Министерство образования и науки РФ

Федеральное государственное бюджетное образовательное учреждение

высшего профессионального образования

Тульский государственный университет

КАФЕДРА АВТОМАТИКИ И ТЕЛЕМЕХАНИКИ

**РАЗАРБОТКА КЛОНА ИГРЫ «СОКОБАН»**

Пояснительная записка

к курсовой работе

по курсу «Объектно-ориентированное программирование»

Выполнил: студент группы 220601 \_\_\_\_\_\_\_\_\_ Белым А.А.

(подпись)

Проверил: д. ф.-м. н., проф. каф. АТМ \_\_\_\_\_\_\_\_\_ Середин О.С.

(подпись)

Тула 2013

**Аннотация**

Темой данной работы является разработка клона логической игры «Сокобан».

В данной работе решается следующая задача: разработка и реализация согласно парадигме объектно-ориентированного программирования на языке Си++ набора классов, моделирующих игру «Сокобан», и использование полученных классов при реализации игровой программы, также программируемой с применением объектно-ориентированной методологии.

Работа содержит в себе следующие разделы: введение, постановка задачи, построение абстрактной модели, описание структуры классов, описание демонстрационной программы, описание тестового примера и заключение. Раздел описания структуры классов содержит подразделы формальное описание классов, описание структур данных и функций, инструкция программисту. Раздел описания демонстрационной программы содержит подразделы текст демонстрационной программы и инструкция пользователю.

Итого в данной работе насчитывается 63 страницы, 5 рисунков, 59 таблиц и 6 пунктов списка использованной литературы.

**Содержание**

*Разраб.*

*Пров.*

*Н. контр.*

*Утв.*

*Изм.*

*Лист*

*№ докум.*

*Подп*.

*Дата*

*Лит.*

*Лист*

*Листов*

*АТМК.138.000000П3*

*«Разработка клона игры Сокобан». Записка пояснительная*

*ТулГУ гр.220601*

*Белым А.А.*

*Середин О.С.*

Введение 4

1. Постановка задачи 7

2. Построение абстрактной модели 7

3. Структура классов 10

3.1. Формальное описание классов 10

3.2. Описание структур данных, функций 23

3.3. Инструкция программисту 49

4. Демонстрационная программа 57

4.1. Текст программы 57

4.2. Инструкция пользователя 58

5. Тестовый пример 59

Заключение 62

Список использованных источников 62

*63*

*3*

# Введение

В ходе выполнения данной работы решается задача написания набора классов, моделирующих игру «Сокобан», и реализация игровой программы с использованием консольного и графического интерфейсов. Также предпринимается попытка достичь максимальной унификации кодовой базы для версий под различные операционные системы, в частности, Microsoft Windows и Linux.

Ради достижения поставленных целей в качестве языка программирования при разработке данного проекта был выбран Си++ - современный, объектно-ориентированный, эффективный и кроссплатформенный язык, созданный как расширение языка Си для работы с объектно-ориентированной парадигмой. В качестве компилятора в среде Linux использовались компиляторы GCC(GNU Compiler’s Collection) и clang/LLVM, в среде Windows – компилятор Microsoft Visual C++ 2010. Для создания кроссплатформенных интерфейсов используются библиотеки Curses для консольного режима – Ncurses под Linux, и PDCurses под Windows, и библиотека SDL для работы в графическом режиме. Для организации работы с файловой системой используется boost::filesystem.

Опишем кратко саму игру «Сокобан».

«Сокобан» (в переводе с японского — «кладовщик») — это классическая головоломка, созданная в Японии, по сложности сопоставимая с кубиком Рубика, шашками и даже шахматами. Автором оригинальной игры является Хироюки Имабаяши, создавший Сокобан в 1980-м году. В 1982 году игра была издана японской компанией Thinking Rabbit («Думающий Кролик»).

Правила игры «Сокобан» очень просты. На складе, представленном в игре в виде плана, находится кладовщик и ящики. Задача состоит в перемещении ящиков по лабиринту (складу) с целью поставить их на заданные конечные места. При этом ящики можно толкать, но нельзя тянуть. Кроме того, нельзя перемещать более одного ящика зараз. Кладовщик может свободно перемещаться по складу, но не может проходить через ящики и стены.

Игра Sokoban представляет интерес с точки зрения вычислительной сложности. Было доказано, что задача решения уровней Sokoban относится к классам сложности NP-hard и PSPACE-complete.

Сложность решения уровней Sokoban вызвана как сильным ветвлением дерева решений (сопоставимым с шахматами), так и большой его глубиной - для решения некоторых уровней требуется больше 1000 толканий ящиков. Большие уровни Sokoban в настоящее время не поддаются решению компьютерными алгоритмами.

В данной работе используется объектно-ориентированный подход к построению абстракций и программированию. Для начала рассмотрим недостатки структурного подхода к программированию, которое использовалось до появления ООП.

Деление программы на функции и модули является основой структурного программирования. Структурное программирование представляет собой нечто не вполне определенное, однако в течение нескольких десятков лет, пока не была разработана концепция объектно-ориентированного программирования, оно оставалось важным способом организации программ.

В непрекращающемся процессе роста и усложнения программ стали постепенно выявляться недостатки структурного подхода к программированию.

Большие программы обычно содержат множество функций и глобальных пе-  
ременных. Проблема процедурного подхода заключается в том, что число воз-  
можных связей между глобальными переменными и функциями может быть  
очень велико, поэтому когда изменения вносятся в глобальные данные больших программ, бывает непросто быстро определить, какие функции необходимо скорректировать. Даже в том случае, когда это удается сделать, из-за многочисленных связей между функциями и данными исправленные функции начинают некорректно работать с другими глобальными данными. Таким образом, любое изменение влечет за собой далеко идущие последствия.

Вторая, более важная, проблема процедурного подхода заключается в том, что отделение данных от функций оказывается малопригодным для отображения картины реального мира. В реальном мире нам приходится иметь дело с физическими объектами, такими, например, как люди или машины. Эти объекты нельзя отнести ни к данным, ни к функциям, поскольку реальные вещи представляют собой совокупность свойств и поведения. Таким образом, ни отдельно взятые данные, ни отдельно взятые функции не способны адекватно отобразить объекты реального мира.

Для решения этих проблем ООП предлагает три основные концепции – инкапсуляция, наследование и полиморфизм.

Основополагающей идеей объектно-ориентированного подхода является инкапсуляция – объединение данных и действий, производимых над этими данными, в единое целое, которое называется объектом.

Функции объекта, называемые в C++ методами или функциями-членами, обычно предназначены для доступа к данным объекта. Если необходимо считать какие-либо данные объекта, нужно вызвать соответствующий метод, который выполнит считывание и возвратит требуемое значение. Прямой доступ к данным невозможен. Данные сокрыты от внешнего воздействия, что защищает их от случайного изменения. Говорят, что данные и методы инкапсулированы. Термины сокрытие и инкапсуляция данных являются ключевыми в описании объектно-ориентированных языков.

Важным понятием ООП является понятие класса. Класс является описанием совокупности сходных между собой объектов. Объект класса часто также называют экземпляром класса.

Понятие класса приводит нас к понятию наследования. В повседневной жизни мы часто сталкиваемся с разбиением классов на подклассы: например, класс животные можно разбить на подклассы млекопитающие, земноводные, насекомые, птицы и т. д. Принцип, положенный в основу такого деления, заключается в том, что каждый подкласс обладает свойствами, присущими тому классу, из которого выделен данный подкласс. Кроме тех свойств, которые являются общими у данных класса и подкласса, подкласс может обладать и собственными свойствами. Подобно этому, в программировании класс также может породить множество подклассов. В C++ класс, который порождает все остальные классы, называется базовым классом, остальные классы наследуют его свойства, одновременно обладая собственными свойствами. Такие классы называются производными. Производные классы имеют свойства как унаследованные от базового класса, так и свои собственные.

Одним из достоинств объектов является то, что они дают пользователю возможность создавать свои собственные типы данных. Представьте себе, что вам необходимо работать с объектами, имеющими две координаты, например x и y. Вам хотелось бы совершать обычные арифметические операции над такими объектами. В C++ имеются средства, облегчающие создание подобных  
пользовательских типов данных.

Использование операций и функций различным образом в зависимости от того, с какими типами величин они работают, называется полиморфизмом. Когда существующая операция, например = или +, наделяется возможностью совершать действия над операндами нового типа, говорят, что такая операция является перегруженной. Перегрузка представляет собой частный случай полиморфизма и является важным инструментом ООП.

# 1. Постановка задачи

Требуется разработать и реализовать программу на языке Си++, являющуюся клоном компьютерной логической игры «Сокобан». Программа должна иметь консольный и графический интерфейс, включать в себя редактор уровней и таблицу рекордов. Программа должна работать под операционными системами Windows и Linux, при этом код для двух платформ должен быть как можно более общим.

# 2. Построение абстрактной модели

Рассмотрим классы, моделирующие игру «Сокобан».

Пожалуй, наиболее важным классом является класс «Уровень» - класс Level. Данный класс включает в себя класс «Игровая доска» - класс Board, и его предназначение заключается в управлением положением объектов уровня – экземпляров класса LevelObject - на игровой доске: добавление, удаление, перемещение и т.д.

Класс Board представляет собой набор экземпляров класса «Игровое поле» - класса Field. Задача данного класса состоит в абстрагировании от структуры данных, используемой для хранения игровых полей, и предоставлению уровню удобного интерфейса для доступа к ним – по номеру строки и столбца. Структура данных при таком подходе может быть любой – связанный список, отображение (хэш-таблица). На текущий момент в этом качестве используется шаблон «Таблица» - шаблон класса Table, который представляет двумерный массив.

Класс Field представляет собой клетку игрового поля и служит для представления некоторого места на игровой доске. Поэтому поле содержит кординаты и указатель на находящийся на нем объект.

Класс LevelObject является абстрактным игровым объектом. Он имеет связи через указатель с уровнем и полем, на которых он находится. В качестве представления сущностей игры используются производные от данного классы «Стена»(Wall), «Ящик»(Box) и «Кладовщик»(Keeper). Первые два представляют одноименные объекты, последний моделирует «главного героя», поэтому обладает способностью передвижения по уровню и соотвествующими методами.

Поскольку уровень оперирует игровыми объектами через базовый класс, то десериализация средстами самого уровня представляет некоторую проблему. Поэтому введен дополнительный класс LevelObjectFactory, который позволяет создавать объекты по имени класса. Для регистрации классов на фабрике в начле выполнения программы используются классы Registrator и NestedRegistrator.

Можно заметить, что класс Level является достаточно общим, и не производит проверки корректности – например, на уровне должен находится один кладовщик, а число ящиков должно соответствовать числу меченых полей. Проверки целостности осуществляет наследник класса Level – класс SokobanLevel(«Уровень Сокобана»).

Итак, диаграмма классов, иллюстрирующая данные связи, изображена на рисунке 1.

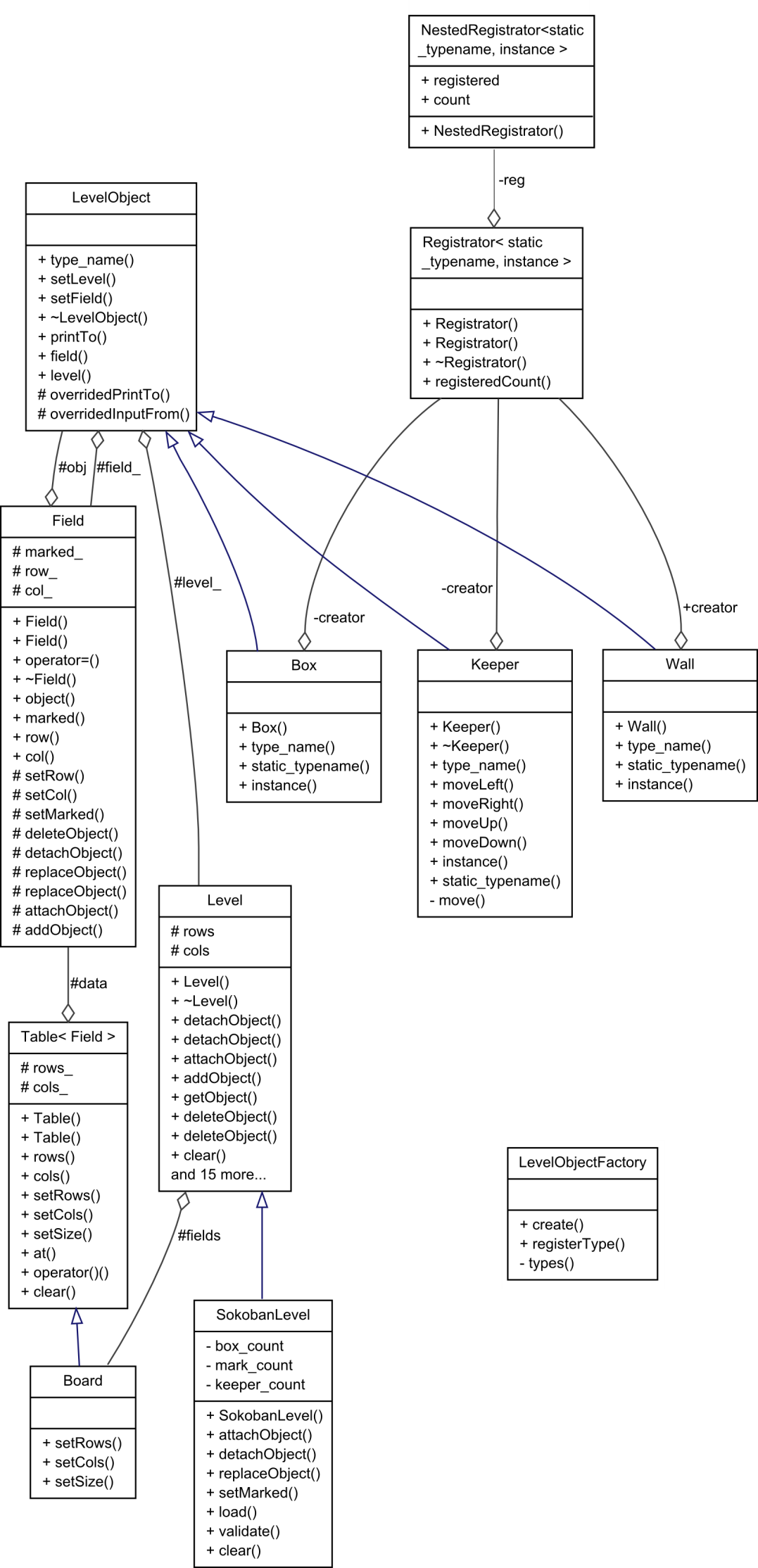


Рисунок 1 – Диаграмма классов, моделирующих игру «Сокобан»

# 3. Структура классов

## 3.1. Формальное описание классов

Далее приводится определение класса "Игровая доска"(набор игровых полей) – файл Board.h:

#if !defined(LEVEL\_H)&&!defined(DOXYGEN)

#include "Level.h"

#else

#pragma once

//!Класс "Игровая доска"(набор игровых полей).

class Board**:**public Table**<**Field**>{**

public**:**

/\*!Установка количества строк.

