Санкт-Петербургский Политехнический Университет Петра Великого
Институт компьютерных наук и технологий
Кафедра компьютерных систем и программных технологий

Программирование

Отчет по курсовой работе
"Жизнь"

Работу выполнил:
Корсков А.В.
Группа:
13501/4
Преподаватель:
Вылегжанина К.Д.

Санкт-Петербург
2016

Содержание

1 Модель Жизнь

1.1 Задание

Жизнь. Из файла считывается прямоугольное поле, каждая клетка которого либо жива, либо мертва. В очередном поколении, мертвая клетка становится живой, если она имела ровно трех живых соседей, в противном случае остается мертвой. Живая клетка остается живой, если она имела двух или трех живых соседей, в противном случае становится мертвой. Реализовать на экране процесс смены поколений. Программа должна позволять сохранять вид игрового поля для использования его в будущем.

1.2 Правила работы программы

В редакторе последовательно сменяются поколения. Программа проверяет у каждой клетки состояния всех соседних клеток. Если у мёртвой клетки три живых соседа, то клетка поменяет свой статус на живой, иначе останется мёртвой. То же самое происходит и у живой клетки, она остаётся живой если два или три соседа живые, иначе становится мёртвой. Ещё одним принципиальным правилом является наличие функции сохранения в файл и загрузки из файла поля с зафиксированным состояние всех клеток.

1.3 Концепция

Готовый проект должен моделировать превращение живых клеток в мёртвые и наоборот. Эти превращения проходят по определённым правилам. Пользователь должен иметь возможность наблюдать за текущим состоянием поля и превращением клеток. Также важной функцией программы является возможность сохранить текущее состояние поля в файл и загрузка поля из файла.

1.4 Минимально работоспособный продукт

Минимально работоспособный продукт должен уметь: предоставить пользователю информацию о текущем состоянии клеток, сохранение и загрузка поля в файл.

1.5 Диаграмма прецедентов использования

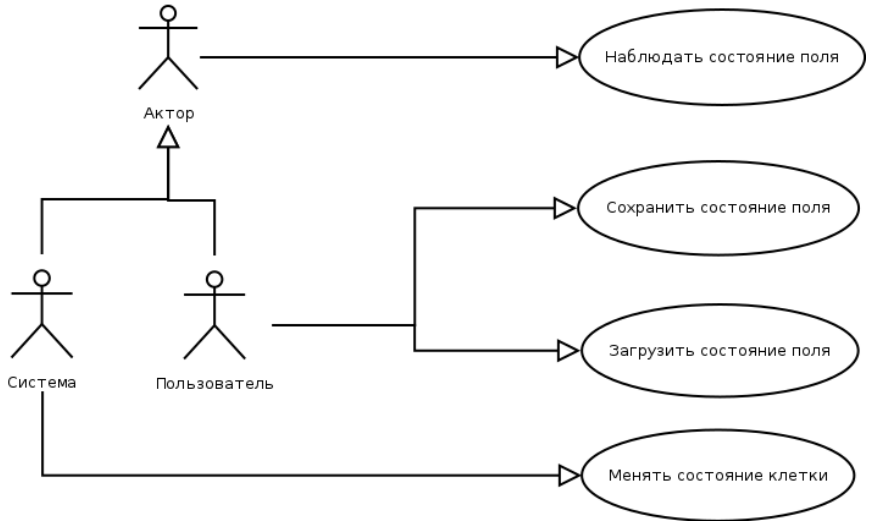

Рис. 1: Диаграмма прецедентов использования

2 Проектрование приложения, реализующего модель Жизнь

Программа разделена на 4 подпроекта: app - консольное приложение, core - библиотека, реализующая модель Жизнь, gui - графическое приложение, test - тесты для программы.

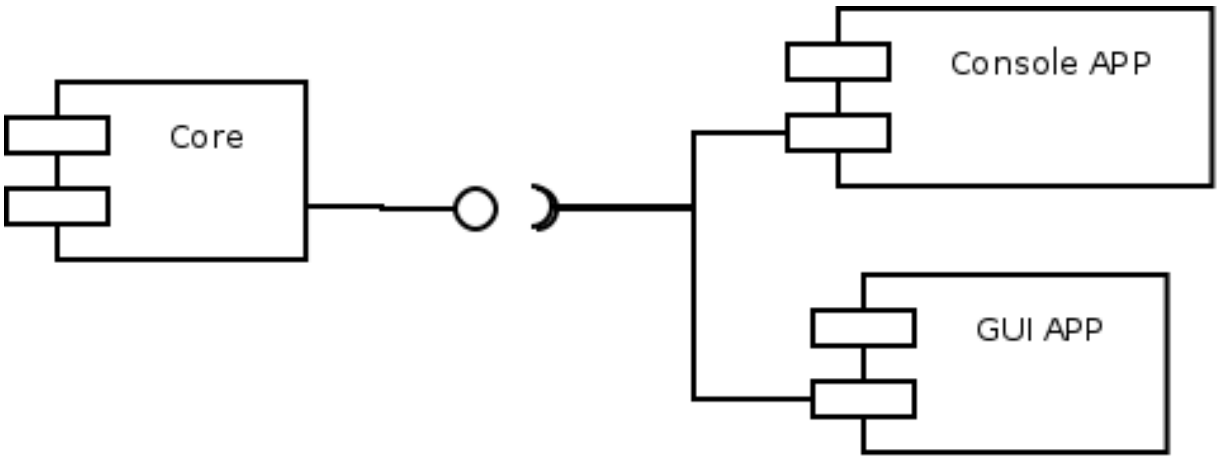

Рис. 2: Диаграмма прецедентов использования

2.1 Библиотека

При написании проекта, была создана библиотека. В ней находятся все необходимые классы для создания и работы модели. Один из классов (api) создан для предоставления всех действий над моделью.
В API выделены следующие методы:

- initialize_settings - метод, задающий размеры поля.
- initialize_field - метод, инициализирующий поле случайными клетками.
- save_field - метод, сохраняющий поле в файл.
- load_field - метод, загружающий поле из файла.
- print_field - метод, выводящий поле в консоль.
- change_field - метод, меняющий поколение на следующее.
- set_cell - метод, задающий клетку на поле.

3 Реализация модели Жизнь

3.1 Версии программ

Операционная система: Debian 8.1, среда разработки: Qt Creator 3.5.0, компилятор: GCC 4.9.1, система документирования: Doxygen 1.8.8, Qt 5.5.0.

