Министерство образования и науки Российской Федерации

Федеральное государственное автономное образовательное учреждение

высшего образования

«Южно-Уральский государственный университет»

(национальный исследовательский университет)

Институт естественных и точных наук

Факультет математики, механики и компьютерных технологий

Кафедра прикладной математики и программирования

Разработка Windows-приложения - игра «Пятнашки»

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОЙ РАБОТЕ

по дисциплине «Операционные системы»

ЮУрГУ–09.03.04.2017.38.ПЗ КР

|  |  |
| --- | --- |
|  | Руководитель работы,  \_\_\_\_\_\_\_\_\_\_\_\_\_/ Е.Ю. Алексеева  «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_2017 г. |
|  | Автор работы  Студент группы ЕТ-414  \_\_\_\_\_\_\_\_\_\_\_\_\_/ А.В. Юферов  «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_2017 г. |
|  | Нормоконтролер,  \_\_\_\_\_\_\_\_\_\_\_\_\_/\_\_\_\_\_\_\_\_\_\_\_  «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_2017 г. |

Челябинск 2017

**Аннотация**

|  |
| --- |
| Юферов А.В. Разработка Windows-приложения - игры «Пятнашки» Челябинск: ЮУрГУ, ЕТ-414, 34 с., 5 ил.,  1 прил. |

Исследованы существующие решения данной задачи. В работе разработано Windows-приложение – игра «Пятнашки» на языке C++ с использованием функций библиотек OpenGL, FreeGLUT и SOIL и среды разработки Microsoft Visual Studio 2015.

**Оглавление**

[Введение 4](#_Toc501047805)

[1 Постановка задачи 5](#_Toc501047806)

[1.1 Определение 5](#_Toc501047807)

[1.2 Описание приложения 5](#_Toc501047808)

[2 Математическая модель 6](#_Toc501047809)

[3 Описание алгоритма решения 7](#_Toc501047810)

[3.1 Основной алгоритм 7](#_Toc501047811)

[3.2 Вспомогательный алгоритм обработки сообщений 8](#_Toc501047812)

[3.3 Описание функций: назначение, входные и выходные данные 9](#_Toc501047813)

[Инструкция по применению разработанной программы 14](#_Toc501047814)

[Заключение 15](#_Toc501047815)

[Литература 16](#_Toc501047816)

[Приложение 1: Исходный код программы 17](#_Toc501047817)

# Введение

Необходимо разработать компьютерную программу на языке программирования C++, реализующую игру «Пятнашки». При разработке программы необходимо использовать только функции библиотек OpenGL, FreeGLUT и SOIL.

Среда разработки: Microsoft Visual Studio 2015.

# Постановка задачи

Необходимо разработать компьютерную программу на языке программирования C++, реализующую игру «Пятнашки». При разработке программы необходимо использовать только функции библиотек OpenGL, FreeGLUT и SOIL.

Среда разработки: Microsoft Visual Studio 2015.

## Определение

**Игра в 15, пятнашки, такен** — популярная головоломка, придуманная в 1878 году Ноем Чепмэном. Представляет собой набор одинаковых квадратных костяшек с нанесёнными числами, заключённых в квадратную коробку. Длина стороны коробки в четыре раза больше длины стороны костяшек для набора из 15 элементов, соответственно в коробке остаётся незаполненным одно квадратное поле. Цель игры — перемещая костяшки по коробке, добиться упорядочивания их по номерам, желательно сделав как можно меньше перемещений. Общие требования

## Описание приложения

Приложение должно состоять из одного окна, созданного с помощью методов библиотеки FreeGLUT. Всё пространство окна занимает контекст OpenGL. Отрисовка элементов игры должна осуществляться с помощью методов библиотеки OpenGL. Элементы игры должны быть текстурированны, используя методы библиотек OpenGL и SOIL. Текстуры подгружаются с диска из папки рядом с исполняемым файлом. Используя методы FreeGLUT, должно быть создано контекстное меню со следующими пунктами и подпунктами:

* «Game» – Основное меню
  + «New game» – Начать новую игру
  + «Exit» – Выйти из приложения
* «Texture» – В этом подменю можно осуществить выбор текстуры основных элементов игры
* «About» – Открыть окно дополнительных сведений о программе

# Математическая модель

Игровое поле является квадратной матрицей порядка 4 и может быть представлено в виде 15 кубов (рис 1). Значения в матрице соответсвуют цифрам, написанным на костях, а пустая ячейка обозначается нулем.

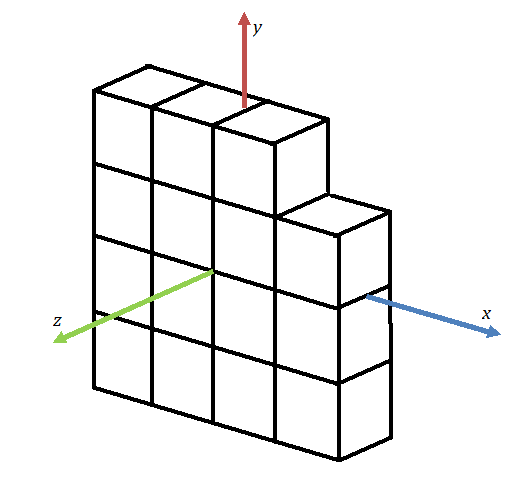


Рис 1. Поле

Игровое поле можно вращать вокруг оси абсцисс и оси ординат. Для этого используются матрицы вращения.