\* **\param** rows

\*/

void setRows**(**size\_t rows**);**

/\*!Установка количества столбцов.

\* **\param** cols

\*/

void setCols**(**size\_t cols**);**

/\*!Установка размера: установка количества строк и столбцов.

\* **\param** rows

\* **\param** cols

\*/

void setSize**(**size\_t rows**,**size\_t cols**);**

**};**

#endif

Далее приводится определение класса "Игровое поле"(клетка игровой доски) – файл Field.h:

#if !defined(LEVEL\_H)&&!defined(DOXYGEN)

#include "Level.h"

#else

#pragma once

//!Класс "Игровое поле"(клетка игровой доски).

class Field**{**

protected**:**

mutable LevelObject **\***obj**;**//!<объект, находящийся на данном поле

bool marked\_**;**//!<метка поля

int row\_**;**//!<номер строка, в которой находится поле

int col\_**;**//!<номер, в котором находится поле

/\*!Установка номера строки, в которой находится поле.

\* **\param** row номер строки

\*/

void setRow**(**int row**);**

/\*!Установка номера столбца, в которой находится поле.

\* **\param** col номер столбца

\*/

void setCol**(**int col**);**

/\*!Установка метки поля.

\* **\param** m флаг метки поля

\*/

void setMarked**(**bool m**);**

/\*!Отключение и удаление объекта, находящегося на поле.

\*/

void deleteObject**()**const**;**

/\*!Отключение объекта, находящегося на поле, от данного поля.

\* **\return** указатель на отключенный объект.

\*/

LevelObject**\*** detachObject**()**const**;**

/\*!Замена объекта, находящегося на поле,

\* на объект, заданный указателем, с возможностью отключения или удаления

\* старого объекта

\* **\param** new\_obj указатель на новый объект

\* **\param** destroy флаг удаления старого объекта

\* **\return** указатель на старый объект.

\*/

LevelObject**\*** replaceObject**(**LevelObject **\***new\_obj**,**bool destroy**=true);**

/\*!Замена объекта, находящегося на поле,

\* на новый создаваемый по имени заданного типа объект,

\* с возможностью отключения или удаления старого объекта.

\* **\param** type\_name имя класса нового объекта

\* **\param** destroy флаг удаления старого объекта

\* **\return** указатель на старый объект.

\*/

LevelObject**\*** replaceObject**(**const std**::**string**&** type\_name**,**bool destroy**=true);**

/\*!Добавление на поле существующего объекта,

\* заданного указателем на него.

\* **\param** new\_obj указатель на добавляемый объект

\* **\return** true, если операция выполнена успешно, иначе false.

\*/

bool attachObject**(**LevelObject**\*** new\_obj**)**const**;**

/\*!Создание и добавление на поле нового объекта,

\* заданного именем типа.

\* **\param** type\_name имя класса нового объекта

\* **\return** true, если операция выполнена успешно, иначе false.

\*/

bool addObject**(**const std**::**string**&** type\_name**)**const**;**

public**:**

friend class Board**;**//!<класс "Игровая доска"

friend class Level**;**//!<класс "Игровой уровень"

/\*!Конструктор по умолчанию.

\*/

Field**();**

/\*!Конструктор копирования.

\* **\param** other копируемое значение

\*/

Field**(**const Field**&** other**);**

/\*!Копирующий оператор присваивания.

\* **\param** other копируемое значение

\*/

void **operator=(**const Field**&** other**);**

/\*!Деструктор.

\*/

**~**Field**();**

/\*!Сохранение поля: оператор вывода в поток.

\* **\param** os поток вывода

\* **\param** f выводимое поле

\* **\return** измененный поток вывода

\*/

friend std**::**ostream**&** **operator<<(**std**::**ostream **&**os**,**const Field**&** f**);**

/\*!Загрузка поля: оператор ввода из потока.

\* **\param** is поток ввода

\* **\param** f загружаемое поле

\* **\return** измененный поток ввода

\*/

friend std**::**istream**&** **operator>>(**std**::**istream**&** is**,**Field **&**f**);**

/\*!Получение объекта, находящегося на поле.

\* **\return** указатель на связанный объект.

\*/

LevelObject**\*** object**();**

/\*!Получение метки поля.

\* **\return** флаг метки поля.

\*/

bool marked**()**const**;**

/\*!Получение номера строки, в которой находится поле.

\* **\return** номер строки поля.

\*/

int row**()**const**;**

/\*!Получение номера столбца, в котором находится поле.

\* **\return** номер столбца поля.

\*/

int col**()**const**;**

**};**

#endif

Далее приводится определение класса "Кладовщик" – файл Keeper.h:

#ifndef KEEPER\_H

#define KEEPER\_H

#include "LevelObjectFactory.h"

#include "Registrator.h"

//!Класс "Кладовщик" (англ. Warehouse keeper, яп. Sokoban).

class Keeper**:**public LevelObject**{**

/\*!Перемещение от текущего положения на указанное количество строк и столбцов.

\* **\param** dx смещение по столбцам

\* **\param** dy смещение по строкам

\*/

void move**(**int dx**,**int dy**);**

public**:**

/\*!Конструктор по умолчанию.

\*/

Keeper**(){**

**}**

/\*!Деструктор.

\*/

**~**Keeper**(){**

**}**

/\*!Создание нового объекта данного класса.

\* **\return** указатель на созданный объект

\*/

static LevelObject**\*** instance**(){return** **new** Keeper**;}**

/\*!Получение имени данного класса.

\* **\return** имя класса данного объекта("Keeper").

\*/

std**::**string type\_name**()**const**{return** static\_typename**();}**

/\*!Статическая функция получения имени класса.

\* **\return** имя класса данного объекта("Keeper").

\*/

static std**::**string static\_typename**(){return** "Keeper"**;}**

/\*!Перемещение влево.

\*/

void moveLeft**(){**move**(-**1**,**0**);}**

/\*!Перемещение врпаво.

\*/

void moveRight**(){**move**(**1**,**0**);}**

/\*!Перемещение вверх.

\*/

void moveUp**(){**move**(**0**,-**1**);}**

/\*!Перемещение вниз.

\*/

void moveDown**(){**move**(**0**,**1**);}**

private**:**

/\*!Объект, регистрирующий класс на фабрике.

\*/

Registrator**<**static\_typename**,**instance**>** creator**;**

**};**

#endif //KEEPER\_H

Далее приводится определение класса "Игровой уровень" файл Level.h:

#ifndef LEVEL\_H

#define LEVEL\_H

#include <iostream>

#include "../Core/DataStructures.h"

#include "LevelObject.h"

#include "Field.h"

#include "Board.h"

//! Класс "Игровой уровень"

class Level**{**

protected**:**

int rows**,**//!<число строк

cols**;**//!<число столбцов

Board fields**;**//!<игровая доска(набор игровых полей)

public**:**

/\*!Конструктор, задающий размер уровня.

\* **\param** rows количество строк

\* **\param** cols количество столбцов

\*/

Level**(**int rows**=**0**,**int cols**=**0**);**

/\*!Виртуальный деструктор.

\*/

virtual **~**Level**();**

/\*!Отключение от уровня объекта,

\* находящегося в указанных строке и столбце.

\* **\param** row строка, в которой находится объект

\* **\param** col столбец, в котором находится объект

\* **\return** указатель на отключенный объект

\*/

virtual LevelObject**\*** detachObject**(**int row**,**int col**);**

/\*!Отключение от уровня объекта, заданного указателем на него.

\* **\param** obj

\* **\return**

\*/

LevelObject**\*** detachObject**(**LevelObject **\***obj**);**

/\*!Добавление на уровень существующего объекта,

\* заданного указателем на него, и размещение его

\* в заданных строке и столбце.

\* **\param** obj указатель на добавляемый объект

\* **\param** row строка, куда добавляется объект

\* **\param** col столбец, куда возвращается объект

\* **\return** true, если операция выполнена успешно, иначе false.

\*/

virtual bool attachObject**(**LevelObject **\***obj**,**int row**,** int col**);**

/\*!Создание и добавление на уровень нового объекта,

\* и размещение его

\* в заданных строке и столбце.

\* **\param** type\_name имя класса нового объекта

\* **\param** row строка, куда добавляется объект

\* **\param** col столбец, куда возвращается объект

\* **\return** true, если операция выполнена успешно, иначе false.

\*/

bool addObject**(**const std**::**string**&** type\_name**,**int row**,**int col**);**

/\*!Получение объекта,находящегося в заданных строке и столбце.

\* **\param** row строка, в которой находится объект

\* **\param** col столбец, в котором находится объект

\* **\return** указатель на запрошенный объект

\*/

LevelObject**\*** getObject**(**int row**,**int col**);**

/\*!Удаление объекта,

\* находящегося в указанных строке и столбце.

\* **\param** row строка, в которой находится объект

\* **\param** col столбец, в котором находится объект

\*/

void deleteObject**(**int row**,** int col**);**

/\*!Удаление объекта, заданного указателем на него.

\* **\param** obj указатель на удаляемый объект

\*/

void deleteObject**(**LevelObject**\*** obj**);**

/\*!Очистка уровня: удаление всех объектов и обнуление размера уровня.

\*/

virtual void clear**();**

/\*!Установка размера уровня.

\* **\param** rows количество строк

\* **\param** cols количество столбцов

\*/

void setSize**(**int rows**,**int cols**);**

/\*!Замена объекта, находящегося в заданных строке и столбце,

\* на объект, заданный указателем, с возможностью отключения или удаления

\* старого объекта.

\* **\param** obj указатель на новый объект

\* **\param** row строка, в которой находится старый объект

\* **\param** col столбец, в котором находится старый объект

\* **\param** destroy флаг удаления старого объекта

\* **\return** указатель на старый объект.

\*/

virtual LevelObject**\*** replaceObject**(**LevelObject **\***obj**,**int row**,** int col**,**bool destroy**=true);**

/\*!Замена объекта, находящегося в заданных строке и столбце,

\* на новый создаваемый по имени заданного типа объект,

\* с возможностью отключения или удаления старого объекта.

\* **\param** type\_name имя класса нового объекта

\* **\param** row строка, в которой находится старый объект

\* **\param** col столбец, в котором находится старый объект

\* **\param** destroy флаг удаления старого объекта

\* **\return** указатель на старый объект.

\*/

LevelObject**\*** replaceObject**(**const std**::**string**&** type\_name**,**int row**,** int col**,**bool destroy**=true);**

/\*!Перемещение объекта, находящегося в заданных строке и столбце,

\* в новое положение, также задаваемое строкой и столбцом.

\* **\param** row0 строка, в которой находится объект

\* **\param** col0 столбец, в котором находится объект

\* **\param** row строка, в которую перемещается объект

\* **\param** col столбец, в который перемещается объект

\* **\return** true, если операция выполнена успешно, иначе false.

\*/

bool moveObject**(**int row0**,**int col0**,**int row**,**int col**);**

/\*!Проверка меченых полей: проверка присутствия на каждом меченом поле

\* объекта указанного типа.

\* **\param** type\_name имя класса объектов, стоящих на меченых полях

\* **\return** true, если объекты заданного класса находятся на всех меченых полях, иначе false.

\*/

bool check**(**const std**::**string**&** type\_name**);**

/\*!Установка или снятие метки с указанного поля.

\* **\param** row строка, в которой находится поле

\* **\param** col столбец, в котором находится поле

\* **\param** val флаг метки поля

\* **\return** true, если операция выполнена успешно, иначе false.

\*/

virtual bool setMarked**(**int row**,**int col**,**bool val**=true);**

/\*!Проверка, является ли указанное поле меченым.

\* **\param** row строка, в которой находится поле

\* **\param** col столбец, в котором находится поле

\* **\return** флаг метки поля.

\*/

bool isMarked**(**int row**,**int col**);**

/\*!Получение размера уровня.

\* **\param** r текущее количество строк

\* **\param** c текущее количество столбцов

\*/

void getSize**(**int **&**r**,**int**&** c**);**

/\*!Получение объекта, относящегося к заданному типу.

\* Возвращается первый найденный объект.

