

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	2
1.1	Задание	2
1.2	Правила работы программы	2
1.3	Концепция	2
1.4	Минимально работоспособный продукт	2
1.5	Диаграмма прецедентов использования	3
2	Проектирование приложения, реализующего модель Жизнь	3
2.1	Библиотека	4
3	Реализация модели Жизнь	4
3.1	Версии программ	4
3.2	Консольное приложение	4
3.3	Библиотека	5
3.4	Графическое приложение	6
4	Процесс обеспечения качества и тестирование	8
4.1	Тестирование	8
5	Вывод	8
6	Приложение 1. Листинги кода	8
6.1	Консольное приложение	8
6.2	Графическое приложение	12
6.3	Библиотека	26
6.4	Тесты	33

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 меняет 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 состоянием 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

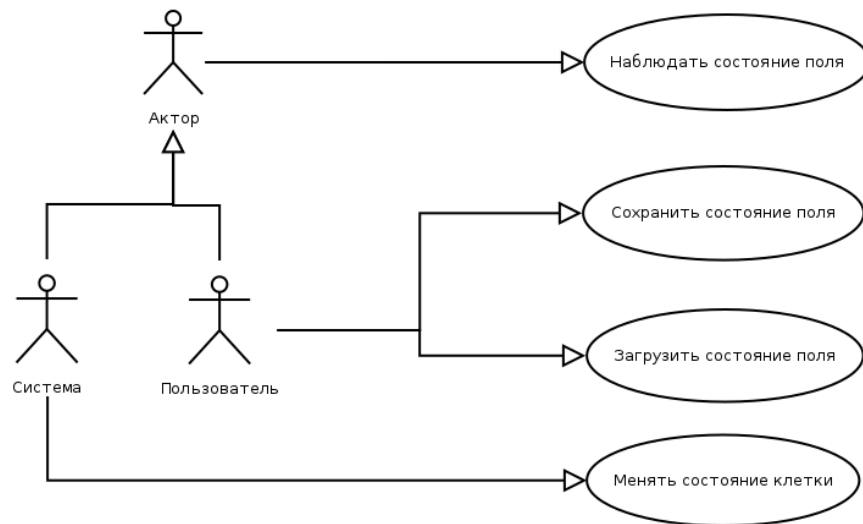


Рис. 1: Диаграмма прецедентов использования

2 Проектирование приложения, реализующего модель Жизнь

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

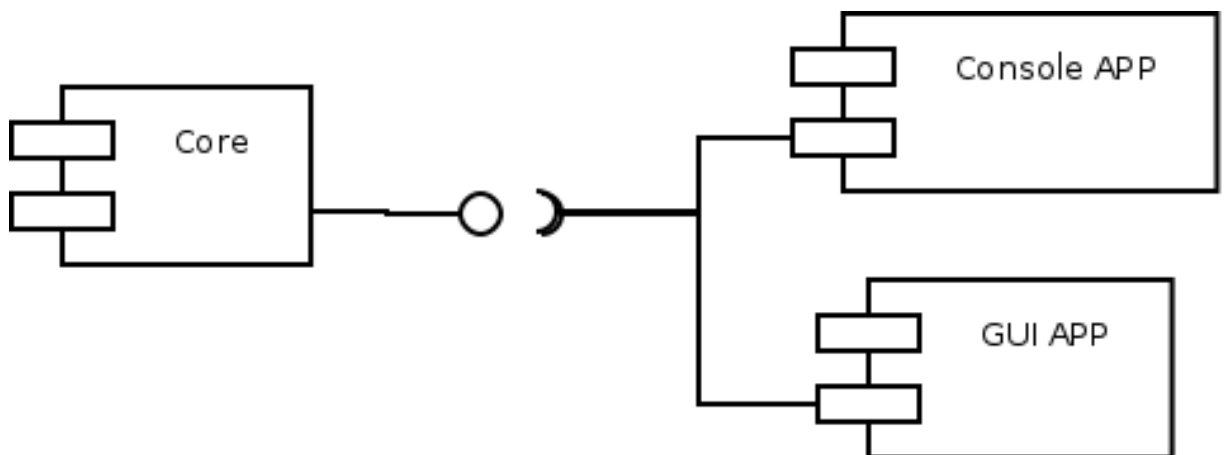


Рис. 2: Диаграмма прецедентов использования

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.

