameOver() – проверяет, завершена ли игра.

void gameOver() – обрабатывает состояние окончания игры.

void dropPiece() – падает фигура с интервалами.

# Описание алгоритма основной программы

Запускается функция initgraph(), которая открывает графическое окно. Создается объект TetrisGame, который инициализирует игровое поле, текущую фигуру и счет. Игра начинает с задания случайного типа фигуры и цвета. В программе используется бесконечный цикл, который отвечает за выполнение игры до тех пор, пока игрок не решит выйти или игра не завершится. Программа содержит проверку состояния клавиш, которое позволяет пользователю управлять падающей фигурой.

VK\_ESCAPE — выход из игры; VK\_LEFT — движение фигуры влево; VK\_RIGHT — движение вправо; VK\_UP — поворот фигуры; VK\_DOWN — резкое падение фигуры.

Когда мы достигли вершины игрового поля, вызывается метод isGameOver(). Если игра завершена, выводится экране сообщение о завершении, и цикл прерывается. В конце каждого падения фигуры вызывается метод update(), который очищает прежнее положение фигуры и обновляет экран, отрисовывая: игровое поле; текущую фигуру; значение счета.

# Примеры работы программы

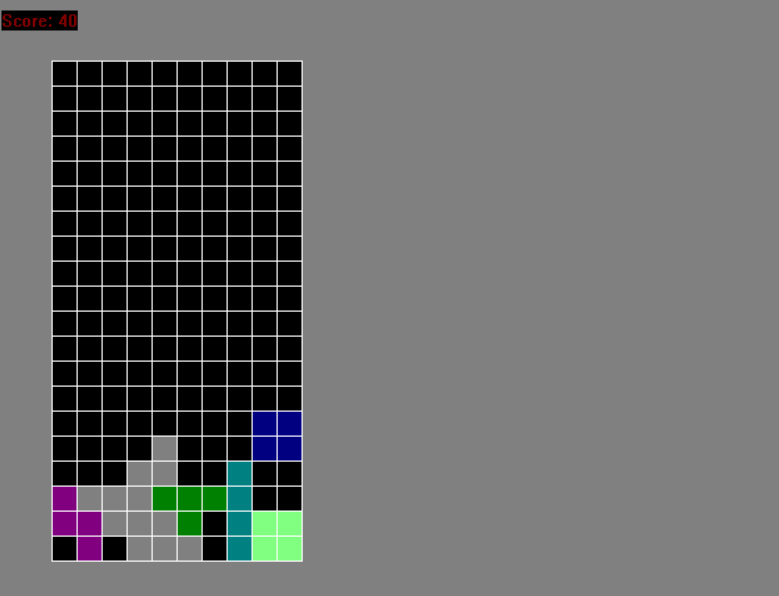


Рисунок 1 – Падение фигуры

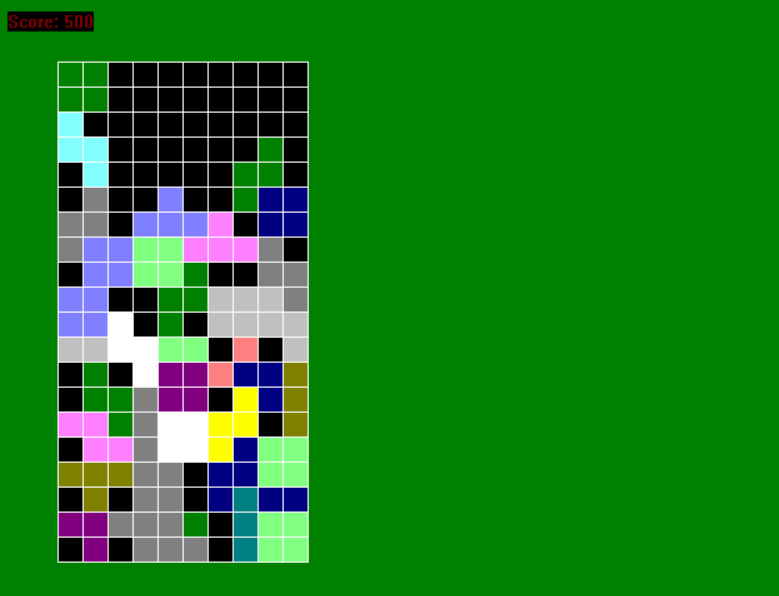


Рисунок 2 – Заполнение игрового поля



Рисунок 3 – Конец игры, вызванный после заполнения всей области игрового поля. Финальное количество очков.

# Заключение

Благодаря курсовому проекту я на практике применил такую методологию программирования, как объектно-ориентированное программирование, а также закрепил знания, полученные в ходе данного курса.