Матрицы вращения вокруг оси абсцисс на угол α и вращения вокруг оси ординат на угол β:

# Описание алгоритма решения

## Основной алгоритм

На рис. 2 представлен основной алгоритм программы, стандартный для Windows-приложений.



Рисунок 2. Основной алгоритм

## Вспомогательный алгоритм обработки сообщений

Вспомогательный алгоритм обработки сообщений от операционной системе представлен на рис. 3.



Рисунок 3. Вспомогательный алгоритм обработки сообщений

## Описание функций: назначение, входные и выходные данные

/\*\*

\* @class Приложение

\*

\* Этот класс управляет основными ресурсами приложения,

\* такими как основной цикл приложения, окно и т.д.

\* Возможно существование только одного экземпляра этого

\* класса в программе.

\*/

class Application

/\*\*

\* @brief Этот метод инициализирует окно GLUT,

\* открывая его по центру экрана.

\* Ширина и высота устанавливаются равними

\* 640 на 480 пикселей соответственно.

\*/

void show(const char \*windowTitle)

/\*\*

\* @brief Этот метод инициализирует окно GLUT,

\* открывая его в полноэкранном режиме.

\*/

void showFullScreen(const char \*windowTitle)

/\*\*

\* @brief Этот метод запускает основной цикл библиотеки GLUT.

\*/

void exec()

/\*\*

\* @brief Этот метод устанавливает функцию обратного

\* вызова на событие перерисовки экрана.

\*/

void setDisplayFunction(DisplayCallback display)

/\*\*

\* @brief Этот метод устанавливает функцию обратного

\* вызова на событие нажатия на клавишу клавиатуры.

\*/

void setKeyBoardFunction(KeyboardCallback keyboard)

/\*\*

\* @brief Этот метод устанавливает функцию обратного

\* вызова на событие нажатия на специальные клавиши клавиатуры.

\*/

void setSpecialFunction(SpecialCallback special)

/\*\*

\* @brief Этот метод добавляет в очереть событие перерисовки окна.

\*/

void repaint()

/\*\*

\* @brief Этот метод устанавливает меню

\*/

void setMenu(Menu &&menu)

/\*\*

\* @brief Этот метод посылает сигнал завершения приложения

\*/

void exit()

/\*\*

\* @brief Этот метод запускает обработку накопившихся сообщений

\*/

void processEvents()

/\*\*

\* @brief Доска

\*

\* Игровая доска с фишками

\*/

class Board

/\*\*

\* @brief Доступ к фишкам (0 - пустая клетка доски)

\*/

auto tiles() const

/\*\*

\* @brief Предикат проверяющий возможность сдвига вправо

\*/

bool canMoveRight() const

/\*\*

\* @brief Предикат проверяющий возможность сдвига влево

\*/

bool canMoveLeft() const

/\*\*

\* @brief Предикат проверяющий возможность сдвига вверх

\*/

bool canMoveUp() const

/\*\*

\* @brief Предикат проверяющий возможность сдвига вниз

\*/

bool canMoveDown() const

/\*\*

\* @brief Сдвинуть фишки вправо

\*/

void moveRight()

/\*\*

\* @brief Сдвинуть фишки влево

\*/

void moveLeft()

/\*\*

\* @brief Сдвинуть фишки вниз

\*/

void moveDown()

/\*\*

\* @brief Сдвинуть фишки вверх

\*/

void moveUp()

/\*\*

\* @brief Начальное положение фишек на доске

\*/

void init()

/\*\*

\* @brief Тасовка фишек

\*/

void shuffle()

/\*\*

\* @brief Сериализация

\*/

void serialize(std::ostream &os)

/\*\*

\* @brief Десериализация

\*/

void deserialize(std::istream &is)

/\*\*

\* @class Игра

\*

\* Основная логика игры пятнашки.

\*/

class Game

/\*\*

\* @brief Предикат окончания игры

\*/

bool isOver() const

/\*\*

\* @brief Метод возвращает доску игры в текущем состоянии

\*/

auto board() const

/\*\*

\* @brief Этот метод регистрирует наблюдателя.

\* Метод не перехватывает владение объектом

\*/

void addObserver(GameObserver \*obs)

/\*\*

\* @brief Этот метод создает новую игру.

\*/

void newGame()

/\*\*

\* @brief Сдвиг фишек вверх.

\*/

void moveUp()

/\*\*

\* @brief Сдвиг фишек вниз.

\*/

void moveDown()

/\*\*

\* @brief Сдвиг фишек вправо.

\*/

void moveRight()

/\*\*

\* @brief Сдвиг фишек влево.

\*/

void moveLeft()

/\*\*

\* @brief Сохранить игру

\*/

void save()

/\*\*

\* @brief Загрузить сохраненную игру

\*/

void load()

/\*\*

\* @class Базовый класс наблюдателя за игрой

\*/

class GameObserver

/\*\*

\* @brief Игра изменила свое состояние

\*/

virtual void onGameChanged() = 0

/\*\*

\* @brief Меню

\*

\* Класс-оболочка над вспомогательным контекстным меню из

\* библиотеки GLUT. Из-за сервисной архитектуры GLUT

\* работа с объектами данного класса должна происходить

\* поочередно, т.е. при создании нового экземпляра, предыдущий

\* экземпляр становится объектом только для чтения. При попытке

\* добавить подменю или вхождение в старые объекты будет брошено

\* исключение.