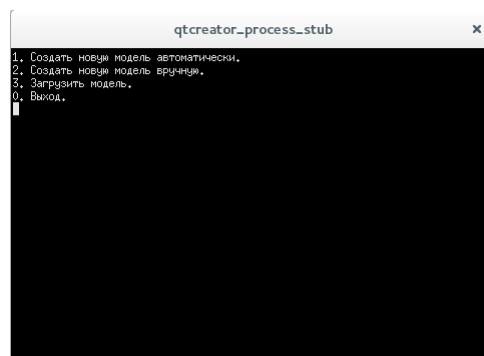


Рис. 3: Главное меню консольного приложения

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.

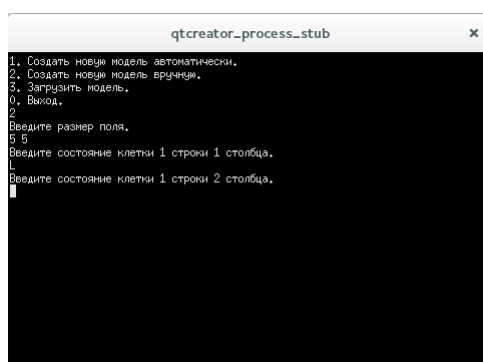


Рис. 5: Поле модели в консольном приложении

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 подтверждения 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.

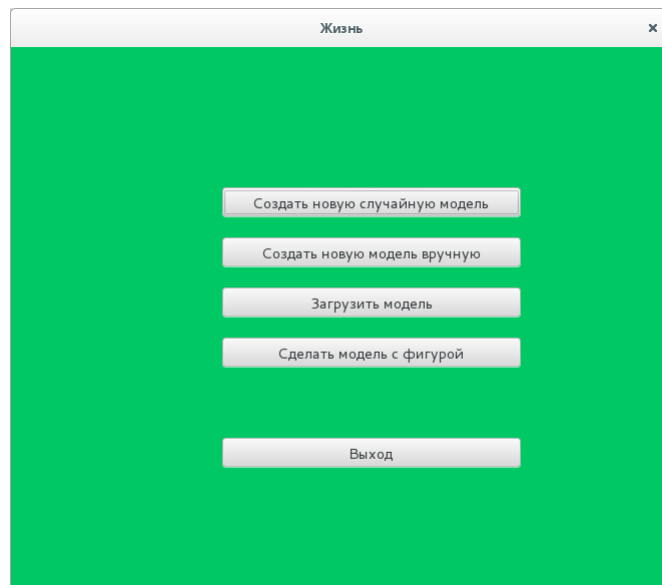


Рис. 6: Главное меню графического приложения

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.

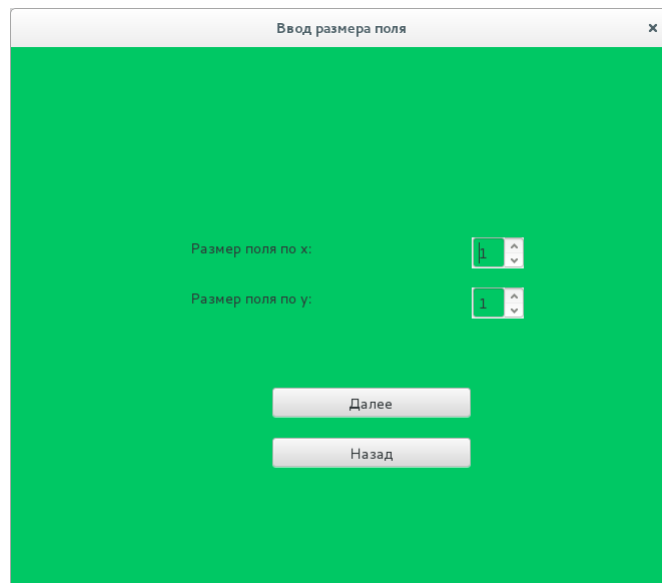


Рис. 7: Настройки модели в графическом приложении

На рис 7 – окно выбора размера приложения.