\* **\param** type\_name имя класса искомого объекта

\* **\return** указатель на найденный объект.

\*/

LevelObject**\*** getObject**(**const std**::**string**&** type\_name**);**

/\*!Сохранение уровня: вывод уровня в поток.

\* **\param** os поток вывода

\* **\return** измененный поток вывода

\*/

virtual std**::**ostream**&** save**(**std**::**ostream**&** os**);**

/\*!Загрузка уровня: ввод уровня из потока.

\* **\param** is поток ввода

\* **\return** измененный поток ввода

\*/

virtual std**::**istream**&** load**(**std**::**istream**&** is**);**

/\*!Оператор вывода уровня в поток.

\* **\param** os поток вывода

\* **\param** lvl выводимый уровень

\* **\return** измененный поток вывода

\*/

friend std**::**ostream**&** **operator<<(**std**::**ostream**&** os**,**Level**&** lvl**);**

/\*!Оператор ввода уровня из потока.

\* **\param** is поток ввода

\* **\param** lvl загружаемый уровень

\* **\return** измененный поток ввода

\*/

friend std**::**istream**&** **operator>>(**std**::**istream**&** is**,**Level**&** lvl**);**

/\*!Загрузка уровня из указанного файла.

\* **\param** filename имя файла, из которого загружается уровень

\*/

void loadLevel**(**const char**\*** filename**);**

/\*!Сохранение уровня в указанный файл.

\* **\param** filename имя файла, в который сохраняется уровень

\*/

void saveLevel**(**const char**\*** filename**);**

/\*!Проверка допустимости координаты, задаваемой строком и столбцом.

\* **\param** row номер строки

\* **\param** col номер столбца

\* **\return** true, если данная координата допустима, иначе false.

\*/

bool isValid**(**int row**,**int col**);**

/\*!Проверка целостности уровня.

\* **\return** true, если уровень находится в допустимом сотоянии, иначе false.

\*/

virtual bool validate**();**

**};**

#endif //LEVEL\_H

Далее приводится определение класса "Объект игрового уровня" – файл LevelObject.h.

#if !defined(LEVEL\_H)&&!defined(DOXYGEN)

#include "Level.h"

#else

#pragma once

#include <iostream>

#include <string>

class Level**;**

class Field**;**

//!Класс "Объект игрового уровня".

class LevelObject**{**

public**:**

/\*!Получение имени класса данного экземпляра.

\* **\return** имя класса данного объекта.

\*/

virtual std**::**string type\_name**()**const**=**0**;**

/\*!Установка игрового уровня,которому принадлежит объект.

\* **\param** level уровень, на который добавляется объект

\*/

virtual void setLevel**(**Level **\***level**);**

/\*!Установка игрового поля, на котором находится объект.

\* **\param** field поле, на которое добавлется объект

\*/

virtual void setField**(**const Field **\***field**);**

/\*!Виртуальный деструктор.

\*/

virtual **~**LevelObject**();**

/\*!Сохранение объекта: вывод его в поток.

\* **\param** os поток вывода

\* **\return** измененный поток вывода

\*/

std**::**ostream**&** printTo**(**std**::**ostream**&** os**)**const **;**

/\*!Оператор вывода в поток.

\* **\param** os поток вывода

\* **\param** obj выводимый объект

\* **\return** измененный поток вывода

\*/

friend std**::**ostream**&** **operator<<(**std**::**ostream**&** os**,**const LevelObject**&** obj**);**

/\*!Получение поля, на котором находится объект.

\* **\return** указатель на связанное поле

\*/

const Field**\*** field**()**const**;**

/\*!Получение уровня, которому принадлежит объект.

\* **\return** указатель на связанный уровень

\*/

Level**\*** level**()**const**;**

protected**:**

Level **\***level\_**;**//!<<уровень, которому принадлежит объект

const Field **\***field\_**;**//!<поле, на котором находится объект

/\*!Сохранение специфических свойств объектов-наследников в поток.

\* **\param** os поток вывода

\* **\return** измененный поток вывода

\*/

virtual std**::**ostream**&** overridedPrintTo**(**std**::**ostream**&** os**)**const**;**

/\*!Загрузка специфических свойств объектов-наследников из потока.

\* **\param** is поток ввода

\* **\return** измененный поток ввода

\*/

virtual std**::**istream**&** overridedInputFrom**(**std**::**istream**&** is**);**

**};**

#endif

Далее приводится определение класса "Фабрика объектов уровня" – файл LevelObjectFactory.h.

#pragma once

#include "LevelObject.h"

#include <map>

#include <string>

//!Класс "Фабрика объектов уровня".

class LevelObjectFactory**{**

public**:**

/\*!Тип указателя на функцию создания объекта.

\*/

**typedef** LevelObject**\*** **(\***CreatorFuncPtr**)();**

/\*!Тип указателя на функцию получения имени класса.

\*/

**typedef** std**::**string **(\***TypeNameFuncPtr**)();**

/\*!Создание объекта указанного класса.

\* **\param** type\_name имя класса создаваемого объекта

\* **\return** указатель на созданный объект.

\*/

static LevelObject**\*** create**(**const std**::**string**&** type\_name**);**

/\*!Регистрация класса на фабрике.

\* **\param** type\_name имя регистрируемого класса

\* **\param** create функция создания объектов класса

\* **\return** true, если операция выполнена успешно, иначе false.

\*/

static bool registerType**(**const std**::**string**&** type\_name**,**CreatorFuncPtr create**);**

private**:**

/\*!Тип отображения имени класса на производящую функцию.

\*/

**typedef** std**::**map**<**std**::**string**,**CreatorFuncPtr**>** TypeInfo**;**

/\*!Получение отображения "имя класса"->"производящая функция".

\* **\return** ссылку на отображение.

\*/

static TypeInfo**&** types**(){**

static TypeInfo types\_**;**

**return** types\_**;**

**}**

**};**

/\*!Оператор считывания из потока объекта через указатель на базовый класс.

\* **\param** is поток ввода

\* **\param** new\_obj указатель на создаваемый и загружаемый объект

\* **\return** измененный поток ввода

\*/

std**::**istream**&** **operator>>(**std**::**istream**&** is**,**LevelObject**\*** **&**new\_obj**);**

Далее приводится определение вспомогательного класса регистрации на фабрике и класса регистрации на фабрике – файл Registrator.h:

#pragma once

#include "LevelObjectFactory.h"

//!Вспомогательный класс регистрации на фабрике.

template **<**LevelObjectFactory**::**TypeNameFuncPtr type\_name**,**

LevelObjectFactory**::**CreatorFuncPtr creator**>**

class NestedRegistrator**{**

public**:**

/\*!Конструктор

\*/

NestedRegistrator**(){**

registered**=**LevelObjectFactory**::**registerType**(**type\_name**(),**creator**);**

count**=**0**;**

**}**

bool registered**;**//!<флаг состояния регистрации класса.

unsigned count**;**//!<количество объектов класса

**};**

//!Класс регистрации на фабрике.

template **<**LevelObjectFactory**::**TypeNameFuncPtr type\_name**,**

LevelObjectFactory**::**CreatorFuncPtr creator**>**

class Registrator**{**

public**:**

//!Конструктор по умолчанию.

Registrator**(){**

reg**.**count**++;**

**}**

/\*!Конструктор копирования.

\* **\param** other игнорируется

\*/

Registrator**(**const Registrator **&**other**){**

reg**.**count**++;**

**}**

//!Деструктор

**~**Registrator**(){**

reg**.**count**--;**

**}**

/\*!Получение количества зарегистрированных объектов.

\* **\return** количество объектов данного класса

\*/

unsigned registeredCount**(){return** reg**.**count**;}**

private**:**

//!Статический вспомогательный объект регистрации на фабрике.

static NestedRegistrator**<**type\_name**,**creator**>** reg**;**

**};**

template **<**LevelObjectFactory**::**TypeNameFuncPtr type\_name**,**

LevelObjectFactory**::**CreatorFuncPtr creator**>**

NestedRegistrator**<**type\_name**,**creator**>**

Registrator**<**type\_name**,**creator**>::**reg**;**

Далее приводится определение класса "Уровень Сокобана" – файл SokobanLevel.h:

#pragma once

#include "Level.h"

#include "SokobanObjects.h"

//!Класс "Уровень Сокобана".

class SokobanLevel**:**public Level**{**

private**:**

int box\_count**,**//!<количество объектов класса "Ящик"

mark\_count**,**//!<количество меченых полей

keeper\_count**;**//!<количество объектов класса "Кладовщик"

public**:**

/\*Конструктор по умолчанию.

\*/

SokobanLevel**();**

/\*!Добавление на уровень существующего объекта,

\* заданного указателем на него, и размещение его

\* в заданных строке и столбце.

\* **\param** obj указатель на добавляемый объект

\* **\param** row строка, куда добавляется объект

\* **\param** col столбец, куда возвращается объект

\* **\return** true, если операция выполнена успешно, иначе false.

\*/

bool attachObject**(**LevelObject **\***obj**,**int row**,**int col**);**

/\*!Отключение от уровня объекта,

\* находящегося в указанных строке и столбце.

\* **\param** row строка, в которой находится объект

\* **\param** col столбец, в котором находится объект

\* **\return** указатель на отключенный объект

\*/

LevelObject**\*** detachObject**(**int row**,**int col**);**

/\*!Замена объекта, находящегося в заданных строке и столбце,

\* на объект, заданный указателем, с возможностью отключения или удаления

\* старого объекта.

\* **\param** obj указатель на новый объект

\* **\param** row строка, в которой находится старый объект

\* **\param** col столбец, в котором находится старый объект

\* **\param** destroy флаг удаления старого объекта

\* **\return** указатель на старый объект.

\*/

LevelObject**\*** replaceObject**(**LevelObject **\***obj**,**int row**,**int col**,**bool destroy**=true);**

/\*!Установка или снятие метки с указанного поля.

\* **\param** row строка, в которой находится поле

\* **\param** col столбец, в котором находится поле

\* **\param** val флаг метки поля

\* **\return** true, если операция выполнена успешно, иначе false.

\*/

bool setMarked**(**int row**,**int col**,**bool val**=true);**

/\*!Загрузка уровня: ввод уровня из потока.

\* **\param** is поток ввода

\* **\return** измененный поток ввода

\*/

std**::**istream**&** load**(**std**::**istream**&** is**);**

/\*!Проверка целостности уровня.

\* **\return** true, если на уровне один кладовщик, и число меченых полей

\* совпадает с числом ящиков, иначе false.

\*/

bool validate**();**

/\*!Очистка уровня: удаление всех объектов и обнуление размера уровня.

\*/

void clear**();**

**};**

Далее приводится определение классов "Ящик" и "Стена" – файл SokobanObjects.h:

#pragma once

#include "LevelObjectFactory.h"

#include "Registrator.h"

//!Класс "Ящик".

class Box**:**public LevelObject**{**

public**:**

/\*!Конструктор по умолчанию.

\*/

Box**(){**

std**::**cout**<<**creator**.**registeredCount**()<<**std**::**endl**;**

**}**

/\*!Получение имени данного класса.

\* **\return** имя класса данного объекта("Box").

\*/

std**::**string type\_name**()**const **{**

**return** static\_typename**();**

**}**

/\*!Статическая функция получения имени класса.

\* **\return** имя класса данного объекта("Box").

\*/

static std**::**string static\_typename**(){**

**return** "Box"**;**

**}**

/\*!Создание нового объекта данного класса.

\* **\return** указатель на созданный объект.

\*/

static LevelObject**\*** instance**(){**

**return** **new** Box**;**

**}**

private**:**

/\*!Объект, регистрирующий класс на фабрике.

\*/

Registrator**<**static\_typename**,**instance**>** creator**;**

**};**

//!Класс "Стена".

class Wall**:**public LevelObject**{**

public**:**

/\*!Конструктор по умолчанию.

\*/

Wall**(){**

std**::**cout**<<**creator**.**registeredCount**()<<**std**::**endl**;**

**}**

/\*!Получение имени данного класса.

\* **\return** имя класса данного объекта("Wall").

\*/

std**::**string type\_name**()**const**{**

**return** static\_typename**();**

**}**

/\*!Статическая функция получения имени класса.

\* **\return** имя класса данного объекта("Wall").

\*/

static std**::**string static\_typename**(){**

**return** "Wall"**;**

**}**

/\*!Создание нового объекта данного класса.

\* **\return** указатель на созданный объект.

\*/

static LevelObject**\*** instance**(){**

**return** **new** Wall**;**

**}**

/\*!Объект, регистрирующий класс на фабрике.

\*/

Registrator**<**static\_typename**,**instance**>** creator**;**

**};**

#include "Keeper.h"

Далее приводится определение класса "Таблица" – файл DataStructures.h:

#pragma once

#include <vector>

**using** **namespace** std**;**

//!Класс "Таблица".

template **<**typename T**>**

class Table**{**

protected**:**

size\_t rows\_**,**//!<количество строк

cols\_**;**//!<количество столбцов

#ifdef DOXYGEN

T**\*** data**;**//!<данные

#else

vector**<**vector **<**T**>** **>** data**;**//!<данные

#endif

public**:**

/\*!Конструктор по умолчанию.

\*/

Table**():**rows\_**(**0**),**cols\_**(**0**){};**

/\*!Конструктор, задающий размер таблицы.

\* **\param** rows количество строк

\* **\param** cols количество столбцов

\*/

Table**(**size\_t rows**,**size\_t cols**){**

setSize**(**rows**,**cols**);**

**}**

/\*!Получение количества строк.

\* **\return** количество строк

\*/

size\_t rows**(){return** rows\_**;}**

/\*!Получение количества столбцов.