3.2 Консольное приложение

Консольное приложение позволяет работать с моделью через консоль. Основные классы, выделенные в консольном приложении:

- Класс console_ui. Сначала выводит главное меню, где можно создать модель или загрузить. Также есть метод выводящий второе меню, где можно запустить следующее поколение или сохранить модель и в третьем меню можно задать размер поля.

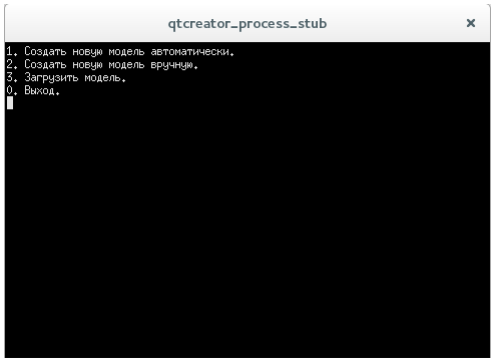

Рис. 3: Главное меню консольного приложения

На рис 3 представлено главное меню приложения. Есть возможность создать новую модель случайными клетками, создать модель вручную, загрузить модель из файла и выйти из программы.

Рис. 4: Поле и второе меню в консольном приложении

На рис 4 показано поле и внизу меню, в котором можно сменить поколение, сохранить поле или вернуться в главное меню.

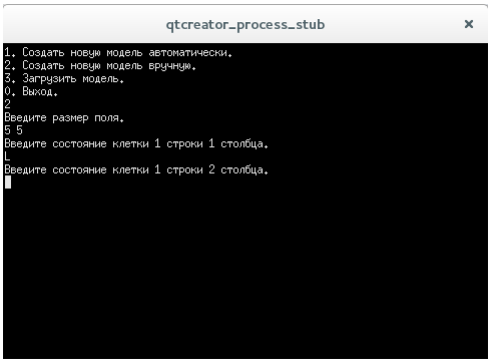

Рис. 5: Поле модели в консольном приложении

На рис 5 показано главное меню ниже расположено меню размера поля и в самом низу два меню ввода состояния клеток.

3.3 Библиотека

Основные классы, выделенные в библиотеке:

- Класс Cell. Реализует клетку. Содержит координаты клетки и её состояние. Присутствуют методы, возвращающие и задающие координаты и состояние клетки. Также есть метод, проверяющий сколько живых соседей у данной клетки.
- Класс Field. Класс представляет поле модели. Содержит размер поля и двумерный массив клеток. Присутствуют методы, возвращающие и задающие размер поля, отдельную клетку и весь двумерный массив клеток.
- Класс Api. Класс, предоставляющий все методы, доступные над моделью. Позволяет задать размер поля, инициализировать поле случайными клетками, задать отдельную клетку, сохранить и загрузить поле из файла и вывести его.

3.4 Графическое приложение

Графическое приложение позволяет работать с моделью через окна графического приложения.
Основные классы, выделенные в графическом приложении.

- Класс MainWindow. Главное окно приложения. Присутствуют кнопки «Создать новую случайную модель», «Создать новую модель вручную», «Загрузить модель», «Сделать модель с фигурой», «Выход».
- Класс exit_window. Окно для полтверждения выхода.
- Класс field_window. Окно отображающее поле. Присутствуют кнопки «Следующее поколение», «Сохранить модель», «Назад в главное меню».
- Класс figure_window. Окно позволяющее выбрать готовое поле. Присутствуют кнопки «Глайдер» и «Космический корабль».
- Класс initialize_window. Окно позволяющее выбрать состояние поле. Присутствуют кнопки «Живая» и «Мёртвая».
- Класс size_window. Окно позволяющее задать размер поля.

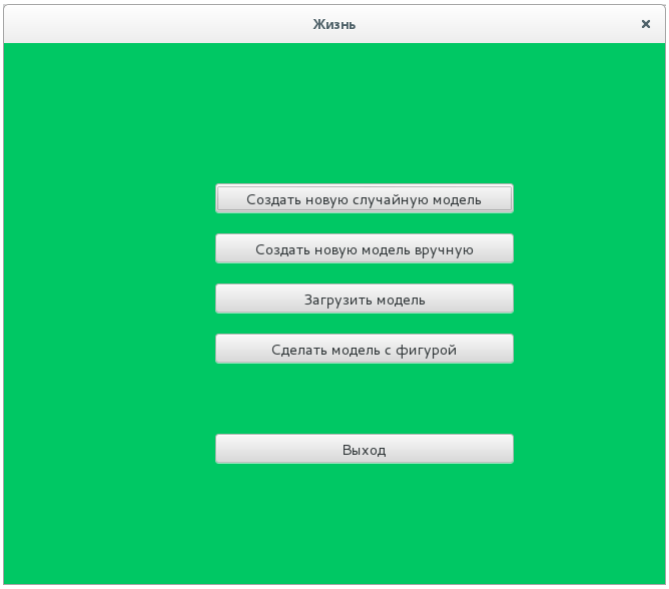

Рис. 6: Главное меню графического приложения

На рис 6 представлено главное окно приложения. В нём пользователю можно создать поле случайными клетками, вручную, загрузить из файла, создать поле с готовыми фигурами или выйти из программы.

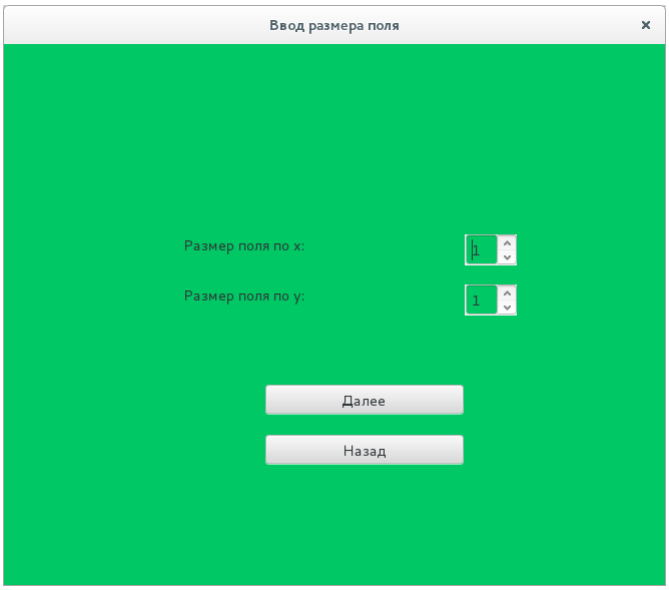

Рис. 7: Настройки модели в графическом приложении

На рис 7 – окно выбора размера приложения.