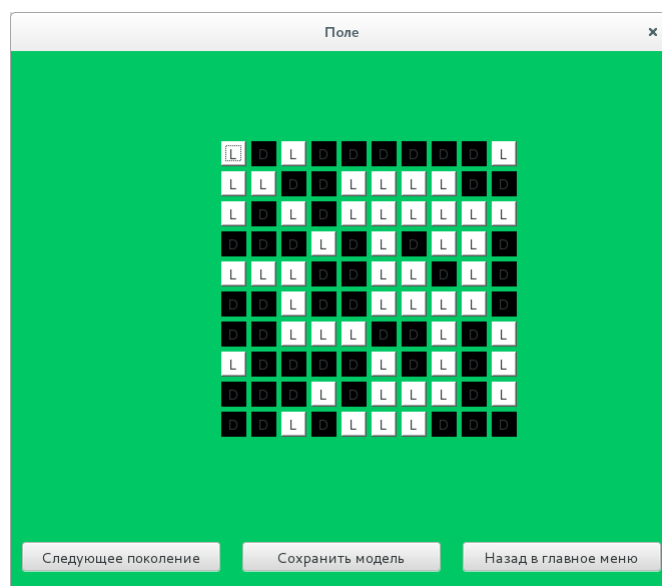


Рис. 8: Представление поля в графическом приложении

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 }
```

```

1 #ifndef CONSOLE_UI_H
2 #define CONSOLE_UI_H
3 #include "api.h"
4 #include "field.h"
5
6 /**
7  * @brief Класс консольного меню
8
9  * Этот класс, посредством консоли, взаимодействует с пользователем.
10 */
11
12 class console_ui
13 {
14 public:
15     /**
16      * @brief Вывести главное меню
17      */
18     void print_menu();
19     /**
20      * @brief Вывести промежуточное меню
21      * @param f поле
22      * @param a объект класса Api
23      */
24     void secondary_menu(Field &f, Api a);
25     /**
26      * @brief Вывести меню задания клеток поля
27      * @param f поле
28      * @param a объект класса Api
29      * @param x размер поля по оси x
30      * @param y размер поля по оси y
31      */
32     void input_menu(Field &f, Api a, int x, int y);
33 };
34
35 #endif // CONSOLE_UI_H

```

```

1 #include "console_ui.h"
2 #include <iostream>
3 #include <fstream>
4 #include "string"
5
6 void console_ui::print_menu()
7 {
8     int choice = -1;
9     Api a;
10    Field f;
11    while(choice != 0) {
12        std::cout << "1. Создать новую модель автоматически." << std::endl;
13        std::cout << "2. Создать новую модель вручную." << std::endl;
14        std::cout << "3. Загрузить модель." << std::endl;
15        std::cout << "0. Выход." << std::endl;
16        std::cin >> choice;
17        int x,y;
18        switch (choice)
19        {
20            case 1:
21                std::cout << "Введите размер поля." << std::endl;
22                std::cin >> x >> y;
23                a.initialize_settings(f,x,y);
24                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

```
1 #include "mainwindow.h"
2 #include <QApplication>
3 #include "field.h"
4 #include "api.h"
5
6 int main(int argc, char *argv[])
7 {
8     QApplication a(argc, argv);
9     MainWindow w(0);
10    w.show();
11
12    return a.exec();
13 }
```

```

1 #ifndef EXIT_WINDOW_H
2 #define EXIT_WINDOW_H
3
4 #include <QPushButton>
5 #include <QLabel>
6 #include "mainwindow.h"
7 #include <QtWidgets>
8
9
10 class exit_window : public QDialog
11 {
12     Q_OBJECT
13
14     const QSize WINDOW_SIZE { 300, 90 };
15     const QSize BUTTON_SIZE { 120, 30 };
16     QPushButton* yes_button;
17     QPushButton* no_button;
18     QWidget* parent;
19     QLabel* exit_label;
20 public:
21     explicit exit_window(QWidget* parent);
22 private slots:
23     void close_app();
24     void close_exit_window();
25 };
26
27 #endif // EXIT_WINDOW_H