\* **\return** количество столбцов

\*/

size\_t cols**(){return** cols\_**;}**

/\*!Установка количества строк.

\* **\param** rows количество строк

\*/

void setRows**(**size\_t rows**){**

data**.**resize**(**rows**);**

rows\_**=**rows**;**

**}**

/\*!Установка количества столбцов.

\* **\param** cols количество столбцов

\*/

void setCols**(**size\_t cols**){**

**for(**int i**=**0**;**i**<**rows\_**;++**i**){**

data**.**at**(**i**).**resize**(**cols**);**

**}**

cols\_**=**cols**;**

**}**

/\*!Установка размера таблицы.

\* **\param** rows количество строк

\* **\param** cols количество строк

\*/

void setSize**(**size\_t rows**,**size\_t cols**){**

setCols**(**cols**);**

setRows**(**rows**);**

**}**

/\*!Получение элемента по номерам строки и столбца.

\* **\param** row строка, в которой находится элемент

\* **\param** col столбец, в котором находится элемент

\* **\return** запрошенный элемент

\*/

T**&** at**(**size\_t row**,**size\_t col**){**

**return** data**.**at**(**row**).**at**(**col**);**

**}**

/\*!Оператор получения элемента по номерам строки и столбца.

\* **\param** row строка, в которой находится элемент

\* **\param** col столбец, в котором находится элемент

\* **\return** запрошенный элемент

\*/

T**&** **operator()(**size\_t row**,**size\_t col**){**

**return** at**(**row**,**col**);**

**}**

/\*!Очистка таблицы, установка размеров в ноль.

\*/

void clear**(){**

**for(**int i**=**0**;**i**<**rows\_**;++**i**){**

data**.**at**(**i**).**clear**();**

**}**

data**.**clear**();**

rows\_**=**0**;**cols\_**=**0**;**

setSize**(**0**,**0**);**

**}**

**};**

## 3.2. Описание структур данных, функций

**Класс Board**

Класс "Игровая доска"(набор игровых полей).

#include <Board.h>

**Открытые члены**

void **setRows** (size\_t rows)

void **setCols** (size\_t cols)

void **setSize** (size\_t rows, size\_t cols)

**Методы**

**void Board::setCols (size\_t *cols*)**

Установка количества столбцов.

**Аргументы:**

|  |  |
| --- | --- |
| *cols* | количество столбцов |

**void Board::setRows (size\_t *rows*)**

Установка количества строк.

**Аргументы:**

|  |  |
| --- | --- |
| *rows* | количество строк |

**void Board::setSize (size\_t *rows*, size\_t *cols*)**

Установка размера: установка количества строк и столбцов.

**Аргументы:**

|  |  |
| --- | --- |
| *rows* | количество строк |
| *cols* | количество столбцов |

**Объявления и описания членов классов находятся в файлах:**

Level/Board.h

Level/Board.cpp

**Класс Box**

Класс "Ящик".

#include <SokobanObjects.h>

**Открытые члены**

**Box** ()

std::string **type\_name** () const

**Открытые статические члены**

static std::string **static\_typename** ()

static **LevelObject** \* **instance** ()

**Закрытые данные**

**Registrator**< **static\_typename**,

**instance** > **creator**

**Конструктор(ы)**

**Box::Box ()[inline]**

Конструктор по умолчанию.

**Методы**

**static LevelObject\* Box::instance ()[inline], [static]**

Создание нового объекта данного класса.

**Возвращает:**

указатель на созданный объект.

**static std::string Box::static\_typename ()[inline], [static]**

Статическая функция получения имени класса.

**Возвращает:**

имя класса данного объекта("Box").

**std::string Box::type\_name () const[inline], [virtual]**

Получение имени данного класса.

**Возвращает:**

имя класса данного объекта("Box").

Замещает **LevelObject** (*стр.39*).

**Данные класса**

**Registrator<static\_typename,instance> Box::creator[private]**

Объект, регистрирующий класс на фабрике.

**Объявления и описания членов класса находятся в файле:**

Level/SokobanObjects.h

**Класс Field**

Класс "Игровое поле"(клетка игровой доски).

#include <Field.h>

**Открытые члены**

**Field** ()

**Field** (const **Field** &other)

void **operator=** (const **Field** &other)

**~Field** ()

**LevelObject** \* **object** ()

bool **marked** () const

int **row** () const

int **col** () const

**Защищенные члены**

void **setRow** (int **row**)

void **setCol** (int **col**)

void **setMarked** (bool m)

void **deleteObject** () const

**LevelObject** \* **detachObject** () const

**LevelObject** \* **replaceObject** (**LevelObject** \*new\_obj, bool destroy=true)

**LevelObject** \* **replaceObject** (const std::string &type\_name, bool destroy=true)

bool **attachObject** (**LevelObject** \*new\_obj) const

bool **addObject** (const std::string &type\_name) const

**Защищенные данные**

**LevelObject** \* **obj**

*объект, находящийся на данном поле*

bool **marked\_**

*метка поля*

int **row\_**

*номер строка, в которой находится поле*

int **col\_**

*номер, в котором находится поле*

**Друзья**

class **Board**

*класс "Игровая доска".*

class **Level**

*класс "Игровой уровень".*

std::ostream & **operator<<** (std::ostream &os, const **Field** &f)

std::istream & **operator>>** (std::istream &is, **Field** &f)

**Конструктор(ы)**

**Field::Field ()**

Конструктор по умолчанию.

**Field::Field (const Field & *other*)**

Конструктор копирования.

**Аргументы:**

|  |  |
| --- | --- |
| *other* | копируемое значение |

**Field::~Field ()**

Деструктор.

**Методы**

**bool Field::addObject (const std::string & *type\_name*) const[protected]**

Создание и добавление на поле нового объекта, заданного именем типа.

**Аргументы:**

|  |  |
| --- | --- |
| *type\_name* | имя класса нового объекта |

**Возвращает:**

true, если операция выполнена успешно, иначе false.

**bool Field::attachObject (LevelObject \* *new\_obj*) const[protected]**

Добавление на поле существующего объекта, заданного указателем на него.

**Аргументы:**

|  |  |
| --- | --- |
| *new\_obj* | указатель на добавляемый объект |

**Возвращает:**

true, если операция выполнена успешно, иначе false.

**int Field::col () const**

Получение номера столбца, в котором находится поле.

**Возвращает:**

номер столбца поля.

**void Field::deleteObject () const[protected]**

Отключение и удаление объекта, находящегося на поле.

**LevelObject \* Field::detachObject () const[protected]**

Отключение объекта, находящегося на поле, от данного поля.

**Возвращает:**

указатель на отключенный объект.

**bool Field::marked () const**

Получение метки поля.

**Возвращает:**

флаг метки поля.

**LevelObject \* Field::object ()**

Получение объекта, находящегося на поле.

**Возвращает:**

указатель на связанный объект.

**void Field::operator= (const Field & *other*)**

Копирующий оператор присваивания.

**Аргументы:**

|  |  |
| --- | --- |
| *other* | копируемое значение |

**LevelObject \* Field::replaceObject (LevelObject \* *new\_obj*, bool *destroy* = true)[protected]**

Замена объекта, находящегося на поле, на объект, заданный указателем, с возможностью отключения или удаления старого объекта

**Аргументы:**

|  |  |
| --- | --- |
| *new\_obj* | указатель на новый объект |
| *destroy* | флаг удаления старого объекта |

**Возвращает:**

указатель на старый объект.

**LevelObject \* Field::replaceObject (const std::string & *type\_name*, bool *destroy* = true)[protected]**

Замена объекта, находящегося на поле, на новый создаваемый по имени заданного типа объект, с возможностью отключения или удаления старого объекта.

**Аргументы:**

|  |  |
| --- | --- |
| *type\_name* | имя класса нового объекта |
| *destroy* | флаг удаления старого объекта |

**Возвращает:**

указатель на старый объект.

**int Field::row () const**

Получение номера строки, в которой находится поле.

**Возвращает:**

номер строки поля.

**void Field::setCol (int *col*)[protected]**

Установка номера столбца, в которой находится поле.

**Аргументы:**

|  |  |
| --- | --- |
| *col* | номер столбца |

**void Field::setMarked (bool *m*)[protected]**

Установка метки поля.

**Аргументы:**

|  |  |
| --- | --- |
| *m* | флаг метки поля |

**void Field::setRow (int *row*)[protected]**

Установка номера строки, в которой находится поле.

**Аргументы:**

|  |  |
| --- | --- |
| *row* | номер строки |

**Документация по друзьям класса и функциям, относящимся к классу**

**std::ostream& operator<< (std::ostream & *os*, const Field & *f*)[friend]**

Сохранение поля: оператор вывода в поток.

**Аргументы:**

|  |  |
| --- | --- |
| *os* | поток вывода |
| *f* | выводимое поле |

**Возвращает:**

измененный поток вывода

**std::istream& operator>> (std::istream & *is*, Field & *f*)[friend]**

Загрузка поля: оператор ввода из потока.

**Аргументы:**

|  |  |
| --- | --- |
| *is* | поток ввода |
| *f* | загружаемое поле |

**Возвращает:**

измененный поток ввода

**Объявления и описания членов классов находятся в файлах:**

Level/Field.h

Level/Field.cpp

**Класс Keeper**

Класс "Кладовщик" (англ. Warehouse keeper, яп. Sokoban).

#include <Keeper.h>

**Открытые члены**

**Keeper** ()

**~Keeper** ()

std::string **type\_name** () const

void **moveLeft** ()

void **moveRight** ()

void **moveUp** ()

void **moveDown** ()

**Открытые статические члены**

static **LevelObject** \* **instance** ()

static std::string **static\_typename** ()

**Закрытые члены**

void **move** (int dx, int dy)

**Закрытые данные**

**Registrator**< **static\_typename**,

**instance** > **creator**

**Конструктор(ы)**

**Keeper::Keeper ()[inline]**

Конструктор по умолчанию.

**Keeper::~Keeper ()[inline]**

Деструктор.

**Методы**

**static LevelObject\* Keeper::instance ()[inline], [static]**

Создание нового объекта данного класса.

**Возвращает:**

указатель на созданный объект

**void Keeper::move (int *dx*, int *dy*)[private]**

Перемещение от текущего положения на указанное количество строк и столбцов.

**Аргументы:**

|  |  |
| --- | --- |
| *dx* | смещение по столбцам |
| *dy* | смещение по строкам |

**void Keeper::moveDown ()[inline]**

Перемещение вниз.

**void Keeper::moveLeft ()[inline]**

Перемещение влево.

**void Keeper::moveRight ()[inline]**

Перемещение врпаво.

**void Keeper::moveUp ()[inline]**

Перемещение вверх.

**static std::string Keeper::static\_typename ()[inline], [static]**

Статическая функция получения имени класса.

**Возвращает:**

имя класса данного объекта("Keeper").

**std::string Keeper::type\_name () const[inline], [virtual]**

Получение имени данного класса.

**Возвращает:**

имя класса данного объекта("Keeper").

Замещает **LevelObject** (*стр.39*).

**Данные класса**

**Registrator<static\_typename,instance> Keeper::creator[private]**

Объект, регистрирующий класс на фабрике.

**Объявления и описания членов классов находятся в файлах:**

Level/Keeper.h

Level/Keeper.cpp

**Класс Level**

Класс "Игровой уровень".

#include <Level.h>

**Открытые члены**

**Level** (int **rows**=0, int **cols**=0)

virtual **~Level** ()

virtual **LevelObject** \* **detachObject** (int row, int col)

**LevelObject** \* **detachObject** (**LevelObject** \*obj)

virtual bool **attachObject** (**LevelObject** \*obj, int row, int col)

bool **addObject** (const std::string &type\_name, int row, int col)

**LevelObject** \* **getObject** (int row, int col)

void **deleteObject** (int row, int col)

void **deleteObject** (**LevelObject** \*obj)

virtual void **clear** ()

void **setSize** (int **rows**, int **cols**)

virtual **LevelObject** \* **replaceObject** (**LevelObject** \*obj, int row, int col, bool destroy=true)

**LevelObject** \* **replaceObject** (const std::string &type\_name, int row, int col, bool destroy=true)

bool **moveObject** (int row0, int col0, int row, int col)

bool **check** (const std::string &type\_name)

virtual bool **setMarked** (int row, int col, bool val=true)

bool **isMarked** (int row, int col)

void **getSize** (int &r, int &c)

**LevelObject** \* **getObject** (const std::string &type\_name)

virtual std::ostream & **save** (std::ostream &os)

virtual std::istream & **load** (std::istream &is)

void **loadLevel** (const char \*filename)

void **saveLevel** (const char \*filename)

bool **isValid** (int row, int col)

virtual bool **validate** ()

**Защищенные данные**

int **rows**

*число строк*

int **cols**

*число столбцов*

**Board** **fields**

*игровая доска(набор игровых полей)*

**Друзья**

std::ostream & **operator<<** (std::ostream &os, **Level** &lvl)

std::istream & **operator>>** (std::istream &is, **Level** &lvl)

**Конструктор(ы)**

**Level::Level (int *rows* = 0, int *cols* = 0)**

Конструктор, задающий размер уровня.

**Аргументы:**

|  |  |
| --- | --- |
| *rows* | количество строк |
| *cols* | количество столбцов |

**Level::~Level ()[virtual]**

Виртуальный деструктор.

**Методы**

**bool Level::addObject (const std::string & *type\_name*, int *row*, int *col*)**

Создание и добавление на уровень нового объекта, и размещение его в заданных строке и столбце.