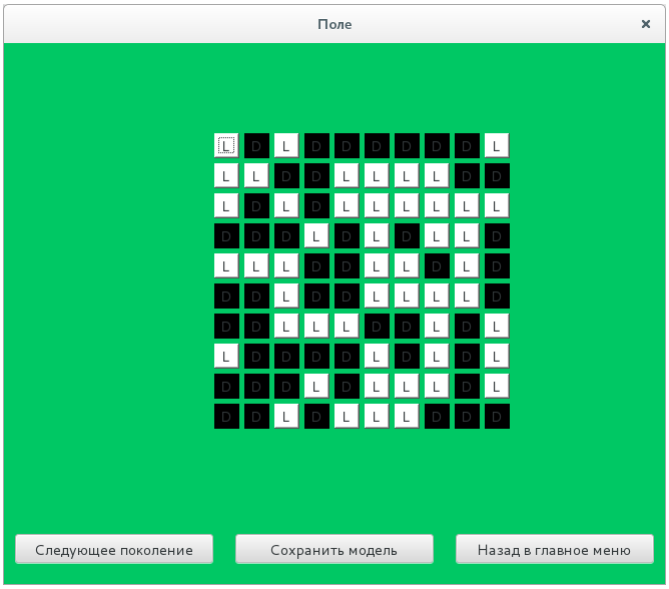

Рис. 8: Представление поля в графическом приложении

На рис 8 – окно с полем, внизу есть кнопки для запуска следующего поколения, сохранения модели и выхода в главное меню.

4 Процесс обеспечения качества и тестирование

4.1 Тестирование

Приложение содержит автоматические тесты. Протестированы некоторые основные функции. Проверяется назначение поля размера, заполнение его случайными клетками и правильность получения следующего поколения.

5 Вывод

По окончании семестра автор проекта научился делать графический интерфейс с помощью Qt, а также получил опыт работы с большими проектами, содержащими много классов и имеющих как консольное приложение, так и графическое.

6 Приложение 1. Листинги кода

6.1 Консольное приложение

```
1  #include <iostream>
2  #include "console_ui.h"
3
4  using namespace std;
5
6  int main()
7  {
8      console_ui c_u;
9      c_u.print_menu();
10     return 0;
11 }
```

```
#ifndef CONSOLE_UI_H
#define CONSOLE_UI_H
#include "api.h"
#include "field.h"

/**
   @brief Класс консольного меню

   Этот класс, посредством консоли, взаимодействует с пользователем.
   */

class console_ui
{
public:
    /**
     * @brief Вывести главное меню
     */
    void print_menu();
    /**
     * @brief Вывести промежуточное меню
     * @param f поле
     * @param a объект класса Api
     */
    void secondary_menu(Field &f, Api a);
    /**
     * @brief Вывести меню задания клеток поля
     * @param f поле
     * @param a объект класса Api
     * @param x размер поля по оси x
     * @param y размер поля по оси y
     */
    void input_menu(Field &f, Api a, int x, int y);
};

#endif // CONSOLE_UI_H
```

```
#include "console_ui.h"
#include <iostream>
#include <fstream>
#include "string"

void console_ui::print_menu()
{
    int choice = -1;
    Api a;
    Field f;
    while(choice != 0) {
        std::cout << "1. Создать новую модель автоматически." << std::
    ↪ endl;
        std::cout << "2. Создать новую модель вручную." << std::endl;
        std::cout << "3. Загрузить модель." << std::endl;
        std::cout << "0. Выход." << std::endl;
        std::cin >> choice;
        int x,y;
        switch (choice)
        {
            case 1:
                std::cout << "Введите размер поля." << std::endl;
                std::cin >> x >> y;
                a.initialize_settings(f,x,y);
                a.initialize_field(f);
```

```
25                  a.print_field(f);
26                  this->secondary_menu(f,a);
27              break;
28              case 2:
29                  std::cout << "Введите размер поля." << std::endl;
30                  std::cin >> x >> y;
31                  a.initialize_settings(f,x,y);
32                  input_menu(f,a,x,y);
33                  a.print_field(f);
34                  this->secondary_menu(f,a);
35              break;
36              case 3:
37                  a.load_field(f);
38                  a.print_field(f);
39                  std::cout << "Модель успешно загружена." << std::endl
    ↪ ;
40                  this->secondary_menu(f,a);
41              break;
42              case 0:
43              break;
44              default:
45                  std::cout << "Некорректная команда." << std::endl;
46              break;
47          }
48      }
49  }
50
51  void console_ui::secondary_menu(Field &f, Api a)
52  {
53      int choice = -1;
54      while (choice != 0) {
55          std::cout << "1. Следующее поколение." << std::endl;
56          std::cout << "2. Сохранить модель." << std::endl;
57          std::cout << "0. Назад в главное меню." << std::endl;
58          std::cin >> choice;
59          switch (choice) {
60              case 1:
61                  a.change_field(f);
62                  a.print_field(f);
63                  this->secondary_menu(f,a);
64              break;
65              case 2:
66                  a.save_field(f);
67                  std::cout << "Модель успешно сохранена." << std::endl
    ↪ ;
68                  this->secondary_menu(f,a);
69              break;
70              case 0:
71              break;
72              default:
73                  std::cout << "Некорректная команда." << std::endl;
74                  a.print_field(f);
75                  this->secondary_menu(f,a);
76              break;
77          }
78          break;
79      }
80  }
81
82  void console_ui::input_menu(Field &f, Api a, int x, int y)
83  {
84      std::string status;
```

```
85      for (int i = 0; i < x; i++){
86          for (int j = 0; j < y; j++){
87              std::cout << "Введите состояние клетки " << i+1 << "
    ↪ строки " << j+1 << " столбца." << std::endl;
88              std::cin >> status;
89              a.set_cell(f,i,j,status);
90          }
91      }
92  }
```