\*/

class Menu

/\*\*

\* @brief Идентификатор меню, полученый от GLUT

\*/

int id() const

/\*\*

\* @brief Добавить вхождение

\*/

Menu &addMenuEntry(std::string title, MenuCallback callback)

/\*\*

\* @brief Добавить подменю

\*/

Menu &addSubMenu(std::string title, Menu &&menu)

/\*\*

\* @brief Отрисовщик

\*

\* Этот класс отрисовывает игру на экране

\*/

class Renderer

/\*\*

\* @brief Этот метод перерисовывает контекст

\*/

void display()

/\*\*

\* @brief Этот метод будет вызван, при изменениях в игре

\*/

void onGameChanged() override

/\*\*

\* @brief Этот метод осуществляет поворот камеры относительно оси X

\*/

void rotareCameraX(double angle)

/\*\*

\* @brief Этот метод осуществляет поворот камеры относительно оси Y

\*/

void rotareCameraY(double angle)

/\*\*

\* @brief Этот метод устанавливает текстуру для клеток

\*/

void setTexturePath(fs::path texture)

# Инструкция по применению разработанной программы

Запустить app.exe из папки bin в корневой директории проекта. Автоматически начнется новая игра. Если приложение запусколось ранее, тогда будет запущена последняя недоигранная партия.

Щелкните правой кнопкой мыши внутри окна для открытия контекстного меню игры. Основые пункты меню:

* «Game» – Основное меню
  + «New game» – Начать новую игру.
  + «Exit» – Выйти из приложения.
* «Texture» – В этом подменю можно осуществить выбор текстуры основных элементов игры. Их подгрузка осуществляется из папки shared в корневой директории проекта. Поддерживаются форматы файлов jpg, png и bmp. По умолчанию загружается текстура с именем файла DEFAULT.PNG, при её отсутсвии программа не запустится, показав предупреждение.
* «About» – Открыть окно дополнительных сведений о программе.

Ходы осуществляются с помощью клавиш W- шаг вверх, A - шаг влево, S – шаг вниз, D - шаг вправо. Повороты камеры осуществляются с помощью клавиш управления курсором (верх, вниз, вправо, влево).

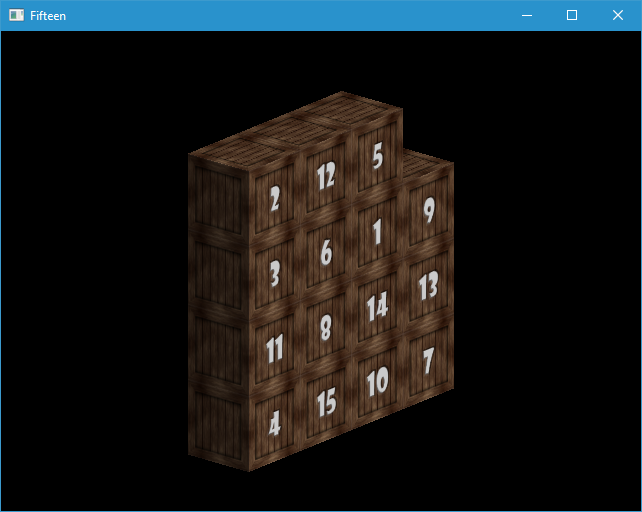


Рисунок 5. Главное окно программы

# Заключение

В ходе выполнения работы был построен алгоритм решения и реализовано приложение «Пятнашки» на языке C++ с использованием библиотек OpenGL, FreeGLUT и SOIL. Все задачи были выполнены, поставленная цель достигнута.

Возможно дальнейшее развитие продукта, планируется добавить искусственный интеллект, для механизма подсказок; добавить окружение (мир).

# Литература

1. Фролов А.В., Фролов Г.В. Библиотека системного программиста — М.: Диалог-МИФИ, 1991- ..(Издание продолжается).
2. Разработка выпускных квалификационных работ: Учебное пособие /Авторы: Г.А. Никитин, А.Д. Липенков. Под ред. В.И. Ширяева. – Челябинск: Изд-во ЮУрГУ, 1999. – 45 с.
3. Джордейн Р. Справочник программиста персонального компьютера типа IBM/PC XT, AT. — М.: Финансы и статистика, 1992
4. Олифер В.Г., Олифер Н.А.. Сетевые операционные системы. — Издательский дом «Питер», 2006.

# Приложение 1: Исходный код программы{={= { SEQ Рисунок } - 1}

### Application.h

#pragma once

#include <functional>

class Menu;

/\*\*

\* @class Приложение

\*

\* Этот класс управляет основными ресурсами приложения,

\* такими как основной цикл приложения, окно и т.д.

\* Возможно существование только одного экземпляра этого

\* класса в программе.

\*/

class Application

{

public:

typedef std::function<void()> DisplayCallback;

typedef std::function<void(unsigned char key, int mouseX, int mouseY)>

KeyboardCallback;

typedef std::function<void(int key, int modifiers, int mouseX, int mouseY)>

SpecialCallback;

Application(int argc, char \*argv[]);

~Application();

/\*\*

\* @brief Этот метод инициализирует окно GLUT,

\* открывая его по центру экрана.