**Аргументы:**

|  |  |
| --- | --- |
| *type\_name* | имя класса нового объекта |
| *row* | строка, куда добавляется объект |
| *col* | столбец, куда возвращается объект |

**Возвращает:**

true, если операция выполнена успешно, иначе false.

**bool Level::attachObject (LevelObject \* *obj*, int *row*, int *col*)[virtual]**

Добавление на уровень существующего объекта, заданного указателем на него, и размещение его в заданных строке и столбце.

**Аргументы:**

|  |  |
| --- | --- |
| *obj* | указатель на добавляемый объект |
| *row* | строка, куда добавляется объект |
| *col* | столбец, куда возвращается объект |

**Возвращает:**

true, если операция выполнена успешно, иначе false.

Переопределяется в **SokobanLevel** (*стр.44*).

**bool Level::check (const std::string & *type\_name*)**

Проверка меченых полей: проверка присутствия на каждом меченом поле объекта указанного типа.

**Аргументы:**

|  |  |
| --- | --- |
| *type\_name* | имя класса объектов, стоящих на меченых полях |

**Возвращает:**

true, если объекты заданного класса находятся на всех меченых полях, иначе false.

**void Level::clear ()[virtual]**

Очистка уровня: удаление всех объектов и обнуление размера уровня.

Переопределяется в **SokobanLevel** (*стр.44*).

**void Level::deleteObject (int *row*, int *col*)**

Удаление объекта, находящегося в указанных строке и столбце.

**Аргументы:**

|  |  |
| --- | --- |
| *row* | строка, в которой находится объект |
| *col* | столбец, в котором находится объект |

**void Level::deleteObject (LevelObject \* *obj*)**

Удаление объекта, заданного указателем на него.

**Аргументы:**

|  |  |
| --- | --- |
| *obj* | указатель на удаляемый объект |

**LevelObject \* Level::detachObject (int *row*, int *col*)[virtual]**

Отключение от уровня объекта, находящегося в указанных строке и столбце.

**Аргументы:**

|  |  |
| --- | --- |
| *row* | строка, в которой находится объект |
| *col* | столбец, в котором находится объект |

**Возвращает:**

указатель на отключенный объект

Переопределяется в **SokobanLevel** (*стр.44*).

**LevelObject \* Level::detachObject (LevelObject \* *obj*)**

Отключение от уровня объекта, заданного указателем на него.

**Аргументы:**

|  |  |
| --- | --- |
| *obj* |  |

**Возвращает:**

**LevelObject \* Level::getObject (int *row*, int *col*)**

Получение объекта,находящегося в заданных строке и столбце.

**Аргументы:**

|  |  |
| --- | --- |
| *row* | строка, в которой находится объект |
| *col* | столбец, в котором находится объект |

**Возвращает:**

указатель на запрошенный объект

**LevelObject \* Level::getObject (const std::string & *type\_name*)**

Получение объекта, относящегося к заданному типу. Возвращается первый найденный объект.

**Аргументы:**

|  |  |
| --- | --- |
| *type\_name* | имя класса искомого объекта |

**Возвращает:**

указатель на найденный объект.

**void Level::getSize (int & *r*, int & *c*)**

Получение размера уровня.

**Аргументы:**

|  |  |
| --- | --- |
| *r* | текущее количество строк |
| *c* | текущее количество столбцов |

**bool Level::isMarked (int *row*, int *col*)**

Проверка, является ли указанное поле меченым.

**Аргументы:**

|  |  |
| --- | --- |
| *row* | строка, в которой находится поле |
| *col* | столбец, в котором находится поле |

**Возвращает:**

флаг метки поля.

**bool Level::isValid (int *row*, int *col*)**

Проверка допустимости координаты, задаваемой строком и столбцом.

**Аргументы:**

|  |  |
| --- | --- |
| *row* | номер строки |
| *col* | номер столбца |

**Возвращает:**

true, если данная координата допустима, иначе false.

**std::istream & Level::load (std::istream & *is*)[virtual]**

Загрузка уровня: ввод уровня из потока.

**Аргументы:**

|  |  |
| --- | --- |
| *is* | поток ввода |

**Возвращает:**

измененный поток ввода

Переопределяется в **SokobanLevel** (*стр.44*).

**void Level::loadLevel (const char \* *filename*)**

Загрузка уровня из указанного файла.

**Аргументы:**

|  |  |
| --- | --- |
| *filename* | имя файла, из которого загружается уровень |

**bool Level::moveObject (int *row0*, int *col0*, int *row*, int *col*)**

Перемещение объекта, находящегося в заданных строке и столбце, в новое положение, также задаваемое строкой и столбцом.

**Аргументы:**

|  |  |
| --- | --- |
| *row0* | строка, в которой находится объект |
| *col0* | столбец, в котором находится объект |
| *row* | строка, в которую перемещается объект |
| *col* | столбец, в который перемещается объект |

**Возвращает:**

true, если операция выполнена успешно, иначе false.

**LevelObject \* Level::replaceObject (LevelObject \* *obj*, int *row*, int *col*, bool *destroy* = true)[virtual]**

Замена объекта, находящегося в заданных строке и столбце, на объект, заданный указателем, с возможностью отключения или удаления старого объекта.

**Аргументы:**

|  |  |
| --- | --- |
| *obj* | указатель на новый объект |
| *row* | строка, в которой находится старый объект |
| *col* | столбец, в котором находится старый объект |
| *destroy* | флаг удаления старого объекта |

**Возвращает:**

указатель на старый объект.

Переопределяется в **SokobanLevel** (*стр.45*).

**LevelObject \* Level::replaceObject (const std::string & *type\_name*, int *row*, int *col*, bool *destroy* = true)**

Замена объекта, находящегося в заданных строке и столбце, на новый создаваемый по имени заданного типа объект, с возможностью отключения или удаления старого объекта.

**Аргументы:**

|  |  |
| --- | --- |
| *type\_name* | имя класса нового объекта |
| *row* | строка, в которой находится старый объект |
| *col* | столбец, в котором находится старый объект |
| *destroy* | флаг удаления старого объекта |

**Возвращает:**

указатель на старый объект.

**std::ostream & Level::save (std::ostream & *os*)[virtual]**

Сохранение уровня: вывод уровня в поток.

**Аргументы:**

|  |  |
| --- | --- |
| *os* | поток вывода |

**Возвращает:**

измененный поток вывода

**void Level::saveLevel (const char \* *filename*)**

Сохранение уровня в указанный файл.

**Аргументы:**

|  |  |
| --- | --- |
| *filename* | имя файла, в который сохраняется уровень |

**bool Level::setMarked (int *row*, int *col*, bool *val* = true)[virtual]**

Установка или снятие метки с указанного поля.

**Аргументы:**

|  |  |
| --- | --- |
| *row* | строка, в которой находится поле |
| *col* | столбец, в котором находится поле |
| *val* | флаг метки поля |

**Возвращает:**

true, если операция выполнена успешно, иначе false.

Переопределяется в **SokobanLevel** (*стр.45*).

**void Level::setSize (int *rows*, int *cols*)**

Установка размера уровня.

**Аргументы:**

|  |  |
| --- | --- |
| *rows* | количество строк |
| *cols* | количество столбцов |

**bool Level::validate ()[virtual]**

Проверка целостности уровня.

**Возвращает:**

true, если уровень находится в допустимом состоянии, иначе false.

Переопределяется в **SokobanLevel** (*стр.45*).

**Документация по друзьям класса и функциям, относящимся к классу**

**std::ostream& operator<< (std::ostream & *os*, Level & *lvl*)[friend]**

Оператор вывода уровня в поток.

**Аргументы:**

|  |  |
| --- | --- |
| *os* | поток вывода |
| *lvl* | выводимый уровень |

**Возвращает:**

измененный поток вывода

**std::istream& operator>> (std::istream & *is*, Level & *lvl*)[friend]**

Оператор ввода уровня из потока.

**Аргументы:**

|  |  |
| --- | --- |
| *is* | поток ввода |
| *lvl* | загружаемый уровень |

**Возвращает:**

измененный поток ввода

**Объявления и описания членов классов находятся в файлах:**

Level/Level.h

Level/Level.cpp

**Класс LevelObject**

Класс "Объект игрового уровня".

#include <LevelObject.h>

**Открытые члены**

virtual std::string **type\_name** () const =0

virtual void **setLevel** (**Level** \***level**)

virtual void **setField** (const **Field** \***field**)

virtual **~LevelObject** ()

std::ostream & **printTo** (std::ostream &os) const

const **Field** \* **field** () const

**Level** \* **level** () const

**Защищенные члены**

virtual std::ostream & **overridedPrintTo** (std::ostream &os) const

virtual std::istream & **overridedInputFrom** (std::istream &is)

**Защищенные данные**

**Level** \* **level\_**

*<уровень, которому принадлежит объект*

const **Field** \* **field\_**

*поле, на котором находится объект*

**Друзья**

std::ostream & **operator<<** (std::ostream &os, const **LevelObject** &obj)

**Конструктор(ы)**

**LevelObject::~LevelObject ()[virtual]**

Виртуальный деструктор.

**Методы**

**const Field \* LevelObject::field () const**

Получение поля, на котором находится объект.

**Возвращает:**

указатель на связанное поле

**Level \* LevelObject::level () const**

Получение уровня, которому принадлежит объект.

**Возвращает:**

указатель на связанный уровень

**std::istream & LevelObject::overridedInputFrom (std::istream & *is*)[protected], [virtual]**

Загрузка специфических свойств объектов-наследников из потока.

**Аргументы:**

|  |  |
| --- | --- |
| *is* | поток ввода |

**Возвращает:**

измененный поток ввода

**std::ostream & LevelObject::overridedPrintTo (std::ostream & *os*) const[protected], [virtual]**

Сохранение специфических свойств объектов-наследников в поток.

**Аргументы:**

|  |  |
| --- | --- |
| *os* | поток вывода |

**Возвращает:**

измененный поток вывода

**std::ostream & LevelObject::printTo (std::ostream & *os*) const**

Сохранение объекта: вывод его в поток.

**Аргументы:**

|  |  |
| --- | --- |
| *os* | поток вывода |

**Возвращает:**

измененный поток вывода

**void LevelObject::setField (const Field \* *field*)[virtual]**

Установка игрового поля, на котором находится объект.

**Аргументы:**

|  |  |
| --- | --- |
| *field* | поле, на которое добавлется объект |

**void LevelObject::setLevel (Level \* *level*)[virtual]**

Установка игрового уровня,которому принадлежит объект.

**Аргументы:**

|  |  |
| --- | --- |
| *level* | уровень, на который добавляется объект |

**virtual std::string LevelObject::type\_name () const[pure virtual]**

Получение имени класса данного экземпляра.

**Возвращает:**

имя класса данного объекта.

Замещается в **Wall** (*стр.48*), **Keeper** (*стр.31*) и **Box** (*стр.25*).

**Документация по друзьям класса и функциям, относящимся к классу**

**std::ostream& operator<< (std::ostream & *os*, const LevelObject & *obj*)[friend]**

Оператор вывода в поток.

**Аргументы:**

|  |  |
| --- | --- |
| *os* | поток вывода |
| *obj* | выводимый объект |

**Возвращает:**

измененный поток вывода

**Объявления и описания членов классов находятся в файлах:**

Level/LevelObject.h

Level/LevelObject.cpp

**Класс LevelObjectFactory**

Класс "Фабрика объектов уровня".

#include <LevelObjectFactory.h>

**Открытые типы**

typedef **LevelObject** \*(\* **CreatorFuncPtr** )()

typedef std::string(\* **TypeNameFuncPtr** )()

**Открытые статические члены**

static **LevelObject** \* **create** (const std::string &type\_name)

static bool **registerType** (const std::string &type\_name, **CreatorFuncPtr** **create**)

**Закрытые типы**

typedef std::map< std::string,

**CreatorFuncPtr** > **TypeInfo**

**Закрытые статические члены**

static **TypeInfo** & **types** ()

**Определения типов**

**typedef LevelObject\*(\* LevelObjectFactory::CreatorFuncPtr)()**

Тип указателя на функцию создания объекта.

**typedef std::map<std::string,CreatorFuncPtr> LevelObjectFactory::TypeInfo[private]**

Тип отображения имени класса на производящую функцию.

**typedef std::string(\* LevelObjectFactory::TypeNameFuncPtr)()**

Тип указателя на функцию получения имени класса.

**Методы**

**LevelObject \* LevelObjectFactory::create (const std::string & *type\_name*)[static]**

Создание объекта указанного класса.

**Аргументы:**

|  |  |
| --- | --- |
| *type\_name* | имя класса создаваемого объекта |

**Возвращает:**

указатель на созданный объект.

**bool LevelObjectFactory::registerType (const std::string & *type\_name*, CreatorFuncPtr *create*)[static]**

Регистрация класса на фабрике.

**Аргументы:**

|  |  |
| --- | --- |
| *type\_name* | имя регистрируемого класса |
| *create* | функция создания объектов класса |

**Возвращает:**

true, если операция выполнена успешно, иначе false.

**static TypeInfo& LevelObjectFactory::types ()[inline], [static], [private]**

Получение отображения "имя класса"->"производящая функция".

**Возвращает:**

ссылку на отображение.

**Объявления и описания членов классов находятся в файлах:**

Level/LevelObjectFactory.h

Level/LevelObjectFactory.cpp

**Шаблон класса NestedRegistrator< type\_name, creator >**

Вспомогательный класс регистрации на фабрике.