6.2 Графическое приложение

```
#include "mainwindow.h"
#include <QApplication>
#include "field.h"
#include "api.h"

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainWindow w(0);
    w.show();

    return a.exec();
}
```

```
#ifndef EXIT_WINDOW_H
#define EXIT_WINDOW_H

#include <QPushButton>
#include <QLabel>
#include "mainwindow.h"
#include <QtWidgets>


class exit_window : public QDialog
{
    Q_OBJECT

    const QSize WINDOW_SIZE { 300, 90 };
    const QSize BUTTON_SIZE { 120, 30 };
    QPushButton* yes_button;
    QPushButton* no_button;
    QWidget* parent;
    QLabel* exit_label;
public:
    explicit exit_window(QWidget* parent);
private slots:
    void close_app();
    void close_exit_window();
};

#endif // EXIT_WINDOW_H
```

```
#include "exit_window.h"

exit_window::exit_window(QWidget* parent) : QDialog(parent)
{
    this->parent = parent;
    this->setFixedSize(WINDOW_SIZE);
    this->setWindowTitle("Подтверждение выхода");
    QPalette pal;
    pal.setColor(QPalette::Background, QColor(0, 200, 100, 255));
    this->setPalette(pal);
    exit_label =  new QLabel(this);
    exit_label->move(WINDOW_SIZE.width() - 283, WINDOW_SIZE.height
    ↪ () - 80);
    exit_label->setText("Вы действительно хотите выйти?");
    exit_label->show();
    yes_button = new QPushButton("Да", this);
    yes_button->resize(BUTTON_SIZE);
    yes_button->move(WINDOW_SIZE.width() - 275, WINDOW_SIZE.height
    ↪ () - 50);
    connect(yes_button, SIGNAL(clicked()), SLOT(close_app()));
    no_button = new QPushButton("Нет", this);
    no_button->resize(BUTTON_SIZE);
    no_button->move(WINDOW_SIZE.width() - 125, WINDOW_SIZE.height()
    ↪ - 50);
    connect(no_button, SIGNAL(clicked()), SLOT(close_exit_window())
    ↪ );
}

void exit_window::close_app()
{
    this->close();
    this->parent->close();
}
```

```
30
31 void exit_window::close_exit_window()
32 {
33     this->close();
34 }
```

```
#ifndef FIELD_WINDOW_H
#define FIELD_WINDOW_H

#include <QWidget>
#include "mainwindow.h"
#include "api.h"
#include "field.h"

class field_window : public QWidget
{
    Q_OBJECT

    QWidget* parent;
    Api a;
    Field f;
    const QSize WINDOW_SIZE { 660, 540 };
    const QSize BUTTON_SIZE { 200, 30 };
    const QSize CELL_SIZE { 25, 25 };
    QPushButton* next_generation_button;
    QPushButton* back_button;
    QPushButton* save_button;
public:
    field_window(QWidget* parent,Api a,Field f);
    void print_field();
private slots:
    void next_generation();
    void back_in_main_menu();
    void save_field();
};

#endif // FIELD_WINDOW_H
```

```
#include "field_window.h"

field_window::field_window(QWidget *parent, Api a, Field f) :
    ↪ QWidget(parent)
{
    this->a = a;
    this->f = f;
    this->setFixedSize(WINDOW_SIZE);
    this->setWindowTitle("Поле");
    QPalette pal;
    pal.setColor(QPalette::Background, QColor(0, 200, 100, 255));
    this->setPalette(pal);
    this->print_field();
    next_generation_button = new QPushButton("Следующее поколение",
    ↪ this);
    next_generation_button->resize(BUTTON_SIZE);
    next_generation_button->move(WINDOW_SIZE.width() - 650,
    ↪ WINDOW_SIZE.height() - 50);
    connect(next_generation_button, SIGNAL(clicked()), SLOT(
    ↪ next_generation()));
    save_button = new QPushButton("Сохранить модель", this);
    save_button->resize(BUTTON_SIZE);
    save_button->move(WINDOW_SIZE.width() - 430, WINDOW_SIZE.height
    ↪ () - 50);
    connect(save_button, SIGNAL(clicked()), SLOT(save_field()));
    back_button = new QPushButton("Назад в главное меню", this);
    back_button->resize(BUTTON_SIZE);
    back_button->move(WINDOW_SIZE.width() - 210, WINDOW_SIZE.height
    ↪ () - 50);
```

```
24      connect(back_button, SIGNAL(clicked()), SLOT(back_in_main_menu
    ↪ ()));
25  }
26
27  void field_window::print_field()
28  {
29      for (int i = 0; i < f.get_x(); i++){
30          for (int j = 0; j < f.get_y(); j++){
31              Cell c = f.get_cell(i,j);
32              bool status = c.get_status();
33              if (status == 0){
34                  QPushButton* cell = new QPushButton("D",this);
35                  cell->resize(CELL_SIZE);
36                  cell->move(WINDOW_SIZE.width() - 450+(30*i),
    ↪ WINDOW_SIZE.height() - 450+(30*j));
37                  cell->setStyleSheet("background-color: black;");
38              }else{
39                  QPushButton* cell = new QPushButton("L",this);
40                  cell->resize(CELL_SIZE);
41                  cell->move(WINDOW_SIZE.width() - 450+(30*i),
    ↪ WINDOW_SIZE.height() - 450+(30*j));
42                  cell->setStyleSheet("background-color: white;");
43              }
44          }
45      }
46  }
47
48  void field_window::next_generation()
49  {
50      a.change_field(f);
51      field_window* field = new field_window(0,a,f);
52      field->show();
53      this->close();
54  }
55
56  void field_window::back_in_main_menu()
57  {
58      MainWindow* w = new MainWindow(0);
59      w->show();
60      this->close();
61  }
62
63  void field_window::save_field()
64  {
65      a.save_field(f);
66  }
```

```
#ifndef FIGURE_WINDOW_H
#define FIGURE_WINDOW_H

#include <QPushButton>
#include <QLabel>
#include "mainwindow.h"
#include "api.h"
#include "field.h"
#include <QtWidgets>

class figure_window : public QDialog
{
    Q_OBJECT

    Api a;
    Field f;
    const QSize WINDOW_SIZE { 430, 430 };
    const QSize BUTTON_SIZE { 160, 30 };
    QPushButton* glider_button;
    QPushButton* spaceship_button;
    QWidget* parent;