```

```

1 #include "exit_window.h"
2
3 exit_window::exit_window(QWidget* parent) : QDialog(parent)
4 {
5     this->parent = parent;
6     this->setFixedSize(WINDOW_SIZE);
7     this->setWindowTitle("Подтверждение_выхода");
8     QPalette pal;
9     pal.setColor(QPalette::Background, QColor(0, 200, 100, 255));
10    this->setPalette(pal);
11    exit_label = new QLabel(this);
12    exit_label->move(WINDOW_SIZE.width() - 283, WINDOW_SIZE.height
    ↪ () - 80);
13    exit_label->setText("Вы_действительно_хотите_выйти?");
14    exit_label->show();
15    yes_button = new QPushButton("Да", this);
16    yes_button->resize(BUTTON_SIZE);
17    yes_button->move(WINDOW_SIZE.width() - 275, WINDOW_SIZE.height
    ↪ () - 50);
18    connect(yes_button, SIGNAL(clicked()), SLOT(close_app()));
19    no_button = new QPushButton("Нет", this);
20    no_button->resize(BUTTON_SIZE);
21    no_button->move(WINDOW_SIZE.width() - 125, WINDOW_SIZE.height()
    ↪ - 50);
22    connect(no_button, SIGNAL(clicked()), SLOT(close_exit_window())
    ↪ );
23 }
24
25 void exit_window::close_app()
26 {
27     this->close();
28     this->parent->close();
29 }

```

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

```

1 #ifndef FIELD_WINDOW_H
2 #define FIELD_WINDOW_H
3
4 #include <QWidget>
5 #include "mainwindow.h"
6 #include "api.h"
7 #include "field.h"
8
9 class field_window : public QWidget
10 {
11     Q_OBJECT
12
13     QWidget* parent;
14     Api a;
15     Field f;
16     const QSize WINDOW_SIZE { 660, 540 };
17     const QSize BUTTON_SIZE { 200, 30 };
18     const QSize CELL_SIZE { 25, 25 };
19     QPushButton* next_generation_button;
20     QPushButton* back_button;
21     QPushButton* save_button;
22 public:
23     field_window(QWidget* parent, Api a, Field f);
24     void print_field();
25 private slots:
26     void next_generation();
27     void back_in_main_menu();
28     void save_field();
29 };
30
31 #endif // FIELD_WINDOW_H

```

```

1 #include "field_window.h"
2
3 field_window::field_window(QWidget *parent, Api a, Field f) :
4     ↪ QWidget(parent)
5 {
6     this->a = a;
7     this->f = f;
8     this->setFixedSize(WINDOW_SIZE);
9     this->setWindowTitle("Поле");
10    QPalette pal;
11    pal.setColor(QPalette::Background, QColor(0, 200, 100, 255));
12    this->setPalette(pal);
13    this->print_field();
14    next_generation_button = new QPushButton("Следующее поколение",
15    ↪ this);
16    next_generation_button->resize(BUTTON_SIZE);
17    next_generation_button->move(WINDOW_SIZE.width() - 650,
18    ↪ WINDOW_SIZE.height() - 50);
19    connect(next_generation_button, SIGNAL(clicked()), SLOT(
20    ↪ next_generation()));
21    save_button = new QPushButton("Сохранить модель", this);
22    save_button->resize(BUTTON_SIZE);
23    save_button->move(WINDOW_SIZE.width() - 430, WINDOW_SIZE.height
24    ↪ () - 50);
25    connect(save_button, SIGNAL(clicked()), SLOT(save_field()));
26    back_button = new QPushButton("Назад в главное меню", this);
27    back_button->resize(BUTTON_SIZE);
28    back_button->move(WINDOW_SIZE.width() - 210, WINDOW_SIZE.height
29    ↪ () - 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 }

```

```

1 #ifndef FIGURE_WINDOW_H
2 #define FIGURE_WINDOW_H
3
4 #include <QPushButton>
5 #include <QLabel>
6 #include "mainwindow.h"
7 #include "api.h"
8 #include "field.h"
9 #include <QtWidgets>
10
11 class figure_window : public QDialog
12 {
13     Q_OBJECT
14
15     Api a;
16     Field f;
17     const QSize WINDOW_SIZE { 430, 430 };
18     const QSize BUTTON_SIZE { 160, 30 };
19     QPushButton* glider_button;
20     QPushButton* spaceship_button;
21     QWidget* parent;
22     QLabel* figure_label;
23 public:
24     explicit figure_window(QWidget* parent);
25 private slots:
26     void create_spaceship();
27     void create_glider();
28 };
29
30 #endif // FIGURE_WINDOW_H