#include <Registrator.h>

**Открытые члены**

**NestedRegistrator** ()

**Открытые атрибуты**

bool **registered**

*флаг состояния регистрации класса.*

unsigned **count**

*количество объектов класса*

**Конструктор(ы)**

**template<LevelObjectFactory::TypeNameFuncPtr type\_name, LevelObjectFactory::CreatorFuncPtr creator> NestedRegistrator< type\_name, creator >::NestedRegistrator ()[inline]**

Конструктор по умолчанию.

**Объявления и описания членов класса находятся в файле:**

Level/Registrator.h

**Шаблон класса Registrator< type\_name, creator >**

Класс регистрации на фабрике.

#include <Registrator.h>

**Открытые члены**

**Registrator** ()

*Конструктор по умолчанию.*

**Registrator** (const **Registrator** &other)

**~Registrator** ()

*Деструктор*

unsigned **registeredCount** ()

**Закрытые статические данные**

static **NestedRegistrator**

< type\_name, creator > **reg**

*Статический вспомогательный объект регистрации на фабрике.*

**Конструктор(ы)**

**template<LevelObjectFactory::TypeNameFuncPtr type\_name, LevelObjectFactory::CreatorFuncPtr creator> Registrator< type\_name, creator >::Registrator (const Registrator< type\_name, creator > & *other*)[inline]**

Конструктор копирования.

**Аргументы:**

|  |  |
| --- | --- |
| *other* | игнорируется |

**Методы**

**template<LevelObjectFactory::TypeNameFuncPtr type\_name, LevelObjectFactory::CreatorFuncPtr creator> unsigned Registrator< type\_name, creator >::registeredCount ()[inline]**

Получение количества зарегистрированных объектов.

**Возвращает:**

количество объектов данного класса

**Объявления и описания членов класса находятся в файле:**

Level/Registrator.h

**Класс SokobanLevel**

Класс "Уровень Сокобана".

#include <SokobanLevel.h>

**Открытые члены**

bool **attachObject** (**LevelObject** \*obj, int row, int col)

**LevelObject** \* **detachObject** (int row, int col)

**LevelObject** \* **replaceObject** (**LevelObject** \*obj, int row, int col, bool destroy=true)

bool **setMarked** (int row, int col, bool val=true)

std::istream & **load** (std::istream &is)

bool **validate** ()

void **clear** ()

**Закрытые данные**

int **box\_count**

*количество объектов класса "Ящик".*

int **mark\_count**

*количество меченых полей*

int **keeper\_count**

*количество объектов класса "Кладовщик".*

**Методы**

**bool SokobanLevel::attachObject (LevelObject \* *obj*, int *row*, int *col*)[virtual]**

Добавление на уровень существующего объекта, заданного указателем на него, и размещение его в заданных строке и столбце.

**Аргументы:**

|  |  |
| --- | --- |
| *obj* | указатель на добавляемый объект |
| *row* | строка, куда добавляется объект |
| *col* | столбец, куда возвращается объект |

**Возвращает:**

true, если операция выполнена успешно, иначе false.

Переопределяет метод предка **Level** (*стр.33*).

**void SokobanLevel::clear ()[virtual]**

Очистка уровня: удаление всех объектов и обнуление размера уровня.

Переопределяет метод предка **Level** (*стр.33*).

**LevelObject \* SokobanLevel::detachObject (int *row*, int *col*)[virtual]**

Отключение от уровня объекта, находящегося в указанных строке и столбце.

**Аргументы:**

|  |  |
| --- | --- |
| *row* | строка, в которой находится объект |
| *col* | столбец, в котором находится объект |

**Возвращает:**

указатель на отключенный объект

Переопределяет метод предка **Level** (*стр.34*).

**std::istream & SokobanLevel::load (std::istream & *is*)[virtual]**

Загрузка уровня: ввод уровня из потока.

**Аргументы:**

|  |  |
| --- | --- |
| *is* | поток ввода |

**Возвращает:**

измененный поток ввода

Переопределяет метод предка **Level** (*стр.35*).

**LevelObject \* SokobanLevel::replaceObject (LevelObject \* *obj*, int *row*, int *col*, bool *destroy* = true)[virtual]**

Замена объекта, находящегося в заданных строке и столбце, на объект, заданный указателем, с возможностью отключения или удаления старого объекта.

**Аргументы:**

|  |  |
| --- | --- |
| *obj* | указатель на новый объект |
| *row* | строка, в которой находится старый объект |
| *col* | столбец, в котором находится старый объект |
| *destroy* | флаг удаления старого объекта |

**Возвращает:**

указатель на старый объект.

Переопределяет метод предка **Level** (*стр.36*).

**bool SokobanLevel::setMarked (int *row*, int *col*, bool *val* = true)[virtual]**

Установка или снятие метки с указанного поля.

**Аргументы:**

|  |  |
| --- | --- |
| *row* | строка, в которой находится поле |
| *col* | столбец, в котором находится поле |
| *val* | флаг метки поля |

**Возвращает:**

true, если операция выполнена успешно, иначе false.

Переопределяет метод предка **Level** (*стр.36*).

**bool SokobanLevel::validate ()[virtual]**

Проверка целостности уровня.

**Возвращает:**

true, если на уровне один кладовщик, и число меченых полей совпадает с числом ящиков, иначе false.

Переопределяет метод предка **Level** (*стр.37*).

**Объявления и описания членов классов находятся в файлах:**

Level/SokobanLevel.h

Level/SokobanLevel.cpp

**Шаблон класса Table< T >**

Класс "Таблица".

#include <DataStructures.h>

**Открытые члены**

**Table** ()

**Table** (size\_t **rows**, size\_t **cols**)

size\_t **rows** ()

size\_t **cols** ()

void **setRows** (size\_t **rows**)

void **setCols** (size\_t **cols**)

void **setSize** (size\_t **rows**, size\_t **cols**)

T & **at** (size\_t row, size\_t col)

T & **operator()** (size\_t row, size\_t col)

void **clear** ()

**Защищенные данные**

size\_t **rows\_**

*количество строк*

size\_t **cols\_**

*количество столбцов*

T \* **data**

*данные*

**Конструктор(ы)**

**template<typename T> Table< T >::Table ()[inline]**

Конструктор по умолчанию.

**template<typename T> Table< T >::Table (size\_t *rows*, size\_t *cols*)[inline]**

Конструктор, задающий размер таблицы.

**Аргументы:**

|  |  |
| --- | --- |
| *rows* | количество строк |
| *cols* | количество столбцов |

**Методы**

**template<typename T> T& Table< T >::at (size\_t *row*, size\_t *col*)[inline]**

Получение элемента по номерам строки и столбца.

**Аргументы:**

|  |  |
| --- | --- |
| *row* | строка, в которой находится элемент |
| *col* | столбец, в котором находится элемент |

**Возвращает:**

запрошенный элемент

**template<typename T> void Table< T >::clear ()[inline]**

Очистка таблицы, установка размеров в ноль.

**template<typename T> size\_t Table< T >::cols ()[inline]**

Получение количества столбцов.

**Возвращает:**

количество столбцов

**template<typename T> T& Table< T >::operator() (size\_t *row*, size\_t *col*)[inline]**

Оператор получения элемента по номерам строки и столбца.

**Аргументы:**

|  |  |
| --- | --- |
| *row* | строка, в которой находится элемент |
| *col* | столбец, в котором находится элемент |

**Возвращает:**

запрошенный элемент

**template<typename T> size\_t Table< T >::rows ()[inline]**

Получение количества строк.

**Возвращает:**

количество строк

**template<typename T> void Table< T >::setCols (size\_t *cols*)[inline]**

Установка количества столбцов.

**Аргументы:**

|  |  |
| --- | --- |
| *cols* | количество столбцов |

**template<typename T> void Table< T >::setRows (size\_t *rows*)[inline]**

Установка количества строк.

**Аргументы:**

|  |  |
| --- | --- |
| *rows* | количество строк |

**template<typename T> void Table< T >::setSize (size\_t *rows*, size\_t *cols*)[inline]**

Установка размера таблицы.

**Аргументы:**

|  |  |
| --- | --- |
| *rows* | количество строк |
| *cols* | количество строк |

**Объявления и описания членов класса находятся в файле:**

Core/DataStructures.h

**Класс Wall**

Класс "Стена".

#include <SokobanObjects.h>

**Открытые члены**

**Wall** ()

std::string **type\_name** () const

**Открытые статические члены**

static std::string **static\_typename** ()

static **LevelObject** \* **instance** ()

**Открытые атрибуты**

**Registrator**< **static\_typename**, **instance** > **creator**

**Конструктор(ы)**

**Wall::Wall ()[inline]**

Конструктор по умолчанию.

**Методы**

**static LevelObject\* Wall::instance ()[inline], [static]**

Создание нового объекта данного класса.

**Возвращает:**

указатель на созданный объект.

**static std::string Wall::static\_typename ()[inline], [static]**

Статическая функция получения имени класса.

**Возвращает:**

имя класса данного объекта("Wall").

**std::string Wall::type\_name () const[inline], [virtual]**

Получение имени данного класса.

**Возвращает:**

имя класса данного объекта("Wall").

Замещает **LevelObject** (*стр.39*).

**Данные класса**

**Registrator<static\_typename,instance> Wall::creator**

Объект, регистрирующий класс на фабрике.

**Объявления и описания членов класса находятся в файле:**

Level/SokobanObjects.h

## 3.3. Инструкция программисту

Далее представлена реализация класса Board, файл Board.cpp:

#include "Board.h"

void Board**::**setRows**(**size\_t rows**){**

int dr**=**rows**-**rows\_**;**

cout**<<**"Board::setRows(int)!"**<<**endl**;**

Table**::**setRows**(**rows**);**

**for(**int i**=**rows\_**-**dr**;**i**<**rows\_**;++**i**)**

**for(**int j**=**0**;**j**<**cols\_**;++**j**){**

at**(**i**,**j**).**setRow**(**i**);**

at**(**i**,**j**).**setCol**(**j**);**

**}**

cout**<<**"Board::setRows(int)!"**<<**endl**;**

**}**

void Board**::**setCols**(**size\_t cols**){**

int dc**=**cols**-**cols\_**;**

cout**<<**"Board::setCols(int)!"**<<**endl**;**

Table**::**setCols**(**cols**);**

**for(**int i**=**0**;**i**<**rows\_**;++**i**)**

**for(**int j**=**cols\_**-**dc**;**j**<**cols\_**;++**j**){**

at**(**i**,**j**).**setRow**(**i**);**

at**(**i**,**j**).**setCol**(**j**);**

**}**

cout**<<**"Board::setCols(int)!"**<<**endl**;**

**}**

void Board**::**setSize**(**size\_t rows**,**size\_t cols**){**

setRows**(**rows**);**

setCols**(**cols**);**

cout**<<**"Board::setSize(int,int)!"**<<**endl**;**

**}**

Далее представлена реализация класса Field, файл Field.cpp:

#include "Field.h"

#include "LevelObjectFactory.h"

void Field**::**setRow**(**int row**){**row\_**=**row**;}**

void Field**::**setCol**(**int col**){**col\_**=**col**;}**

void Field**::**setMarked**(**bool m**){**marked\_**=**m**;}**

void Field**::**deleteObject**()**const**{**

**delete** detachObject**();**

**}**

LevelObject**\*** Field**::**detachObject**()**const**{**

cout**<<**"Field::detachObject()!"**<<**row\_**<<**","**<<**col\_**<<**endl**;**

LevelObject **\***temp**=**obj**;**

**if(**temp**){**

//temp->setLevel(0);

temp**->**setField**(**0**);**

**}**

obj**=**0**;**

cout**<<**"Field::detachObject()!"**<<**endl**;**

**return** temp**;**

**}**

LevelObject**\*** Field**::**replaceObject**(**LevelObject **\***new\_obj**,**bool destroy**){**

LevelObject **\***lo**=**detachObject**();**

**if(!**attachObject**(**new\_obj**)){**

attachObject**(**lo**);**

**return** 0**;**

**}**

**if** **(**lo**&&**destroy**){**

**delete** lo**;**

lo**=**0**;**

**};**

**return** lo**;**

**}**

LevelObject**\*** Field**::**replaceObject**(**const std**::**string**&** type\_name**,**bool destroy**){**

**return** replaceObject**(**LevelObjectFactory**::**create**(**type\_name**),**destroy**);**

**}**

bool Field**::**attachObject**(**LevelObject**\*** new\_obj**)**const**{**

**if(**obj**||!**new\_obj**){**

cout**<<**"attach object failed!"**<<**endl**;**

**return** **false;**

**}**

obj**=**new\_obj**;**

obj**->**setField**(this);**

**return** **true;**

**}**

bool Field**::**addObject**(**const std**::**string**&** type\_name**)**const**{**

**return** attachObject**(**LevelObjectFactory**::**create**(**type\_name**));**

**};**

Field**::**Field**(){**obj**=**0**;**row\_**=**0**;**col\_**=**0**;**marked\_**=false;}**

Field**::**Field**(**const Field**&** other**){**

obj**=**0**;**

LevelObject **\***obj**=**other**.**detachObject**();**

attachObject**(**obj**);**

row\_**=**other**.**row\_**;**

col\_**=**other**.**col\_**;**

marked\_**=**other**.**marked\_**;**

**}**

void Field**::operator=(**const Field**&** other**){**

LevelObject **\***obj**=**other**.**detachObject**();**

attachObject**(**obj**);**

row\_**=**other**.**row\_**;**

col\_**=**other**.**col\_**;**

marked\_**=**other**.**marked\_**;**

**}**

Field**::~**Field**(){**

**;**

deleteObject**();**

**;**

**}**

std**::**ostream**&** **operator<<(**std**::**ostream **&**os**,**const Field**&** f**){**