    QLabel* figure_label;
public:
    explicit figure_window(QWidget* parent);
private slots:
    void create_spaceship();
    void create_glider();
};

#endif // FIGURE_WINDOW_H
```

```
#include "figure_window.h"

figure_window::figure_window(QWidget* parent) : QDialog(parent)
{
    this->parent = parent;
    this->setFixedSize(WINDOW_SIZE);
    this->setWindowTitle("Выбор фигуры");
    QPalette pal;
    pal.setColor(QPalette::Background, QColor(0, 200, 100, 255));
    this->setPalette(pal);
    figure_label =  new QLabel(this);
    figure_label->move(WINDOW_SIZE.width() - 283, WINDOW_SIZE.
    ↪ height()  - 380);
    figure_label->setText("Выберете фигуру");
    figure_label->show();
    glider_button = new QPushButton("Глайдер", this);
    glider_button->resize(BUTTON_SIZE);
    glider_button->move(WINDOW_SIZE.width() - 300, WINDOW_SIZE.
    ↪ height() - 350);
    connect(glider_button, SIGNAL(clicked()), SLOT(create_glider())
    ↪ );
    spaceship_button = new QPushButton("Космический корабль", this);
    spaceship_button->resize(BUTTON_SIZE);
    spaceship_button->move(WINDOW_SIZE.width() - 300, WINDOW_SIZE.
    ↪ height() - 300);
    connect(spaceship_button, SIGNAL(clicked()), SLOT(
    ↪ create_spaceship()));
}

void figure_window::create_spaceship()
```

```
26 {
27     a.initialize_settings(f,10,10);
28     a.set_cell(f,0,0,"L");
29     a.set_cell(f,0,1,"D");
30     a.set_cell(f,0,2,"L");
31     for (int i = 3; i < 10; i++){
32         a.set_cell(f,0,i,"D");
33     }
34     for (int i = 0; i < 3; i++){
35         a.set_cell(f,1,i,"D");
36     }
37     a.set_cell(f,1,3,"L");
38     for (int i = 4; i < 10; i++){
39         a.set_cell(f,1,i,"D");
40     }
41     for (int i = 0; i < 3; i++){
42         a.set_cell(f,2,i,"D");
43     }
44     a.set_cell(f,2,3,"L");
45     for (int i = 4; i < 10; i++){
46         a.set_cell(f,2,i,"D");
47     }
48     a.set_cell(f,3,0,"L");
49     a.set_cell(f,3,1,"D");
50     a.set_cell(f,3,2,"D");
51     a.set_cell(f,3,3,"L");
52     for (int i = 4; i < 10; i++){
53         a.set_cell(f,3,i,"D");
54     }
55     a.set_cell(f,4,0,"D");
56     a.set_cell(f,4,1,"L");
57     a.set_cell(f,4,2,"L");
58     a.set_cell(f,4,3,"L");
59     for (int i = 4; i < 10; i++){
60         a.set_cell(f,4,i,"D");
61     }
62     for (int i = 5; i < 10; i++){
63         for (int j = 0; j < 10; j++){
64             a.set_cell(f,i,j,"D");
65         }
66     }
67     field_window* field = new field_window(0,a,f);
68     field->show();
69     this->close();
70     this->parent->close();
71 }
72
73 void figure_window::create_glider()
74 {
75     a.initialize_settings(f,10,10);
76     a.set_cell(f,0,0,"L");
77     for (int i = 1; i < 10; i++){
78         a.set_cell(f,0,i,"D");
79     }
80     a.set_cell(f,1,0,"D");
81     a.set_cell(f,1,1,"L");
82     a.set_cell(f,1,2,"L");
83     for (int i = 3; i < 10; i++){
84         a.set_cell(f,1,i,"D");
85     }
86     a.set_cell(f,2,0,"L");
87     a.set_cell(f,2,1,"L");
```

```
88      for (int i = 2; i < 10; i++){
89          a.set_cell(f,2,i,"D");
90      }
91      for (int i = 3; i < 10; i++){
92          for (int j = 0; j < 10; j++){
93              a.set_cell(f,i,j,"D");
94          }
95      }
96      field_window* field = new field_window(0,a,f);
97      field->show();
98      this->close();
99      this->parent->close();
100 }
```

```
#ifndef INITIALIZE_WINDOW_H
#define INITIALIZE_WINDOW_H

#include <QWidget>
#include <QPushButton>
#include <QLabel>
#include <QtWidgets>
#include "api.h"
#include "field.h"

class initialize_window : public QDialog
{
    Q_OBJECT

    QWidget* parent;
    Api* a;
    Field* f;
    int x;
    int y;
    const QSize WINDOW_SIZE { 430, 220 };
    const QSize BUTTON_SIZE { 200, 30 };
    QLabel* status_cell;
    QPushButton* live_button;
    QPushButton* dead_button;
public:
    explicit initialize_window(QWidget* parent, Api* a, Field* f,
    ↪ int i, int j);
private slots:
    void clicked_button_live();
    void clicked_button_dead();
};

#endif // INITIALIZE_WINDOW_H
```

```
#include "initialize_window.h"

initialize_window::initialize_window(QWidget *parent, Api *a, Field
    ↪  *f, int i, int j) : QDialog(parent)
{
    this->parent = parent;
    this->f = f;
    this->a = a;
    this->x = i;
    this->y = j;
    this->setFixedSize(WINDOW_SIZE);
    this->setWindowTitle("Выбор состояния клетки");
    QPalette pal;
    pal.setColor(QPalette::Background, QColor(0, 200, 100, 255));
    this->setPalette(pal);
    status_cell =  new QLabel(this);
    status_cell->move(WINDOW_SIZE.width() - 385, WINDOW_SIZE.height
    ↪ () - 160);
    status_cell->setText(QString("Выберете состояние клетки %1 строки,
    ↪ %2 столбца").arg(y+1).arg(x+1));
    status_cell->show();
    live_button = new QPushButton("Живая", this);
    live_button->resize(BUTTON_SIZE);
    live_button->move(WINDOW_SIZE.width() - 325, WINDOW_SIZE.height
    ↪ () - 50);
    connect(live_button, SIGNAL(clicked()), SLOT(
    ↪ clicked_button_live()));
```

```
23      dead_button = new QPushButton("Мёртвая", this);
24      dead_button->resize(BUTTON_SIZE);
25      dead_button->move(WINDOW_SIZE.width() - 325, WINDOW_SIZE.height
        ↪ () - 100);
26      connect(dead_button, SIGNAL(clicked()), SLOT(
        ↪ clicked_button_dead()));
27  }
28
29  void initialize_window::clicked_button_live()
30  {
31      a->set_cell(*f,x,y,"L");
32      this->close();
33  }
34
35  void initialize_window::clicked_button_dead()
36  {
37      a->set_cell(*f,x,y,"D");
38      this->close();
39  }
```