\* Ширина и высота устанавливаются равними

\* 640 на 480 пикселей соответственно.

\*/

void show(const char \*windowTitle);

/\*\*

\* @brief Этот метод инициализирует окно GLUT,

\* открывая его в полноэкранном режиме.

\*/

void showFullScreen(const char \*windowTitle);

/\*\*

\* @brief Этот метод запускает основной цикл библиотеки GLUT.

\*/

void exec();

/\*\*

\* @brief Этот метод устанавливает функцию обратного

\* вызова на событие перерисовки экрана.

\*/

void setDisplayFunction(DisplayCallback display);

/\*\*

\* @brief Этот метод устанавливает функцию обратного

\* вызова на событие нажатия на клавишу клавиатуры.

\*/

void setKeyBoardFunction(KeyboardCallback keyboard);

/\*\*

\* @brief Этот метод устанавливает функцию обратного

\* вызова на событие нажатия на специальные клавиши клавиатуры.

\*/

void setSpecialFunction(SpecialCallback special);

/\*\*

\* @brief Этот метод добавляет в очереть событие перерисовки окна.

\*/

void repaint();

/\*\*

\* @brief Этот метод устанавливает меню

\*/

void setMenu(Menu &&menu);

/\*\*

\* @brief Этот метод посылает сигнал завершения приложения

\*/

void exit();

/\*\*

\* @brief Этот метод запускает обработку накопившихся сообщений

\*/

void processEvents();

private:

bool m\_isWork = true;

};

### Board.h

#pragma once

#include <vector>

/\*\*

\* @brief Фишки

\*/

typedef unsigned char Tile;

/\*\*

\* @brief Доска

\*

\* Игровая доска с фишками

\*/

class Board

{

public:

Board();

/\*\*

\* @brief Доступ к фишкам (0 - пустая клетка доски)

\*/

auto tiles() const

{ return m\_tiles; }

/\*\*

\* @brief Предикат проверяющий возможность сдвига вправо

\*/

bool canMoveRight() const;

/\*\*

\* @brief Предикат проверяющий возможность сдвига влево

\*/

bool canMoveLeft() const;

/\*\*

\* @brief Предикат проверяющий возможность сдвига вверх

\*/

bool canMoveUp() const;

/\*\*

\* @brief Предикат проверяющий возможность сдвига вниз

\*/

bool canMoveDown() const;

/\*\*

\* @brief Сдвинуть фишки вправо

\*/

void moveRight();

/\*\*

\* @brief Сдвинуть фишки влево

\*/

void moveLeft();

/\*\*

\* @brief Сдвинуть фишки вниз

\*/

void moveDown();

/\*\*

\* @brief Сдвинуть фишки вверх

\*/

void moveUp();

/\*\*

\* @brief Начальное положение фишек на доске

\*/

void init();

/\*\*

\* @brief Тасовка фишек

\*/

void shuffle();

/\*\*

\* @brief Сериализация

\*/

void serialize(std::ostream &os);

/\*\*

\* @brief Десериализация

\*/

void deserialize(std::istream &is);

private:

std::vector<std::vector<Tile>> m\_tiles;

size\_t m\_empty\_x;

size\_t m\_empty\_y;

};

### Game.h

#pragma once

#include "Board.h"

class Application;

class GameObserver;

/\*\*

\* @class Игра

\*

\* Основная логика игры пятнашки.

\*/

class Game

{

public:

Game(Application &app);

/\*\*

\* @brief Предикат окончания игры

\*/

bool isOver() const

{ return m\_isOver; }

/\*\*

\* @brief Метод возвращает доску игры в текущем состоянии

\*/

auto board() const

{ return m\_board; }

/\*\*

\* @brief Этот метод регистрирует наблюдателя.

\* Метод не перехватывает владение объектом

\*/

void addObserver(GameObserver \*obs);

/\*\*

\* @brief Этот метод создает новую игру.

\*/

void newGame();

/\*\*

\* @brief Сдвиг фишек вверх.

\*/

void moveUp();

/\*\*

\* @brief Сдвиг фишек вниз.

\*/

void moveDown();

/\*\*

\* @brief Сдвиг фишек вправо.

\*/

void moveRight();

/\*\*

\* @brief Сдвиг фишек влево.

\*/

void moveLeft();

/\*\*

\* @brief Сохранить игру

\*/

void save();

/\*\*

\* @brief Загрузить сохраненную игру

\*/

void load();

private:

void checkOver();

void emitGameChanged();

Application &m\_app;

std::vector<GameObserver \*> m\_observers;

Board m\_board;

bool m\_isOver = false;

};

/\*\*

\* @class Базовый класс наблюдателя за игрой

\*/

class GameObserver

{

public:

virtual ~GameObserver() = default;

virtual void onGameChanged() = 0;

};

### Menu.h

#pragma once

#include <functional>

#include <map>

#include <vector>

/\*\*

\* @brief Меню

\*

\* Класс-оболочка над вспомогательным контекстным меню из

\* библиотеки GLUT. Из-за сервисной архитектуры GLUT

\* работа с объектами данного класса должна происходить

\* поочередно, т.е. при создании нового экземпляра, предыдущий

\* экземпляр становится объектом только для чтения. При попытке

\* добавить подменю или вхождение в старые объекты будет брошено

\* исключение.