**if(**f**.**obj**)**

os**<<\***f**.**obj**;**

**else**

os**<<**"None"**;**

os**<<**" "

**<<**f**.**marked\_**<<**" "

**<<**f**.**row\_**<<**" "

**<<**f**.**col\_**<<**endl**;**

**return** os**;**

**}**

std**::**istream**&** **operator>>(**std**::**istream**&** is**,**Field **&**f**){**

LevelObject **\***new\_obj**;**

is**>>**new\_obj**;**

is**>>**f**.**marked\_

**>>**f**.**row\_

**>>**f**.**col\_**;**

f**.**attachObject**(**new\_obj**);**

**return** is**;**

**}**

LevelObject**\*** Field**::**object**(){return** obj**;};**

bool Field**::**marked**()**const**{return** marked\_**;}**

int Field**::**row**()**const**{return** row\_**;}**

int Field**::**col**()**const**{return** col\_**;}**

Далее представлена реализация класса Keeper, файл Keeper.cpp:

#include "Keeper.h"

#include <iostream>

void Keeper**::**move**(**int dy**,**int dx**){**

int r**=**field\_**->**row**(),**

c**=**field\_**->**col**(),**

new\_r**=**r**+**dx**,**

new\_c**=**c**+**dy**;**

**if(!**field\_**){**

**return;**

**}**

**if(!**level\_**){**

**return;**

**}**

LevelObject **\***o**=**level\_**->**getObject**(**new\_r**,**new\_c**);**

**if(!**o**){**

level\_**->**moveObject**(**r**,**c**,**new\_r**,**new\_c**);**

**}**

**else** **if(**o**->**type\_name**()==**"Box"**){**

level\_**->**moveObject**(**new\_r**,**new\_c**,**new\_r**+**dx**,**new\_c**+**dy**);**

level\_**->**moveObject**(**r**,**c**,**new\_r**,**new\_c**);**

**};**

**}**

Далее представлена реализация класса Level, файл Level.cpp:

#include "Level.h"

#include "LevelObjectFactory.h"

#include <fstream>

**using** **namespace** std**;**

Level**::**Level**(**int rows**,**int cols**){**

cout**<<**"Level() constructor!"**<<**endl**;**

**this->**rows**=**0**;this->**cols**=**0**;**

setSize**(**rows**,**cols**);**

**}**

Level**::~**Level**(){**

clear**();**

**}**

void Level**::**clear**(){**

cout**<<**"Level::clear()!"**<<**endl**;**

**for(**int i**=**0**;**i**<**rows**;++**i**)**

**for(**int j**=**0**;**j**<**cols**;++**j**){**

cout**<<**"Level::deleteObject("**<<**i**<<**","**<<**j**<<**")!"**<<**endl**;**

deleteObject**(**i**,**j**);**

cout**<<**"Level::deleteObject("**<<**i**<<**","**<<**j**<<**")!"**<<**endl**;**

**}**

fields**.**clear**();**

rows**=**0**;**

cols**=**0**;**

cout**<<**"Level::clear()!"**<<**endl**;**

**}**

void Level**::**setSize**(**int rows**,**int cols**){**

cout**<<**"Level::setSize("**<<**rows**<<**","**<<**cols**<<**")!"**<<**endl**;**

clear**();**

cout**<<**"Level::setSize(int,int)!"**<<**endl**;**

fields**.**setSize**(**rows**,**cols**);**

**this->**rows**=**rows**;this->**cols**=**cols**;**

cout**<<**"Level::setSize("**<<**rows**<<**","**<<**cols**<<**")!"**<<**endl**;**

**}**

void Level**::**getSize**(**int **&**r**,**int**&** c**){**r**=**rows**;**c**=**cols**;};**

bool Level**::**isValid**(**int row**,**int col**){**

**return** **(**row**>=**0**&&**row**<**rows**&&**

col**>=**0**&&**col**<**cols**);**

**}**

bool Level**::**validate**(){return** **true;}**

LevelObject**\*** Level**::**getObject**(**int row**,**int col**){**

**if(!**isValid**(**row**,**col**))**

**return** 0**;**

**return** fields**(**row**,**col**).**object**();**

**}**

bool Level**::**addObject**(**const std**::**string**&** type\_name**,**int row**,**int col**){**

**return** attachObject**(**LevelObjectFactory**::**create**(**type\_name**),**row**,**col**);**

**}**

bool Level**::**attachObject**(**LevelObject **\***obj**,**int row**,** int col**){**

**if(!**isValid**(**row**,**col**)||**

**!**obj**)**

**return** **false;**

Field **&**f**=**fields**(**row**,**col**);**

**if(**f**.**attachObject**(**obj**))**

obj**->**setLevel**(this);**

**return** **true;**

**}**

bool Level**::**moveObject**(**int row0**,**int col0**,**int row**,**int col**){**

**if(!**isValid**(**row**,**col**)||!**isValid**(**row0**,**col0**))**

**return** **false;**

**if(!**fields**(**row0**,**col0**).**object**()||**

fields**(**row**,**col**).**object**())**

**return** **false;**

LevelObject**\*** temp**=**fields**(**row0**,**col0**).**detachObject**();**

**return** fields**(**row**,**col**).**attachObject**(**temp**);**

**}**

bool Level**::**check**(**const std**::**string**&** type\_name**){**

**for(**int i**=**0**;**i**<**rows**;++**i**)**

**for(**int j**=**0**;**j**<**cols**;++**j**){**

Field **&**f**=**fields**(**i**,**j**);**

**if(**f**.**marked**()&&**

**(!**f**.**object**()||**f**.**object**()->**type\_name**()!=**type\_name**))**

**return** **false;**

**};**

**return** **true;**

**}**

bool Level**::**setMarked**(**int row**,**int col**,**bool val**){**

**if(!**isValid**(**row**,**col**))**

**return** **false;**

fields**(**row**,**col**).**setMarked**(**val**);**

**return** **true;**

**}**

bool Level**::**isMarked**(**int row**,**int col**){**

**if(!**isValid**(**row**,**col**))**

**return** **false;**

**return** fields**(**row**,**col**).**marked**();**

**}**

ostream**&** **operator<<(**ostream**&** os**,**Level**&** lvl**){**

**return** lvl**.**save**(**os**);**

**}**

istream**&** **operator>>(**istream**&** is**,**Level**&** lvl**){**

**return** lvl**.**load**(**is**);**

**}**

LevelObject**\*** Level**::**getObject**(**const std**::**string**&** type\_name**){**

**for(**int i**=**0**;**i**<**rows**;++**i**){**

**for(**int j**=**0**;**j**<**cols**;++**j**){**

LevelObject **\***obj**=**fields**(**i**,**j**).**object**();**

**if(**obj**&&**obj**->**type\_name**()==**type\_name**)**

**return** obj**;**

**}**

**}**

cout**<<**"GetObject failed!"**<<**endl**;**

**return** 0**;**

**}**

LevelObject**\*** Level**::**detachObject**(**int row**,** int col**){**

**if(!**isValid**(**row**,**col**))**

**return** 0**;**

LevelObject **\***obj**=**fields**(**row**,**col**).**detachObject**();**

**if(**obj**)**

obj**->**setLevel**(**0**);**

**return** obj**;**

**}**

void Level**::**deleteObject**(**int row**,** int col**){**

**delete** detachObject**(**row**,**col**);**

**}**

LevelObject**\*** Level**::**detachObject**(**LevelObject**\*** obj**){**

cout**<<**"detaching object"**<<**endl**;**

**if(!(**obj**&&**

obj**->**level**()==this&&**

obj**->**field**()))**

**return** 0**;**

int r**=**obj**->**field**()->**row**(),**

c**=**obj**->**field**()->**col**();**

**if(**obj**->**field**()==&**fields**(**r**,**c**)&&**

fields**(**r**,**c**).**object**()==**obj**)**

**return** detachObject**(**r**,**c**);**

cout**<<**"detachObject failed!"**<<**endl**;**

**return** 0**;**

**}**

void Level**::**deleteObject**(**LevelObject**\*** obj**){**

**delete** detachObject**(**obj**);**

**}**

LevelObject**\*** Level**::**replaceObject**(**LevelObject **\***obj**,**int row**,** int col**,**bool destroy**){**

**if** **(!**isValid**(**row**,**col**))**

**return** 0**;**

LevelObject **\***temp**=**fields**(**row**,**col**).**replaceObject**(**obj**,false);**

**if(**fields**(**row**,**col**).**object**()==**obj**){**

obj**->**setLevel**(this);**

**}**

**if(**temp**){**

temp**->**setLevel**(**0**);**

**if(**destroy**){**

**delete** temp**;**

temp**=**0**;**

**}**

**}**

**return** temp**;**

**}**

LevelObject**\*** Level**::**replaceObject**(**const std**::**string **&**type\_name**,**int row**,** int col**,**bool destroy**){**

**return** replaceObject**(**LevelObjectFactory**::**create**(**type\_name**),**row**,**col**,**destroy**);**

**}**

void Level**::**loadLevel**(**const char**\*** filename**){**

ifstream ifs**;**

ifs**.**open**(**filename**);**

ifs**>>\*this;**

ifs**.**close**();**

**}**

void Level**::**saveLevel**(**const char**\*** filename**){**

ofstream ofs**;**

ofs**.**open**(**filename**);**

ofs**<<\*this;**

ofs**.**close**();**

**}**

std**::**ostream**&** Level**::**save**(**std**::**ostream**&** os**){**

os**<<**rows**<<**" "**<<**cols**<<**endl**;**

**for(**int i**=**0**;**i**<**rows**;++**i**){**

**for(**int j**=**0**;**j**<**cols**;++**j**){**

Field **&**f**=**fields**(**i**,**j**);**

**if(**f**.**object**()||**f**.**marked**())**

os**<<**f**;**

**}**

**}**

**return** os**;**

**}**

std**::**istream**&** Level**::**load**(**std**::**istream**&** is**){**

int rows**,**cols**;**

is**>>**rows**>>**cols**;**

setSize**(**rows**,**cols**);**

**while(!**is**.**eof**()){**

Field f**;**

is**>>**f**;**

LevelObject **\***o**=**f**.**object**();**

**if(**o**)**

o**->**setLevel**(this);**

fields**(**f**.**row**(),**f**.**col**())=**f**;**

**}**

**return** is**;**

**}**

Далее представлена реализация класса LevelObject, файл LevelObject.cpp:

#include "LevelObject.h"

void LevelObject**::**setLevel**(**Level **\***level**){**

level\_**=**level**;**

**}**

void LevelObject**::**setField**(**const Field **\***field**){**

field\_**=**field**;**

**}**

LevelObject**::~**LevelObject**(){**

cout**<<**"~LevelObject()!"**<<**endl**;**

cout**<<**"~LevelObject()!"**<<**endl**;**

**};**

std**::**ostream**&** **operator<<(**std**::**ostream**&** os**,**const LevelObject**&** obj**){**

**return** obj**.**printTo**(**os**);**

**}**

std**::**ostream**&** LevelObject**::**printTo**(**std**::**ostream**&** os**)**const **{**

os**<<**type\_name**();**

**return** overridedPrintTo**(**os**);**

**}**

const Field**\*** LevelObject**::**field**()**const**{**

**return** field\_**;**

**}**

Level**\*** LevelObject**::**level**()**const**{**

**return** level\_**;**

**}**

std**::**ostream**&** LevelObject**::**overridedPrintTo**(**std**::**ostream**&** os**)**const**{**

**return** os**;**

**};**

std**::**istream**&** LevelObject**::**overridedInputFrom**(**std**::**istream**&** is**){**

**return** is**;**

**}**

Далее представлена реализация класса LevelObjectFactory, файл LevelObjectFactory.cpp:

#include "LevelObjectFactory.h"

#include <algorithm>

LevelObject**\*** LevelObjectFactory**::**create**(**const std**::**string**&** type\_name**){**

TypeInfo**::**iterator found**;**

cout**<<**"Requested a "**<<**type\_name**<<**" instance..."**<<**endl**;**

found**=**types**().**find**(**type\_name**);**

**if(**found**!=**types**().**end**())**

**return** found**->**second**();**

**else**

**return** 0**;**

**}**

bool LevelObjectFactory**::**registerType**(**const std**::**string**&** type\_name**,**CreatorFuncPtr create**){**

cout**<<**"Type "**<<**type\_name**<<**" registration..."**<<**endl**;**

TypeInfo**::**iterator found**;**

found**=**types**().**find**(**type\_name**);**

**if(**found**==**types**().**end**()){**

types**().**insert**(**make\_pair**(**type\_name**,**create**));**

cout**<<**"Type "**<<**type\_name**<<**" registered!"**<<**endl**;**

**return** **true;**

**}** **else**

**return** **false;**

**}**

istream**&** **operator>>(**istream**&** is**,**LevelObject**\*** **&**new\_obj**){**

std**::**string type**;**

is**>>**type**;**

new\_obj**=**LevelObjectFactory**::**create**(**type**);**

**return** is**;**

**}**

Далее представлена реализация класса SokobanLevel, файл SokobanLevel.cpp:

#include "SokobanLevel.h"