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QWidget>
#include <QPushButton>
#include <QInputDialog>
#include "size_window.h"
#include "exit_window.h"
#include "figure_window.h"
#include "field.h"
#include "api.h"

class MainWindow : public QWidget
{
    Q_OBJECT

    const QSize WINDOW_SIZE { 660, 540 };
    const QSize BUTTON_SIZE { 300, 30 };
    QPushButton* new_random_model_button;
    QPushButton* load_button;
    QPushButton* figure_button;
    QPushButton* new_model_button;
    QPushButton* exit_button;
public:
    MainWindow(QWidget *parent);
private slots:
    void figure_menu();
    void close_menu();
    void create_model_random();
    void create_model();
    void load_model();
};

#endif // MAINWINDOW_H
```

```
#include "mainwindow.h"

MainWindow::MainWindow(QWidget *parent)
    : QWidget(parent)
{
    this->setFixedSize(WINDOW_SIZE);
    this->setWindowTitle("Жизнь");
    QPalette pal;
    pal.setColor(QPalette::Background, QColor(0, 200, 100, 255));
    this->setPalette(pal);
    new_random_model_button = new QPushButton("Создать новую
    ↪ случайную модель", this);
    new_random_model_button->resize(BUTTON_SIZE);
    new_random_model_button->move(WINDOW_SIZE.width() - 450,
    ↪ WINDOW_SIZE.height() - 400);
    connect(new_random_model_button, SIGNAL(clicked()), SLOT(
    ↪ create_model_random()));
    load_button = new QPushButton("Загрузить модель", this);
    load_button->resize(BUTTON_SIZE);
    load_button->move(WINDOW_SIZE.width() - 450,WINDOW_SIZE.height
    ↪ () - 300);
    connect(load_button, SIGNAL(clicked()), SLOT(load_model()));
    new_model_button = new QPushButton("Создать новую модель вручную"
    ↪ , this);
    new_model_button->resize(BUTTON_SIZE);
```

```
21      new_model_button->move(WINDOW_SIZE.width() - 450,WINDOW_SIZE.
        ↪ height() - 350);
22      connect(new_model_button, SIGNAL(clicked()), SLOT(create_model
        ↪ ()));
23      figure_button = new QPushButton("Сделать модель с фигурой", this)
        ↪ ;
24      figure_button->resize(BUTTON_SIZE);
25      figure_button->move(WINDOW_SIZE.width() - 450,WINDOW_SIZE.
        ↪ height() - 250);
26      connect(figure_button, SIGNAL(clicked()), SLOT(figure_menu()));
27      exit_button = new QPushButton("Выход", this);
28      exit_button->resize(BUTTON_SIZE);
29      exit_button->move(WINDOW_SIZE.width() - 450, WINDOW_SIZE.height
        ↪ () - 150);
30      connect(exit_button, SIGNAL(clicked()), SLOT(close_menu()));
31  }
32
33  void MainWindow::figure_menu()
34  {
35      figure_window* figure = new figure_window(this);
36      figure->exec();
37      delete figure;
38  }
39
40  void MainWindow::close_menu()
41  {
42      exit_window* exit_menu = new exit_window(this);
43      exit_menu->exec();
44      delete exit_menu;
45  }
46
47  void MainWindow::create_model_random()
48  {
49      size_window* size_menu = new size_window(0,1);
50      size_menu->show();
51      this->close();
52  }
53
54  void MainWindow::create_model()
55  {
56      size_window* size_menu = new size_window(0,2);
57      size_menu->show();
58      this->close();
59  }
60
61  void MainWindow::load_model()
62  {
63      Api a;
64      Field f;
65      a.load_field(f);
66      field_window* field = new field_window(0,a,f);
67      field->show();
68      this->close();
69  }
```

```
#ifndef SIZE_WINDOW_H
#define SIZE_WINDOW_H

#include <QLabel>
#include <QSpinBox>
#include "mainwindow.h"
#include "field_window.h"
#include "field.h"
#include "api.h"
#include "initialize_window.h"

class size_window : public QWidget
{
    Q_OBJECT

    QWidget* parent;
    Api a;
    Field f;
    int mode;
    const QSize WINDOW_SIZE { 660, 540 };
    const QSize BUTTON_SIZE { 200, 30 };
    QPushButton* next_button;
    QPushButton* back_button;
    QLabel* size_field_x;
    QLabel* size_field_y;
    QSpinBox* size_Field_x;
    QSpinBox* size_Field_y;
    QLabel* create_label(QString text, int coordinate_x, int
    ↪ coordinate_y);
    QSpinBox* create_spin_box(int min, int max, int coordinate_x,
    ↪ int coordinate_y);
public:
    size_window(QWidget* parent, int mode);
private slots:
    void print_field();
    void close_size_window();
};

#endif // SIZE_WINDOW_H
```

```
#include "size_window.h"

size_window::size_window(QWidget* parent, int mode) : QWidget(
    ↪ parent)
{
    this->mode = mode;
    this->setFixedSize(WINDOW_SIZE);
    this->setWindowTitle("Ввод размера поля");
    QPalette pal;
    pal.setColor(QPalette::Background, QColor(0, 200, 100, 255));
    this->setPalette(pal);
    size_field_x = create_label("Размер поля по x:", WINDOW_SIZE.
    ↪ width() - 480, WINDOW_SIZE.height() - 350);
    size_Field_x = create_spin_box(1, 10, WINDOW_SIZE.width() -
    ↪ 200, WINDOW_SIZE.height() - 350);
    size_field_y = create_label("Размер поля по y:", WINDOW_SIZE.
    ↪ width() - 480, WINDOW_SIZE.height() - 300);
    size_Field_y = create_spin_box(1, 10, WINDOW_SIZE.width() -
    ↪ 200, WINDOW_SIZE.height() - 300);
    next_button = new QPushButton("Далее", this);
    next_button->resize(BUTTON_SIZE);
```

```
17      next_button->move(WINDOW_SIZE.width() - 400, WINDOW_SIZE.height
        ↪ () - 200);
18      connect(next_button, SIGNAL(clicked()), SLOT(print_field()));
19      back_button = new QPushButton("Назад", this);
20      back_button->resize(BUTTON_SIZE);
21      back_button->move(WINDOW_SIZE.width() - 400, WINDOW_SIZE.height
        ↪ () - 150);
22      connect(back_button, SIGNAL(clicked()), SLOT(close_size_window
        ↪ ()));
23  }
24
25  void size_window::print_field()
26  {
27      a.initialize_settings(f,size_Field_x->value(),size_Field_y->
        ↪ value());
28      if (mode == 1){
29          a.initialize_field(f);
30      }else{
31          for (int i = 0; i < f.get_x(); i++){
32              for (int j = 0; j < f.get_y(); j++){
33                  initialize_window* init = new initialize_window(
        ↪ this, &a, &f, i, j);
34                  init->exec();
35                  delete init;
36              }
37          }
38      }
39      field_window* field = new field_window(0,a,f);
40      field->show();
41      this->close();
42  }
43
44  QLabel *size_window::create_label(QString text, int coordinate_x,
        ↪ int coordinate_y)
45  {
46      QLabel* label = new QLabel(this);
47      label->move(coordinate_x, coordinate_y);
48      label->setText(text);
49      label->show();
50      return label;
51  }
52
53  QSpinBox* size_window::create_spin_box(int min, int max, int
        ↪ coordinate_x, int coordinate_y)
54  {
55      QSpinBox* spin_box = new QSpinBox(this);
56      spin_box->setRange(min, max);
57      spin_box->move(coordinate_x, coordinate_y);
58      spin_box->show();
59      return spin_box;
60  }
61
62  void size_window::close_size_window()
63  {
64      MainWindow* w = new MainWindow(0);
65      w->show();
66      this->close();
67  }
```

6.3 Библиотека