```

```

1 #include "figure_window.h"
2
3 figure_window::figure_window(QWidget* parent) : QDialog(parent)
4 {
5     this->parent = parent;
6     this->setFixedSize(WINDOW_SIZE);
7     this->setWindowTitle("Выбор_фигуры");
8     QPalette pal;
9     pal.setColor(QPalette::Background, QColor(0, 200, 100, 255));
10    this->setPalette(pal);
11    figure_label = new QLabel(this);
12    figure_label->move(WINDOW_SIZE.width() - 283, WINDOW_SIZE.
13    ↪ height() - 380);
14    figure_label->setText("Выберете_фигуру");
15    figure_label->show();
16    glider_button = new QPushButton("Глайдер", this);
17    glider_button->resize(BUTTON_SIZE);
18    glider_button->move(WINDOW_SIZE.width() - 300, WINDOW_SIZE.
19    ↪ height() - 350);
20    connect(glider_button, SIGNAL(clicked()), SLOT(create_glider()));
21    ↪ );
22    spaceship_button = new QPushButton("Космический_корабль", this);
23    spaceship_button->resize(BUTTON_SIZE);
24    spaceship_button->move(WINDOW_SIZE.width() - 300, WINDOW_SIZE.
25    ↪ height() - 300);
26    connect(spaceship_button, SIGNAL(clicked()), SLOT(
27    ↪ create_spaceship()));
28 }
29
30 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 }

```

```

1 #ifndef INITIALIZE_WINDOW_H
2 #define INITIALIZE_WINDOW_H
3
4 #include <QWidget>
5 #include <QPushButton>
6 #include <QLabel>
7 #include <QtWidgets>
8 #include "api.h"
9 #include "field.h"
10
11 class initialize_window : public QDialog
12 {
13     Q_OBJECT
14
15     QWidget* parent;
16     Api* a;
17     Field* f;
18     int x;
19     int y;
20     const QSize WINDOW_SIZE { 430, 220 };
21     const QSize BUTTON_SIZE { 200, 30 };
22     QLabel* status_cell;
23     QPushButton* live_button;
24     QPushButton* dead_button;
25 public:
26     explicit initialize_window(QWidget* parent, Api* a, Field* f,
27     ↪ int i, int j);
28 private slots:
29     void clicked_button_live();
30     void clicked_button_dead();
31 };
32 #endif // INITIALIZE_WINDOW_H

```

```

1 #include "initialize_window.h"
2
3 initialize_window::initialize_window(QWidget *parent, Api *a, Field
4     ↪ *f, int i, int j) : QDialog(parent)
5 {
6     this->parent = parent;
7     this->f = f;
8     this->a = a;
9     this->x = i;
10    this->y = j;
11    this->setFixedSize(WINDOW_SIZE);
12    this->setWindowTitle("Выбор_состояния_клетки");
13    QPalette pal;
14    pal.setColor(QPalette::Background, QColor(0, 200, 100, 255));
15    this->setPalette(pal);
16    status_cell = new QLabel(this);
17    status_cell->move(WINDOW_SIZE.width() - 385, WINDOW_SIZE.height
18    ↪ () - 160);
19    status_cell->setText(QString("Выберете_состояние_клетки_%1_строки,
20    ↪ %2_столбца").arg(y+1).arg(x+1));
21    status_cell->show();
22    live_button = new QPushButton("Живая", this);
23    live_button->resize(BUTTON_SIZE);
24    live_button->move(WINDOW_SIZE.width() - 325, WINDOW_SIZE.height
25    ↪ () - 50);
26    connect(live_button, SIGNAL(clicked()), SLOT(
27    ↪ 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 }

```

```

1 #ifndef MAINWINDOW_H
2 #define MAINWINDOW_H
3
4 #include <QWidget>
5 #include <QPushButton>
6 #include <QInputDialog>
7 #include "size_window.h"
8 #include "exit_window.h"
9 #include "figure_window.h"
10 #include "field.h"
11 #include "api.h"
12
13 class MainWindow : public QWidget
14 {
15     Q_OBJECT
16
17     const QSize WINDOW_SIZE { 660, 540 };
18     const QSize BUTTON_SIZE { 300, 30 };
19     QPushButton* new_random_model_button;
20     QPushButton* load_button;
21     QPushButton* figure_button;
22     QPushButton* new_model_button;
23     QPushButton* exit_button;
24 public:
25     MainWindow(QWidget *parent);
26 private slots:
27     void figure_menu();
28     void close_menu();
29     void create_model_random();
30     void create_model();
31     void load_model();
32 };
33
34 #endif // MAINWINDOW_H