\*/

class Menu

{

public:

typedef std::function<void ()> MenuCallback;

Menu();

/\*\*

\* @brief Идентификатор меню, полученый от GLUT

\*/

int id() const { return m\_id; }

/\*\*

\* @brief Добавить вхождение

\*/

Menu &addMenuEntry(std::string title, MenuCallback callback);

/\*\*

\* @brief Добавить подменю

\*/

Menu &addSubMenu(std::string title, Menu &&menu);

private:

static void emitCallback(int);

static std::map<int, MenuCallback> m\_callbacks;

static int m\_lastCallbackId;

static int m\_lastMenuChanged;

int m\_id;

};

### Renderer.h

#pragma once

#include "Game.h"

#include <experimental/filesystem>

namespace fs = std::experimental::filesystem;

class Application;

struct RendererPrivate;

/\*\*

\* @brief Отрисовщик

\*

\* Этот класс отрисовывает игру на экране

\*/

class Renderer : public GameObserver

{

public:

Renderer(Application &app, Game &game);

~Renderer();

/\*\*

\* @brief Этот метод перерисовывает контекст

\*/

void display();

/\*\*

\* @brief Этот метод будет вызван, при изменениях в игре

\*/

void onGameChanged() override;

/\*\*

\* @brief Этот метод осуществляет поворот камеры относительно оси X

\*/

void rotareCameraX(double angle);

/\*\*

\* @brief Этот метод осуществляет поворот камеры относительно оси Y

\*/

void rotareCameraY(double angle);

/\*\*

\* @brief Этот метод устанавливает текстуру для клеток

\*/

void setTexturePath(fs::path texture);

private:

RendererPrivate \*d;

};

### Application.cpp

#include <stdexcept>

#include <GL/freeglut.h>

#include <chrono>

#include <thread>

#include "Menu.h"

#include "Application.h"

namespace {

static Application \*app = nullptr;

static Application::DisplayCallback s\_display;

static Application::KeyboardCallback s\_keyboard;

static Application::SpecialCallback s\_special;

static void displayCallback()

{

if (s\_display) s\_display();

}

static void keyboardCallback(unsigned char key, int x, int y)

{

if (s\_keyboard) s\_keyboard(key, x, y);

}

static void specialCallback(int key, int x, int y)

{

int modifiers = glutGetModifiers();

if (s\_special) s\_special(key, modifiers, x, y);

}

static void reshapeCallback(int w, int h)

{

glViewport(0, 0, (GLsizei)w, (GLsizei)h);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

if (w <= h) {

glOrtho(-1.5, 1.5, -1.5\*(GLfloat)h / (GLfloat)w, 1.5\*(GLfloat)h / (GLfloat)w, -10.0, 10.0);

}

else {

glOrtho(-1.5\*(GLfloat)w / (GLfloat)h, 1.5\*(GLfloat)w / (GLfloat)h, -1.5, 1.5, -10.0, 10.0);

}

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

}

} // namespace

Application::Application(int argc, char \*argv[])

{

if (app != nullptr) {

throw std::logic\_error("Application already exists");

}

app = this;

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGB | GLUT\_DEPTH);

const int width = 640;

const int height = 480;

const int x = (glutGet(GLUT\_SCREEN\_WIDTH) - width) / 2;

const int y = (glutGet(GLUT\_SCREEN\_HEIGHT) - height) / 2;

glutInitWindowPosition(x, y);

glutInitWindowSize(width, height);

}

Application::~Application()

{

app = nullptr;

s\_display = nullptr;

s\_keyboard = nullptr;

s\_special = nullptr;

}

void Application::show(const char \*title)

{

glutCreateWindow(title);

glutKeyboardFunc(&::keyboardCallback);

glutDisplayFunc(&::displayCallback);

glutReshapeFunc(&::reshapeCallback);

glutSpecialFunc(&::specialCallback);

glutSetOption(GLUT\_ACTION\_ON\_WINDOW\_CLOSE, GLUT\_ACTION\_GLUTMAINLOOP\_RETURNS);

}

void Application::showFullScreen(const char \*title)

{

show(title);

glutFullScreen();

}

void Application::exec()

{

glutMainLoop();

}

void Application::setDisplayFunction(Application::DisplayCallback display)

{

s\_display = display;

}

void Application::setKeyBoardFunction(Application::KeyboardCallback keyboard)

{

s\_keyboard = keyboard;

}

void Application::setSpecialFunction(SpecialCallback special)

{

s\_special = special;

}

void Application::repaint()

{

glutPostRedisplay();

}

void Application::setMenu(Menu &&menu)

{

glutSetMenu(menu.id());

glutAttachMenu(GLUT\_RIGHT\_BUTTON);

}

void Application::exit()

{

glutLeaveMainLoop();

}

void Application::processEvents()

{

glutMainLoopEvent();

}

### Board.cpp

#include <algorithm>

#include <ctime>

#include <istream>

#include <ostream>

#include <functional>

#include <random>

#include <stdexcept>

#include "Board.h"

constexpr size\_t m\_size { 4 };

constexpr size\_t m\_suffleLoops { 1000 };

Board::Board()

: m\_tiles(m\_size, std::vector<Tile>(m\_size))

{}

bool Board::canMoveRight() const

{

return m\_empty\_y < m\_size - 1;