```
#ifndef API_H
#define API_H
#include "field.h"
#include "string"

/**
   @brief Класс Api

  Этот класс предоставляет методы ядра.
  */

class Api
{
public:
    /**
     * @brief Задать размеры поля
     * @param f поле
     * @param f_x размер поля по оси x
     * @param f_y размер поля по оси y
     */
    void initialize_settings(Field &f, int f_x, int f_y);
    /**
     * @brief Инициализировать поле случайными клетками
     * @param f поле
     */
    void initialize_field(Field &f);
    /**
     * @brief Сохранить поле в файл
     * @param f поле
     */
    void save_field(Field &f);
    /**
     * @brief Загрузить поле из файла
     * @param f поле
     */
    void load_field(Field &f);
    /**
     * @brief Вывести поле в консоль
     * @param f поле
     */
    void print_field(Field &f);
    /**
     * @brief Сменить поколение и изменить поле
     * @param f поле
     */
    void change_field(Field &f);
    /**
     * @brief Задать клетку поля
     * @param f поле
     * @param x координата клетки по оси x
     * @param y координата клетки по оси y
     * @param status статус клетки
     */
    void set_cell(Field &f, int x, int y, std::string status);
};

#endif // API_H
```

```
#include "api.h"
#include <iostream>
```

```
3 #include <stdlib.h>
4 #include <time.h>
5 #include <iostream>
6 #include <fstream>
7
8 void Api::initialize_settings(Field &f, int f_x, int f_y)
9 {
10     f.set_x(f_x);
11     f.set_y(f_y);
12 }
13
14 void Api::initialize_field(Field &f){
15     srand(time(0));
16     for (int i = 0; i < f.get_x(); i++){
17         for (int j = 0; j < f.get_y(); j++){
18             f.set_cell(i,j,rand()%2);
19         }
20     }
21 }
22
23 void Api::save_field(Field &f)
24 {
25     std::ofstream file;
26     file.open("save.txt");
27     file << f.get_x() << " " << f.get_y() << std::endl;
28     for (int i = 0; i < f.get_x(); i++){
29         for (int j = 0; j < f.get_y(); j++){
30             Cell c = f.get_cell(i,j);
31             if (c.get_status() == true){
32                 file << "L ";
33             }else{
34                 file << "D ";
35             }
36         }
37         file << std::endl;
38     }
39     file.close();
40 }
41
42 void Api::load_field(Field &f)
43 {
44     std::ifstream file;
45     file.open("save.txt");
46     int x,y;
47     file >> x >> y;
48     f.set_x(x);
49     f.set_y(y);
50     std::string st;
51     for (int i = 0; i < x; i++){
52         for (int j = 0; j < y; j++){
53             file >> st;
54             if (st == "D"){
55                 f.set_cell(i,j,0);
56             }else{
57                 f.set_cell(i,j,1);
58             }
59         }
60     }
61     file.close();
62 }
63
64 void Api::print_field(Field &f)
```

```
65 {
66     for (int i = 0; i < f.get_x(); i++){
67         for (int j = 0; j < f.get_y(); j++){
68             Cell c = f.get_cell(i,j);
69             switch (c.get_status()) {
70             case false:
71                 std::cout<<"D ";
72                 break;
73             case true:
74                 std::cout<<"L ";
75                 break;
76             }
77         }
78         std::cout<<std::endl;
79     }
80 }
81
82 void Api::change_field(Field &f){
83     int x = f.get_x(), y = f.get_y();
84     Field new_f(x,y);
85     new_f.set_cells(f.get_cells());
86     for (int i = 0; i < x; ++i){
87         for (int j = 0; j < y; ++j){
88             Cell c = f.get_cell(i,j);
89             int number = c.search_living(f.get_cells());
90             if ((c.get_status() == 0) && (number == 3)){
91                 new_f.set_cell(i,j,1);
92             }else{
93             if ((c.get_status() == 1) && ((number == 3) || (number
    ↪ == 2))){
94                 new_f.set_cell(i,j,1);
95             }else{
96                 new_f.set_cell(i,j,0);
97             }
98             }
99         }
100     }
101     f.set_cells(new_f.get_cells());
102 }
103
104 void Api::set_cell(Field &f, int x, int y, std::string status)
105 {
106     if (status == "D"){
107         f.set_cell(x,y,0);
108     }else{
109         f.set_cell(x,y,1);
110     }
111 }
```

```
#ifndef CELL_H
#define CELL_H
#include <vector>

/**
   @brief Класс Клетка

   Этот класс моделирует объект клетка из которых будет состоять поле.
   */

class Cell
{
public:
    /**
     * @brief Конструктор
     * @param a координата по оси x
     * @param b координата по оси y
     * @param st состояние клетки
     */
    Cell(const int a = 0, const int b = 0, const bool st = false):
    ↪ x(a), y(b), status(st) {}
    /**
     * @brief Получить координату по оси x
     * @return координата по оси x
     */
    int get_x() {return x;}
    /**
     * @brief Получить координату по оси y
     * @return координата по оси y
     */
    int get_y() {return y;}
    /**
     * @brief Получить состояние клетки
     * @return состояние клетки
     */
    bool get_status() {return status;}
    /**
     * @brief Установить значение поля x, равное координате по оси x
     * @param a координата по оси x
     */
    void set_x(const int a) {x = a;}
    /**
     * @brief Установить значение поля y, равное координате по оси y
     * @param b координата по оси y
     */
    void set_y(const int b) {y = b;}
    /**
     * @brief Установить значение поля status, равное состоянию клетки
     * @param st статус клетки
     */
    void set_status(const bool st) {status = st;}
    /**
     * @brief Подсчёт колличества соседних живых клеток
     * @param c двумерный массив клеток
     */
    int search_living(const std::vector<std::vector<Cell>> c);
    /**
     * @brief Проверка состояние клетки
     * @param проверяемая клетка
     */
    int check(Cell);
private:
```

```
62      int x;
63      int y;
64      bool status;
65  };
66
67  #endif // CELL_H
```