```

```

1 #include "mainwindow.h"
2
3 MainWindow::MainWindow(QWidget *parent)
4     : QWidget(parent)
5 {
6     this->setFixedSize(WINDOW_SIZE);
7     this->setWindowTitle("Жизнь");
8     QPalette pal;
9     pal.setColor(QPalette::Background, QColor(0, 200, 100, 255));
10    this->setPalette(pal);
11    new_random_model_button = new QPushButton("Создать_новую_
    ↪ случайную_модель", this);
12    new_random_model_button->resize(BUTTON_SIZE);
13    new_random_model_button->move(WINDOW_SIZE.width() - 450,
    ↪ WINDOW_SIZE.height() - 400);
14    connect(new_random_model_button, SIGNAL(clicked()), SLOT(
    ↪ create_model_random()));
15    load_button = new QPushButton("Загрузить_модель", this);
16    load_button->resize(BUTTON_SIZE);
17    load_button->move(WINDOW_SIZE.width() - 450, WINDOW_SIZE.height
    ↪ () - 300);
18    connect(load_button, SIGNAL(clicked()), SLOT(load_model()));
19    new_model_button = new QPushButton("Создать_новую_модель_вручную"
    ↪ , this);
20    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 }

```



```

1 #ifndef SIZE_WINDOW_H
2 #define SIZE_WINDOW_H
3
4 #include <QLabel>
5 #include <QSpinBox>
6 #include "mainwindow.h"
7 #include "field_window.h"
8 #include "field.h"
9 #include "api.h"
10 #include "initialize_window.h"
11
12 class size_window : public QWidget
13 {
14     Q_OBJECT
15
16     QWidget* parent;
17     Api a;
18     Field f;
19     int mode;
20     const QSize WINDOW_SIZE { 660, 540 };
21     const QSize BUTTON_SIZE { 200, 30 };
22     QPushButton* next_button;
23     QPushButton* back_button;
24     QLabel* size_field_x;
25     QLabel* size_field_y;
26     QSpinBox* size_Field_x;
27     QSpinBox* size_Field_y;
28     QLabel* create_label(QString text, int coordinate_x, int
    ↪ coordinate_y);
29     QSpinBox* create_spin_box(int min, int max, int coordinate_x,
    ↪ int coordinate_y);
30 public:
31     size_window(QWidget* parent, int mode);
32 private slots:
33     void print_field();
34     void close_size_window();
35 };
36
37 #endif // SIZE_WINDOW_H

```

```

1 #include "size_window.h"
2
3 size_window::size_window(QWidget* parent, int mode) : QWidget(
    ↪ parent)
4 {
5     this->mode = mode;
6     this->setFixedSize(WINDOW_SIZE);
7     this->setWindowTitle("Ввод_размера_поля");
8     QPalette pal;
9     pal.setColor(QPalette::Background, QColor(0, 200, 100, 255));
10    this->setPalette(pal);
11    size_field_x = create_label("Размер_поля_по_x:", WINDOW_SIZE.
    ↪ width() - 480, WINDOW_SIZE.height() - 350);
12    size_Field_x = create_spin_box(1, 10, WINDOW_SIZE.width() -
    ↪ 200, WINDOW_SIZE.height() - 350);
13    size_field_y = create_label("Размер_поля_по_y:", WINDOW_SIZE.
    ↪ width() - 480, WINDOW_SIZE.height() - 300);
14    size_Field_y = create_spin_box(1, 10, WINDOW_SIZE.width() -
    ↪ 200, WINDOW_SIZE.height() - 300);
15    next_button = new QPushButton("Далее", this);
16    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

```
1 #ifndef API_H
2 #define API_H
3 #include "field.h"
4 #include "string"
5
6 /**
7  * @brief Класс Api
8
9  * Этот класс предоставляет методы ядра.
10  */
11
12 class Api
13 {
14 public:
15     /**
16      * @brief Задать размеры поля
17      * @param f поле
18      * @param f_x размер поля по оси x
19      * @param f_y размер поля по оси y
20      */
21     void initialize_settings(Field &f, int f_x, int f_y);
22     /**
23      * @brief Инициализировать поле случайными клетками
24      * @param f поле
25      */
26     void initialize_field(Field &f);
27     /**
28      * @brief Сохранить поле в файл
29      * @param f поле
30      */
31     void save_field(Field &f);
32     /**
33      * @brief Загрузить поле из файла
34      * @param f поле
35      */
36     void load_field(Field &f);
37     /**
38      * @brief Вывести поле в консоль
39      * @param f поле
40      */
41     void print_field(Field &f);
42     /**
43      * @brief Сменить поколение и изменить поле
44      * @param f поле
45      */
46     void change_field(Field &f);
47     /**
48      * @brief Задать клетку поля
49      * @param f поле
50      * @param x координата клетки по оси x
51      * @param y координата клетки по оси y
52      * @param status статус клетки
53      */
54     void set_cell(Field &f, int x, int y, std::string status);
55 };
56
57 #endif // API_H
```