# Используемые источники

1. Тетрис: история и значение в компьютерных играх [Электронный ресурс] // Wikipedia. URL: https://ru.wikipedia.org/wiki/Tetris (дата обращения: 25.11.24)
2. C++ и графика: основы работы с библиотекой graphics.h [Электронный ресурс] // DevDocs. URL: https://devdocs.io/cpp/library/graphics.h (дата обращения: 24.11.24)
3. Основы объектно-ориентированного программирования на C++ [Электронный ресурс] // CPlusPlus.com. URL: https://www.cplusplus.com/doc/tutorial/classes/ (дата обращения: 27.11.24)
4. Конструкторы и инициализация объектов [Электронный ресурс] // METANIT.com. URL: https://metanit.com/cpp/tutorial/5.2.php (дата обращения: 28.11.24)
5. Конструкторы и инициализация объектов [Электронный ресурс] // METANIT.com. URL: https://metanit.com/cpp/tutorial/5.2.php (дата обращения: 28.11.24)
6. Списки инициализации в C++: хороший, плохой, злой [Электронный ресурс] // habr.com. URL: https://habr.com/ru/articles/330402/ (дата обращения: 30.11.24)
7. Виртуальные функции и их переопределение [Электронный ресурс] // METANIT.com. URL: https://metanit.com/cpp/tutorial/5.11.php (дата обращения: 1.12.24)

# *Приложение* Листинг

**main.сpp**

#include <graphics.h>

#include <conio.h>

#include <cstdlib>

#include <ctime>

#include "TetrisGame.h"

int main() {

int gd = DETECT, gm;

initgraph(&gd, &gm, "C:\\MinGW\\lib\\libbgi.a");

srand(time(NULL));

TetrisGame game;

while (true) {

delay(100);

if (GetAsyncKeyState(VK\_ESCAPE)) break;

if (game.isGameOver()) {

game.gameOver();

break;

}

if (GetAsyncKeyState(VK\_LEFT)) game.move(-1);

if (GetAsyncKeyState(VK\_RIGHT)) game.move(1);

if (GetAsyncKeyState(VK\_UP)) game.rotatePiece();

if (GetAsyncKeyState(VK\_DOWN)) game.hardDrop();

game.dropPiece();

game.update();

}

closegraph();

return 0;}

**Piece.h**

#ifndef PIECE\_H

#define PIECE\_H

#include <graphics.h>

#include <iostream>

#include <cstring>

class Point {

public:

int x, y;

Point(int x = 0, int y = 0) : x(x), y(y) {}

};

class Piece {

protected:

Point a[4];

int color;

public:

Piece() : color(0) {

memset(a, 0, sizeof(a));}

Piece(int colorValue) : color(colorValue) {

memset(a, 0, sizeof(a));

}

virtual void draw() = 0;

virtual void rotate(int N, int M) = 0;

Point\* getPoints() { return a; }

int getColor() { return color; }

virtual ~Piece() {}

static const int figures[7][4][2];

};

const int Piece::figures[7][4][2] = {

{{0, 0}, {1, 0}, {2, 0}, {3, 0}},

{{0, 0}, {0, 1}, {1, 1}, {1, 2}},

{{0, 1}, {0, 0}, {1, 0}, {1, -1}},

{{0, 0}, {1, 0}, {2, 0}, {1, 1}},

{{0, 0}, {1, 0}, {1, 1}, {2, 1}},

{{1, 0}, {0, 0}, {0, 1}, {1, 1}},

{{0, 0}, {0, 1}, {1, 0}, {1, 1}}

};

#endif

**TetrisPiece.h**

#ifndef TETRISPIECE\_H

#define TETRISPIECE\_H

#include "Shape.h"

#include <cstdlib>

class TetrisPiece : public Shape {

public:

TetrisPiece(int figureType, int c) : Shape(c) {

initializePoints(Piece::figures, figureType);

}

void draw() override {

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

int x = a[i].x \* 20 + 50;

int y = a[i].y \* 20 + 50;

setfillstyle(1, color);

floodfill(x + 1, y + 1, WHITE);

}

}

void rotate(int N, int M) override {

Point center = a[1];

Point temp[4];

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

int x = a[i].y - center.y;

int y = a[i].x - center.x;

temp[i] = Point(center.x - x, center.y + y);

}

Point old[4];

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

old[i] = a[i];

}

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

a[i] = temp[i];

}

if (!check(N, M)) {

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

a[i] = old[i];

}

}

}

bool check(int N, int M) {

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

int x = a[i].x;

int y = a[i].y;

if (x < 0 || x >= N || y < 0 || y >= M) {

return false;

}

}

return true;

}

};

#endif

**TetrisGame.h**

#ifndef TETRISGAME\_H

#define TETRISGAME\_H

#include "TetrisPiece.h"