```
1   #include "cell.h"
2
3   int Cell::check(Cell c){
4       if (c.get_status() != 0){
5           return 1;
6       }else{
7           return 0;
8       }
9   }
10
11  int Cell::search_living(const std::vector<std::vector<Cell> > c){
12      int number = 0;
13      if (x - 1 != -1){
14          Cell cell = c[x-1][y];
15          number += this->check(cell);
16      }
17      if ((x - 1 != -1) && (y - 1 != -1)){
18          Cell cell = c[x-1][y-1];
19          number += this->check(cell);
20      }
21      if (y - 1 != -1){
22          Cell cell = c[x][y-1];
23          number += this->check(cell);
24      }
25      if ((x + 1 != (int) c.size()) && (y - 1 != -1)) {
26          Cell cell = c[x+1][y-1];
27          number += this->check(cell);
28      }
29      if (x + 1 != (int) c.size()){
30          Cell cell = c[x+1][y];
31          number += this->check(cell);
32      }
33      if ((x + 1 != (int) c.size()) && (y + 1 != (int) c[x].size()))
        ↪ {
34          Cell cell = c[x+1][y+1];
35          number += this->check(cell);
36      }
37      if (y + 1 != (int) c[x].size()){
38          Cell cell = c[x][y+1];
39          number += this->check(cell);
40      }
41      if ((x - 1 != -1) && (y + 1 != (int) c[x].size())){
42          Cell cell = c[x-1][y+1];
43          number += this->check(cell);
44      }
45      return number;
46  }
```

```
#ifndef FIELD_H
#define FIELD_H
#include "cell.h"
#include <vector>

/**
   @brief Класс Поле

   Этот класс моделирует объект поле на котором располагаются клетки.
   */

class Field
{
public:
    /**
     * @brief Конструктор
     * @param a размер поля по оси x
     * @param b размер поля по оси y
     */
    Field(const int a = 0, const int b = 0): x(a), y(b) {
        cells.resize(x);
        for (int i = 0; i < x; ++i){
            cells[i].resize(y);
        }
    }
    /**
     * @brief Получить размер поля по оси x
     * @return размер поля по оси x
     */
    int get_x() {return x;}
    /**
     * @brief Получить размер поля по оси y
     * @return размер поля по оси y
     */
    int get_y() {return y;}
    /**
     * @brief Получить клетку из поля
     * @param x_cell координата клетки по x
     * @param y_cell координата клетки по y
     * @return клетки из поля
     */
    Cell get_cell(const int x_cell, const int y_cell) {return cells
    ↪ [x_cell][y_cell];}
    /**
     * @brief Получить все клетки поля
     * @return двумерный массив клеток
     */
    std::vector<std::vector<Cell> > get_cells() {return cells;}
    /**
     * @brief Установить значение поля x, равное размеру поля по оси x
     * @param a размер поля по оси x
     */
    void set_x(const int a);
    /**
     * @brief Установить значение поля y, равное размеру поля по оси y
     * @param b размер поля по оси y
     */
    void set_y(const int b);
    /**
     * @brief Задать данные новой клетки
     * @param x_cell координата клетки по x
     * @param y_cell координата клетки по y
```

```
62          * @param status состояние клетки
63          */
64         void set_cell(const int x_cell, const int y_cell, const int
        ↪ status);
65         /**
66          * @brief Задать массив клеток
67          * @param c двумерный массив клеток
68          */
69         void set_cells(const std::vector<std::vector<Cell> > c) {cells
        ↪ = c;}
70     private:
71         int x;
72         int y;
73         std::vector<std::vector<Cell> > cells;
74     };
75
76     #endif // FIELD_H
```

```
1  #include "field.h"
2  #include "cell.h"
3
4  void Field::set_x(const int a){
5      x = a;
6      cells.resize(x);
7  }
8
9  void Field::set_y(const int b)
10 {
11     y = b;
12     for (int i = 0; i < x; i++){
13         cells[i].resize(y);
14     }
15 }
16
17 void Field::set_cell(const int x_cell, const int y_cell, const int
       ↪ status)
18 {
19     Cell c(x_cell,y_cell,status);
20     cells[x_cell][y_cell] = c;
21 }
```

6.4 Тесты

```
#include <QString>
#include <QtTest>
#include "api.h"
#include "field.h"

class TestTest : public QObject
{
    Q_OBJECT

public:
    TestTest();

private Q_SLOTS:
    void test_initialize_field();
    void test_change_field();
    void test_initialize_settings();
};

TestTest::TestTest()
{
}

void TestTest::test_initialize_settings()
{
    Field f;
    Api model;
    model.initialize_settings(f,5,5);
    QCOMPARE(f.get_x(),5);
    QCOMPARE(f.get_y(),5);
}

void TestTest::test_initialize_field()
{
    Field f(5,5);
    Api model;
    bool flag = true;
    model.initialize_field(f);
    for (int i = 0; i < 5; ++i){
        for (int j = 0; j < 5; ++j){
            Cell c = f.get_cell(i,j);
            if ((c.get_status() != 0) && (c.get_status() != 1)){
                flag = false;
            }
        }
    }
    QCOMPARE(flag,true);
}

void TestTest::test_change_field(){
    Field f(3,3);
    Api model;
    f.set_cell(0,0,1);
    f.set_cell(0,1,1);
    f.set_cell(0,2,1);
    f.set_cell(1,0,1);
    f.set_cell(1,1,0);
    f.set_cell(1,2,1);
    f.set_cell(2,0,0);
    f.set_cell(2,1,1);
    f.set_cell(2,2,1);
```

```
61      model.change_field(f);
62      Cell c = f.get_cell(0,0);
63      QCOMPARE(c.get_status(),true);
64      c = f.get_cell(0,1);
65      QCOMPARE(c.get_status(),false);
66      c = f.get_cell(0,2);
67      QCOMPARE(c.get_status(),true);
68      c = f.get_cell(1,0);
69      QCOMPARE(c.get_status(),true);
70      c = f.get_cell(1,1);
71      QCOMPARE(c.get_status(),false);
72      c = f.get_cell(1,2);
73      QCOMPARE(c.get_status(),false);
74      c = f.get_cell(2,0);
75      QCOMPARE(c.get_status(),false);
76      c = f.get_cell(2,1);
77      QCOMPARE(c.get_status(),true);
78      c = f.get_cell(2,2);
79      QCOMPARE(c.get_status(),true);
80  }
81
82  QTEST_APPLESS_MAIN(TestTest)
83
84  #include "tst_testtest.moc"
```