SokobanLevel**::**SokobanLevel**():**box\_count**(**0**),**

mark\_count**(**0**),**

keeper\_count**(**0**)**

**{};**

bool SokobanLevel**::**attachObject**(**LevelObject **\***obj**,**int row**,**int col**){**

cout**<<**"attaching object"**<<**endl**;**

**if(**Level**::**attachObject**(**obj**,**row**,**col**)){**

std**::**string type**=**obj**->**type\_name**();**

**if(**type**==**"Keeper"**)**

keeper\_count**++;**

**else** **if(**type**==**"Box"**)**

box\_count**++;**

**return** **true;**

**}** **else**

**return** **false;**

**}**

LevelObject**\*** SokobanLevel**::**detachObject**(**int row**,**int col**){**

LevelObject **\***obj**=**Level**::**detachObject**(**row**,**col**);**

**if(**obj**){**

std**::**string type**=**obj**->**type\_name**();**

**if(**type**==**"Keeper"**)**

keeper\_count**--;**

**else** **if(**type**==**"Box"**)**

box\_count**--;**

**}**

**return** obj**;**

**}**

LevelObject**\*** SokobanLevel**::**replaceObject**(**LevelObject **\***obj**,**int row**,**int col**,**bool destroy**){**

cout**<<**"replacing object"**<<**endl**;**

LevelObject **\***temp**=**Level**::**replaceObject**(**obj**,**row**,**col**,false);**

std**::**string type**;**

**if(**obj**&&**obj**->**level**()==this){**

type**=**obj**->**type\_name**();**

**if(**type**==**"Keeper"**)**

keeper\_count**++;**

**else** **if(**type**==**"Box"**)**

box\_count**++;**

**}**

**if(**temp**){**

type**=**temp**->**type\_name**();**

**if(**type**==**"Keeper"**)**

keeper\_count**--;**

**else** **if(**type**==**"Box"**)**

box\_count**--;**

**if(**destroy**){**

**delete** temp**;**

temp**=**0**;**

**}**

**}**

**return** temp**;**

**}**

bool SokobanLevel**::**setMarked**(**int row**,**int col**,**bool val**){**

**if(**Level**::**setMarked**(**row**,**col**,**val**)){**

val**?**mark\_count**++:**mark\_count**--;**

**return** **true;**

**}**

**return** **false;**

**}**

std**::**istream**&** SokobanLevel**::**load**(**std**::**istream**&** is**){**

Level**::**load**(**is**);**

**for(**int i**=**0**;**i**<**rows**;++**i**)**

**for(**int j**=**0**;**j**<**cols**;++**j**){**

**if(**fields**(**i**,**j**).**marked**()){**

cout**<<**"new mark"**<<**endl**;**

mark\_count**++;**

**}**

**if(!**fields**(**i**,**j**).**object**())**

**continue;**

std**::**string type**=**fields**(**i**,**j**).**object**()->**type\_name**();**

**if(**type**==**"Keeper"**)**

keeper\_count**++;**

**else** **if(**type**==**"Box"**)**

box\_count**++;**

**}**

**return** is**;**

**}**

bool SokobanLevel**::**validate**(){**

cout**<<**"k: "**<<**keeper\_count

**<<**"m: "**<<**mark\_count

**<<**"b: "**<<**box\_count**<<**endl**;**

**return** keeper\_count**==**1**&&(**mark\_count**==**box\_count**);**

**}**

void SokobanLevel**::**clear**(){**

Level**::**clear**();**

keeper\_count**=**0**;**

mark\_count**=**0**;**

box\_count**=**0**;**

**}**

# 4. Демонстрационная программа

## 4.1. Текст программы

Ввиду большого объема и высокой сложности самой демонстрационной программы, приведем лишь её главную часть – файл sokoban.cpp:

#include "Logic/SokobanLogic.h"

#include "Engine/CursesEngine.h"

#include "Engine/SDLEngine.h"

#ifdef \_\_unix\_\_

#include "Platform/nix.h"

#elif \_WIN32

#include "Platform/win.h"

#endif

**namespace** fs**=**boost**::**filesystem**;**

int main**(**int argc**,**char **\*\***argv**){**

fs**::**path path**=**fs**::**path**(**argv**[**0**]).**parent\_path**();**

fs**::**current\_path**(**path**);**

SDLEngine e\_sdl**;**

CursesEngine e\_curs**;**

GameEngine **\***e**;**

SokobanLogic sokoban**;**

**if(argc<=1&&**getConsole**())**

e**=&**e\_curs**;**

**else**

e**=&**e\_sdl**;**

sokoban**.**setEngine**(**e**);**

sokoban**.**Main**();**

**return** 0**;**

**}**

## 4.2. Инструкция пользователя

Данная программа реализует логическую игру «Сокобан».

Правила игры «Сокобан» очень просты. На складе, представленном в игре в виде плана, находится кладовщик и ящики. Задача состоит в перемещении ящиков по лабиринту (складу) с целью поставить их на заданные конечные места. При этом ящики можно толкать, но нельзя тянуть. Кроме того, нельзя перемещать более одного ящика зараз. Кладовщик может свободно перемещаться по складу, но не может проходить через ящики и стены.

Данная программа имеет графический и консольный интерфейс. По умолчанию используется консольный интерфейс. Чтобы запустить игру в графическом режиме, программе необходимо передать параметр командной строки.

В программе активно используется система меню. Для изменения текущего пункта меню используются клавиши «вверх» и «вниз». Для выбора пункта меню используется клавиша «вправо». Для возврата к предыдущему экрану – клавиша «влево».

После запуска программы на экране появляется главное меню, имеющее следующие элементы:

1. New game – после выбора данного пункта, на экране появляется список уровней. После выбора уровня на экране появляется игровое поле. В консольном режиме свободное место обозначается точкой «.», кладовщик – символом «@», ящики, стоящие на свободном месте – «о», на отмеченном для установки ящика – «0», стена обозначается символом «#». Управлять кладовщиком можно с помощью клавиш со стрелками или клавиш «W», «A», «S», «D». При нажатии клавиши «Escape» происходит выход в главное меню. Текущую игру можно сохранить, нажав клавишу F2. После успешного прохождения уровня показывается таблица рекордов – чем меньше ходов затрачено, тем выше место.

2. Load game – позволяет загрузить ранее сохраненную игру.

3. Editor – редактор позволяет создавать новые игровые уровни. После выбора данного пункта необходимо задать размеры игрового поля, после чего можно приступать к редактированию уровня. На игровом поле отображается курсор. На поле, где находится курсор, можно добавлять и удалять объекты. Для добавления стены нажмите кнопку 1, ящика – 2, кладовщика – 3. Для того, чтобы отметить поле как то, на которое нужно поставить ящик, нажмите 4. Для удаления объекта нажмите «Backspace» или «Delete». Текущее состояние можно сохранить с помощью клавиши F2.

4. Exit – выход из программы.

# 5. Тестовый пример

Рассмотрим уровень, представленный в [2]. Создадим его в редакторе, а затем загрузим его в графическом режиме, как показано на рисунке 2, и в консольном, как показано на рисунке 3.

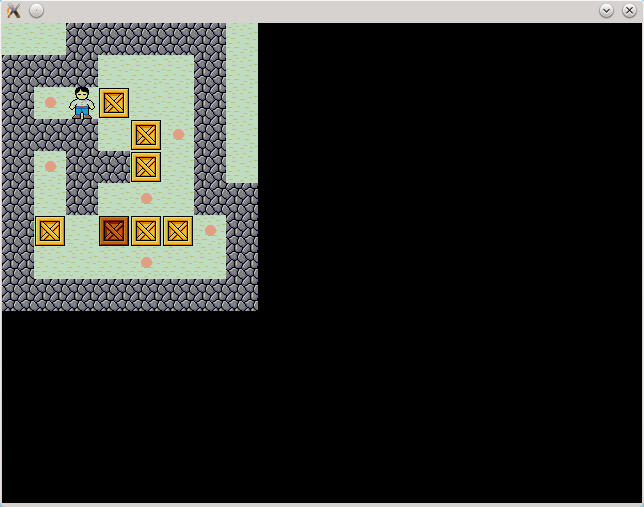


Рисунок 2 – Вид тестового уровня в графическом режиме

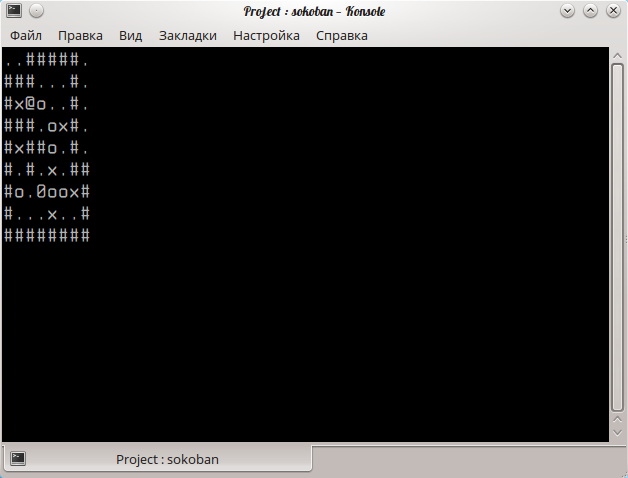


Рисунок 3 – Вид тестового уровня в текстовом режиме

После чего, следуя методике, указанной в [2], решим уровень, сначала в графическом режиме, затем и в текстовом. Вид таблицы рекордов в графическом режиме представлен на рисунке 4, а в текстовом – на рисунке 5.

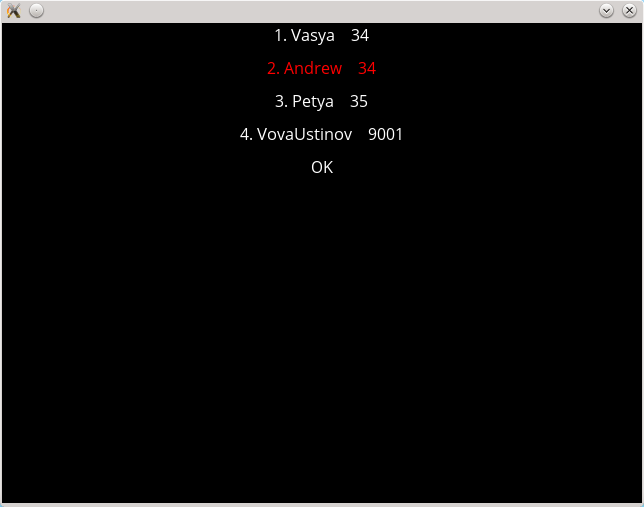


Рисунок 4 – Вид таблицы рекордов в графическом режиме

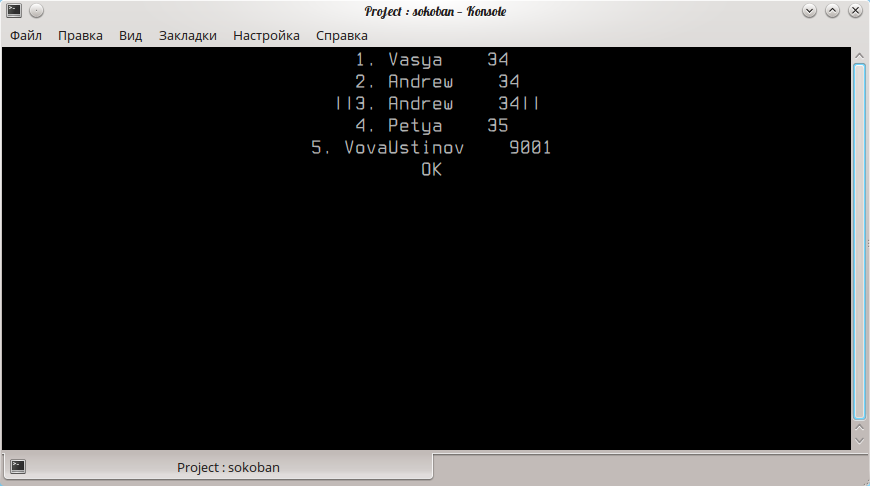


Рисунок 5 – Вид таблицы рекордов в текстовом режиме

Как видно, уровень был пройден за 34 хода.

# Заключение

В данной работе была разработана и реализована логическая игра «Сокобан». Следуя объектно-ориентированной парадигме разработки программного обеспечения, была спроектирована и реализована иерархия классов, моделирующая игровую ситуацию, затем также в объектно-ориентированном стиле была спроектирована и разработана игровая программа, содержащая в себе различные меню, редактор уровней и таблицу рекордов, обладающая консольным и графическим интерфейсом, выполняющаяся под Windows и системами класса UNIX, такими, как Linux.

Отметим, что разработанный набор классов, моделирующих игровую ситуацию «Сокобан» является достаточно общим – допуски на конкретные игровые правила и моменты учитываются лишь в конечных элементах иерархии классов, поэтому данная разработанная библиотека классов позволяет реализовать с минимальными усилиями большое количество двумерных игр на основе игровой доски – шахматы, шашки, «Lines», даже Rogue-like RPG, и многие другие, что раскрывает одну из наиболее привлекательных сторон объектно-ориентированного программирования – широчайшие возможности повторного использования кода.

# Список использованных источников

1. Википедия, Свободная энциклопедия – Sokoban,

http://ru.wikipedia.org/wiki/Sokoban

2. Wikipedia, The free encyclopedia – Sokoban,

http://en.wikipedia.org/wiki/Sokoban

3. Логические игры – Сокобан, <http://logic-games.spb.ru/sokoban/>

4. Лафоре Р. – Объектно-ориентированное программирование в C++. Классика Computer Science. 4-е изд. – СПБ.: Питер, 2012. – 928 с.: ил.

5. Буч, Гради, Максимчук, Роберт Α., Энгл, Майкл У, Янг, Бобби Дж., Коналлен, Джим, Хьюстон, Келли А. – Объектно-ориентированный анализ и проектирование с примерами приложений, 3-е изд.: Пер. с англ. - М.: ООО "И.Д. Вильяме", 2008. – 720 с.: ил.

6. Подбельский В.В. – Язык Си++. Учеб. пособие. -5-е изд. – М.: Финансы и стастика, 2003 - 560с.: ил.