}

bool Board::canMoveLeft() const

{

return m\_empty\_y > 0;

}

bool Board::canMoveUp() const

{

return m\_empty\_x > 0;

}

bool Board::canMoveDown() const

{

return m\_empty\_x < m\_size - 1;

}

void Board::moveRight()

{

if (!canMoveRight()) return;

std::swap(m\_tiles[m\_empty\_x][m\_empty\_y], m\_tiles[m\_empty\_x][m\_empty\_y + 1]);

++m\_empty\_y;

}

void Board::moveLeft()

{

if (!canMoveLeft()) return;

std::swap(m\_tiles[m\_empty\_x][m\_empty\_y], m\_tiles[m\_empty\_x][m\_empty\_y - 1]);

--m\_empty\_y;

}

void Board::moveDown()

{

if (!canMoveDown()) return;

std::swap(m\_tiles[m\_empty\_x][m\_empty\_y], m\_tiles[m\_empty\_x + 1][m\_empty\_y]);

++m\_empty\_x;

}

void Board::moveUp()

{

if (!canMoveUp()) return;

std::swap(m\_tiles[m\_empty\_x][m\_empty\_y], m\_tiles[m\_empty\_x - 1][m\_empty\_y]);

--m\_empty\_x;

}

void Board::init()

{

Tile n {};

m\_empty\_x = m\_empty\_y = 0;

for (auto &row : m\_tiles) {

for (auto &node : row) {

node = n++;

}

}

}

void Board::shuffle()

{

typedef void (Board::\*MoveFunc)();

MoveFunc move[] = { &Board::moveUp , &Board::moveDown,

&Board::moveLeft, &Board::moveRight };

std::mt19937 rand(static\_cast<uint32\_t>(time(nullptr)));

std::uniform\_int\_distribution<> dist(0, std::size(move) - 1);

for (auto i = m\_suffleLoops; i > 0; --i) {

std::invoke(move[dist(rand)], this);

}

}

void Board::serialize(std::ostream &os)

{

for (auto &&row : m\_tiles) {

for (auto &&tile : row) {

os << int(tile) << ' ';

}

os << '\n';

}

}

void Board::deserialize(std::istream &is)

{

std::vector<std::vector<Tile>> tiles(m\_size, std::vector<Tile>(m\_size));

std::vector<Tile> check;

check.reserve(m\_size \* m\_size);

int temp;

int empty\_x = 0;

int empty\_y = 0;

int x = 0;

int y = 0;

for (auto &row : tiles) {

y = 0;

for (auto &tile : row) {

is >> temp;

tile = static\_cast<Tile>(temp);

check.push\_back(static\_cast<Tile>(temp));

if (temp == 0) {

empty\_x = x;

empty\_y = y;

}

++y;

}

++x;

}

// Check

std::sort(check.begin(), check.end());

Tile expected{ 0 };

for (auto &&actual : check) {

if (expected != actual) {

throw std::runtime\_error("Can't deserialize board");

}

++expected;

}

// If ok

m\_tiles = tiles;

m\_empty\_x = empty\_x;

m\_empty\_y = empty\_y;

}

### Game.cpp

#include <Windows.h>

#include <fstream>

#include "Application.h"

#include "Game.h"

Game::Game(Application &app)

: m\_app(app)

{

}

void Game::addObserver(GameObserver \*obs)

{

m\_observers.emplace\_back(std::move(obs));

}

void Game::newGame()

{

m\_board.init();

m\_board.shuffle();

m\_isOver = false;

emitGameChanged();

}

void Game::moveUp()

{

if (!m\_board.canMoveUp()) return;

m\_board.moveUp();

checkOver();

emitGameChanged();

}

void Game::moveDown()

{

if (!m\_board.canMoveDown()) return;

m\_board.moveDown();

checkOver();

emitGameChanged();

}

void Game::moveRight()

{

if (!m\_board.canMoveRight()) return;

m\_board.moveRight();

checkOver();

emitGameChanged();

}

void Game::moveLeft()

{

if (!m\_board.canMoveLeft()) return;

m\_board.moveLeft();

checkOver();

emitGameChanged();

}

void Game::save()

{

std::ofstream os("./Fifteen.save");

m\_board.serialize(os);

}

void Game::load()

{

std::ifstream is("./Fifteen.save");

m\_board.deserialize(is);

}

void Game::checkOver()

{

Tile expected { 0 };

m\_isOver = false;

for (auto &&row : m\_board.tiles()) {

for (auto &&actual : row) {

if (actual != expected) {

return ;

}

++expected;

}

}

m\_isOver = true;

}

void Game::emitGameChanged()

{

for (auto &&obs : m\_observers) {

obs->onGameChanged();

}

m\_app.processEvents();

if (isOver()) {

MessageBoxA(0, "You win", "Fifteen", MB\_ICONINFORMATION | MB\_OK);

newGame();

}

}

### main.cpp

#include <Windows.h>

#include <GL/freeglut.h>

#include <regex>

#include "Application.h"

#include "Game.h"

#include "Menu.h"

#include "Renderer.h"

static const char s\_title[] = "Fifteen";

static const char s\_about[] =

"Fifteen Puzzle Game\n"

"Created by Yuferov Aleksandr, ET-414, SUSU 2017\n"

"\n"

"Use W, A, S, D for game movements\n"