#include <ctime>

const int M = 20;

const int N = 10;

class TetrisGame {

private:

Piece\* currentPiece;

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

static int totalScore;

int dropInterval;

clock\_t lastDropTime;

public:

static void resetScore() { totalScore = 0; }

static int getScore() { return totalScore; }

TetrisGame() : currentPiece(NULL), dropInterval(700), lastDropTime(clock()) {

memset(field, 0, sizeof(field));

resetScore();

spawnNewPiece();

}

~TetrisGame() {

delete currentPiece;

}

void spawnNewPiece() {

delete currentPiece;

int figureType = rand() % 7;

int colorNum = 1 + rand() % 15;

currentPiece = new TetrisPiece(figureType, colorNum);

}

bool check() {

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

int x = currentPiece->getPoints()[i].x;

int y = currentPiece->getPoints()[i].y;

if (x < 0 || x >= N || y < 0 || y >= M || (y >= 0 && field[y][x] != 0)) {

return false;

}

}

return true;

}

void clearOldPosition() {

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

int x = currentPiece->getPoints()[i].x;

int y = currentPiece->getPoints()[i].y;

if (y >= 0 && y < M) {

setcolor(BLACK);

setfillstyle(SOLID\_FILL, BLACK);

bar(x \* 20 + 50, y \* 20 + 50, x \* 20 + 70, y \* 20 + 70);

}

}

}

void hardDrop() {

while (true) {

clearOldPosition();

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

currentPiece->getPoints()[i].y++;

}

if (!check()) {

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

currentPiece->getPoints()[i].y--;

}

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

int x = currentPiece->getPoints()[i].x;

int y = currentPiece->getPoints()[i].y;

if (y >= 0) {

field[y][x] = currentPiece->getColor();

}

}

clearFilledLines();

spawnNewPiece();

break;

}

delay(10);

}

totalScore += 10;

spawnNewPiece();

}

void move(int dx) {

clearOldPosition();

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

currentPiece->getPoints()[i].x += dx;

}

if (!check()) {

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

currentPiece->getPoints()[i].x -= dx;

}

}

update();

}

void rotatePiece() {

clearOldPosition();

currentPiece->rotate(N, M);

if (!check()) {

for (int i = 0; i < 3; i++) {

currentPiece->rotate(N, M);

}

}

update();

}

void drawScore() {

setcolor(RED);

char scoreText[30];

sprintf(scoreText, "Score: %d", totalScore);

printf("%d\n", totalScore);

outtextxy(10, 10, scoreText);

}

void clearFilledLines() {

for (int i = M - 1; i >= 0; i--) {

bool filled = true;

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

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

filled = false;

break;

}

}

if (filled) {

totalScore += 100;

for (int row = i; row > 0; row--) {

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

field[row][col] = field[row - 1][col];

}

}

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

field[0][col] = 0;

}

i++;

}

}

}

void update() {

clearOldPosition();

drawField();

currentPiece->draw();

drawScore();

}

void drawField() {

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

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

setcolor(WHITE);

setfillstyle(1, field[i][j]);

rectangle(j \* 20 + 50, i \* 20 + 50, j \* 20 + 70, i \* 20 + 70);

floodfill(j \* 20 + 51, i \* 20 + 51, WHITE);

}

}

}

bool isGameOver() {

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

if (field[0][i] != 0) return true;

}

return false;

}

void gameOver() {

cleardevice();

setcolor(RED);

settextstyle(DEFAULT\_FONT, HORIZ\_DIR, 3);

outtextxy(100, 200, "GAME OVER!");

char scoreText[30];

sprintf(scoreText, "Final Score: %d", totalScore);

outtextxy(100, 250, scoreText);

delay(5000);

}

void dropPiece() {

if (clock() - lastDropTime >= dropInterval) {

clearOldPosition();

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

currentPiece->getPoints()[i].y++;

}

if (!check()) {

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

currentPiece->getPoints()[i].y--;

}

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

int x = currentPiece->getPoints()[i].x;

int y = currentPiece->getPoints()[i].y;

if (y >= 0) {

field[y][x] = currentPiece->getColor();

}

}

clearFilledLines();

spawnNewPiece();

}

update();

lastDropTime = clock();

}

}

};

int TetrisGame::totalScore = 0;

#endif

**Shape.h**

#ifndef SHAPE\_H

#define SHAPE\_H

#include "Piece.h"

class Shape : public Piece {

protected:

void initializePoints(const int figures[7][4][2], int figureType) {

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

a[i] = Point(figures[figureType][i][0], figures[figureType][i][1]);

}

}

public:

Shape(int c) : Piece(c) {}

};

#endif