"Use Up, Down, Left, Right for camera movement";

int main(int argc, char \*argv[])

{

Application app(argc, argv);

app.show(s\_title);

// app.showFullScreen(s\_title);

Game game(app);

try {

game.load();

} catch (std::exception &) {

game.newGame();

}

Renderer renderer(app, game);

fs::path defaultTexPath{ "../shared/default.png" };

try {

renderer.setTexturePath(defaultTexPath);

}

catch (std::exception &) {

MessageBoxA(0, "Can't find or open file \"default.png\"",

s\_title, MB\_ICONERROR | MB\_OK);

return EXIT\_FAILURE;

}

app.setDisplayFunction([&]() {

renderer.display();

});

app.setKeyBoardFunction([&](char key, int, int) {

constexpr const unsigned char esc = 27;

switch (key) {

case 'w':

case 'W':

case 'ц':

case 'Ц':

game.moveUp();

break;

case 'a':

case 'A':

case 'ф':

case 'Ф':

game.moveLeft();

break;

case 's':

case 'S':

case 'ы':

case 'Ы':

game.moveDown();

break;

case 'd':

case 'D':

case 'в':

case 'В':

game.moveRight();

break;

case esc:

app.exit();

break;

};

});

app.setSpecialFunction([&](int key, int, int, int) {

constexpr const double angle = 10.0;

switch (key) {

case GLUT\_KEY\_UP:

renderer.rotareCameraX(+angle);

break;

case GLUT\_KEY\_DOWN:

renderer.rotareCameraX(-angle);

break;

case GLUT\_KEY\_LEFT:

renderer.rotareCameraY(-angle);

break;

case GLUT\_KEY\_RIGHT:

renderer.rotareCameraY(+angle);

break;

}

});

using std::regex;

using std::regex\_match;

regex re("\\.(png|jpg|jpeg|bmp)");

Menu textureMenu;

for (auto &&entry : fs::directory\_iterator("../shared")) {

if (fs::is\_regular\_file(entry) && regex\_match(entry.path().extension().string(), re)) {

textureMenu.addMenuEntry(entry.path().stem().string(), [&renderer, defaultTexPath, entry]() {

try {

renderer.setTexturePath(entry.path());

}

catch (std::exception &) {

MessageBoxA(0, "Can't use this file", s\_title, MB\_ICONERROR | MB\_OK);

renderer.setTexturePath(defaultTexPath);

}

});

}

}

auto gameMenu = Menu()

.addMenuEntry("New game", [&]() { game.newGame(); })

.addMenuEntry("Exit", [&]() { app.exit(); });

auto mainMenu = Menu()

.addSubMenu("Game", std::move(gameMenu))

.addSubMenu("Texture", std::move(textureMenu))

.addMenuEntry("About", [](){ MessageBoxA(0, s\_about, s\_title, MB\_OK); });

app.setMenu(std::move(mainMenu));

app.exec();

game.save();

}

### Menu.cpp

#include <GL/glut.h>

#include <stdexcept>

#include "Menu.h"

std::map<int, Menu::MenuCallback> Menu::m\_callbacks;

int Menu::m\_lastCallbackId = 0;

int Menu::m\_lastMenuChanged = 0;

void Menu::emitCallback(int id)

{

m\_callbacks[id]();

}

Menu::Menu()

: m\_id(glutCreateMenu(&Menu::emitCallback))

{

m\_lastMenuChanged = m\_id;

}

Menu &Menu::addMenuEntry(std::string title, MenuCallback callback)

{

if (m\_lastMenuChanged != m\_id) {

throw std::logic\_error("Can't add menu entry to this menu");

}

glutAddMenuEntry(title.c\_str(), m\_lastCallbackId);

m\_callbacks[m\_lastCallbackId] = callback;

++m\_lastCallbackId;

return \*this;

}

Menu &Menu::addSubMenu(std::string title, Menu &&menu)

{

if (m\_lastMenuChanged != m\_id) {

throw std::logic\_error("Can't add menu entry to this menu");

}

glutAddSubMenu(title.c\_str(), menu.m\_id);

return \*this;

}

### Renderer.cpp

#include <Windows.h>

#include <gl/GL.h>

#include <GL/glut.h>

#include <SOIL.h>

#include <stdexcept>

#include "Application.h"

#include "Renderer.h"

struct RendererPrivate

{

RendererPrivate(Application &app, Game &game);

Application &app;

Game &game;

double camRotationX = 0;

double camRotationY = 0;

GLuint tileTexture = SOIL\_CREATE\_NEW\_ID;

void init();

void drawTile(Tile number);

};

RendererPrivate::RendererPrivate(Application &app, Game &game)

: app(app), game(game)

{}

void RendererPrivate::init()

{

glEnable(GL\_LIGHTING);

glEnable(GL\_LIGHT0);

glEnable(GL\_DEPTH\_TEST);

glEnable(GL\_COLOR\_MATERIAL);

glEnable(GL\_TEXTURE\_2D);

GLfloat mat\_specular[] = { 1.0, 1.0, 1.0, 1.0 };

GLfloat mat\_shininess[] = { 100.0 };

GLfloat light\_position[] = { 1.0, 1.0, 2.0, 0.0 };

glClearColor(0.0, 0.0, 0.0, 0.0);

glShadeModel(GL\_SMOOTH);

glMaterialfv(GL\_FRONT, GL\_SPECULAR, mat\_specular);

glMaterialfv(GL\_FRONT, GL\_SHININESS, mat\_shininess);

glLightfv(GL\_LIGHT0, GL\_POSITION, light\_position);

}

void RendererPrivate::drawTile(Tile number)

{

// Zero tile is empty

if (number == 0) {

return ;

}

// Size of cube

constexpr static const float size = 0.5;

// Texture quads size

constexpr const float tex\_size = 0.25;

// Face with number

const float tex\_base\_v = (int(number) / 4) \* tex\_size;

const float tex\_base\_u = (int(number) % 4) \* tex\_size;

// Base texture coords

const GLfloat tex\_base[4][2]

{

{ tex\_base\_u + tex\_size, tex\_base\_v },

{ tex\_base\_u + tex\_size, tex\_base\_v + tex\_size },

{ tex\_base\_u, tex\_base\_v + tex\_size },

{ tex\_base\_u, tex\_base\_v }

};

// Other sides

constexpr const float tex\_other\_v = 0.0;

constexpr const float tex\_other\_u = 0.0;

// Other texture coords

constexpr const static GLfloat tex\_other[4][2]

{

{ tex\_other\_v + tex\_size, tex\_other\_u },

{ tex\_other\_v, tex\_other\_u },

{ tex\_other\_v, tex\_other\_u + tex\_size },

{ tex\_other\_v + tex\_size, tex\_other\_u + tex\_size }

};

constexpr const static GLfloat n[6][3] =

{

{ -1.0, 0.0, 0.0 },

{ 0.0, 1.0, 0.0 },

{ 1.0, 0.0, 0.0 },

{ 0.0, -1.0, 0.0 },

{ 0.0, 0.0, 1.0 },

{ 0.0, 0.0, -1.0 }

};

constexpr const static GLint faces[6][4] =

{

{ 0, 1, 2, 3 },

{ 3, 2, 6, 7 },

{ 7, 6, 5, 4 },

{ 4, 5, 1, 0 },

{ 5, 6, 2, 1 },

{ 7, 4, 0, 3 }

};

GLfloat v[8][3];

GLint i;

v[0][0] = v[1][0] = v[2][0] = v[3][0] = -size / 2;

v[4][0] = v[5][0] = v[6][0] = v[7][0] = size / 2;

v[0][1] = v[1][1] = v[4][1] = v[5][1] = -size / 2;

v[2][1] = v[3][1] = v[6][1] = v[7][1] = size / 2;

v[0][2] = v[3][2] = v[4][2] = v[7][2] = -size / 2;

v[1][2] = v[2][2] = v[5][2] = v[6][2] = size / 2;

auto tex = tex\_base;

for (i = 5; i >= 0; i--) {

tex = i == 4 ? tex\_base : tex\_other;

glBegin(GL\_QUADS);

glNormal3fv(&n[i][0]);

glTexCoord2fv(&tex[0][0]);

glVertex3fv(&v[faces[i][0]][0]);

glTexCoord2fv(&tex[1][0]);

glVertex3fv(&v[faces[i][1]][0]);

glTexCoord2fv(&tex[2][0]);

glVertex3fv(&v[faces[i][2]][0]);

glTexCoord2fv(&tex[3][0]);

glVertex3fv(&v[faces[i][3]][0]);

glEnd();

}

}

Renderer::Renderer(Application &app, Game &game)

: d(new RendererPrivate(app, game))

{

d->init();

game.addObserver(this);

}

Renderer::~Renderer()

{

delete d;

d = nullptr;

}

void Renderer::display()

{

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

glRotated(d->camRotationX, 1.0, 0.0, 0.0);

glRotated(d->camRotationY, 0.0, 1.0, 0.0);

glTranslated(-0.75, 0.75, 0);

glBindTexture(GL\_TEXTURE\_2D, d->tileTexture);

for (auto row : d->game.board().tiles()) {

for (auto tile : row ) {

d->drawTile(tile);

glTranslated(0.5, 0, 0);

}

glTranslated(-2, -0.5, 0);

}

glBindTexture(GL\_TEXTURE\_2D, 0);

glutSwapBuffers();

}

void Renderer::onGameChanged()

{

d->app.repaint();

}

void Renderer::rotareCameraX(double angle)

{

d->camRotationX += angle;

d->app.repaint();

}

void Renderer::rotareCameraY(double angle)

{

d->camRotationY += angle;

d->app.repaint();

}

void Renderer::setTexturePath(fs::path path)

{

d->tileTexture = SOIL\_load\_OGL\_texture

(

path.string().c\_str(),

SOIL\_LOAD\_RGB,

d->tileTexture,

SOIL\_FLAG\_MIPMAPS | SOIL\_FLAG\_INVERT\_Y

);

if (d->tileTexture == 0) {

throw std::runtime\_error(SOIL\_last\_result());

}

glBindTexture(GL\_TEXTURE\_2D, d->tileTexture);

glTexParameterf(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_LINEAR);

glTexParameterf(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, GL\_LINEAR);

glTexEnvf(GL\_TEXTURE\_ENV, GL\_TEXTURE\_ENV\_MODE, GL\_MODULATE);

glBindTexture(GL\_TEXTURE\_2D, 0);

d->app.repaint();

}