```
1 #include "api.h"
2 #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
94 ↪ == 2))){
95                     new_f.set_cell(i,j,1);
96                 }else{
97                     new_f.set_cell(i,j,0);
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 }

```

```

1 #ifndef CELL_H
2 #define CELL_H
3 #include <vector>
4
5 /**
6  * @brief Класс Клетка
7
8  * Этот класс моделирует объект клетка из которых будет состоять поле.
9  */
10
11 class Cell
12 {
13 public:
14     /**
15      * @brief Конструктор
16      * @param a координата по оси x
17      * @param b координата по оси y
18      * @param st состояние клетки
19      */
20     Cell(const int a = 0, const int b = 0, const bool st = false):
21     ↪ x(a), y(b), status(st) {}
22     /**
23      * @brief Получить координату по оси x
24      * @return координата по оси x
25      */
26     int get_x() {return x;}
27     /**
28      * @brief Получить координату по оси y
29      * @return координата по оси y
30      */
31     int get_y() {return y;}
32     /**
33      * @brief Получить состояние клетки
34      * @return состояние клетки
35      */
36     bool get_status() {return status;}
37     /**
38      * @brief Установить значение поля x, равное координате по оси x
39      * @param a координата по оси x
40      */
41     void set_x(const int a) {x = a;}
42     /**
43      * @brief Установить значение поля y, равное координате по оси y
44      * @param b координата по оси y
45      */
46     void set_y(const int b) {y = b;}
47     /**
48      * @brief Установить значение поля status, равное состоянию клетки
49      * @param st статус клетки
50      */
51     void set_status(const bool st) {status = st;}
52     /**
53      * @brief Подсчёт количества соседних живых клеток
54      * @param c двумерный массив клеток
55      */
56     int search_living(const std::vector<std::vector<Cell>> c);
57     /**
58      * @brief Проверка состояние клетки
59      * @param проверяемая клетка
60      */
61     int check(Cell);
62 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     ↪ {
35         Cell cell = c[x+1][y+1];
36         number += this->check(cell);
37     }
38     if (y + 1 != (int) c[x].size()){
39         Cell cell = c[x][y+1];
40         number += this->check(cell);
41     }
42     if ((x - 1 != -1) && (y + 1 != (int) c[x].size())){
43         Cell cell = c[x-1][y+1];
44         number += this->check(cell);
45     }
46     return number;

```

```

1 #ifndef FIELD_H
2 #define FIELD_H
3 #include "cell.h"
4 #include <vector>
5
6 /**
7  * @brief Класс Поле
8  *
9  * Этот класс моделирует объект поле на котором располагаются клетки.
10 */
11
12 class Field
13 {
14 public:
15     /**
16      * @brief Конструктор
17      * @param a размер поля по оси x
18      * @param b размер поля по оси y
19      */
20     Field(const int a = 0, const int b = 0): x(a), y(b) {
21         cells.resize(x);
22         for (int i = 0; i < x; ++i){
23             cells[i].resize(y);
24         }
25     }
26     /**
27      * @brief Получить размер поля по оси x
28      * @return размер поля по оси x
29      */
30     int get_x() {return x;}
31     /**
32      * @brief Получить размер поля по оси y
33      * @return размер поля по оси y
34      */
35     int get_y() {return y;}
36     /**
37      * @brief Получить клетку из поля
38      * @param x_cell координата клетки по x
39      * @param y_cell координата клетки по y
40      * @return клетки из поля
41      */
42     Cell get_cell(const int x_cell, const int y_cell) {return cells
43     → [x_cell][y_cell];}
44     /**
45      * @brief Получить все клетки поля
46      * @return двумерный массив клеток
47      */
48     std::vector<std::vector<Cell> > get_cells() {return cells;}
49     /**
50      * @brief Установить значение поля x, равное размеру поля по оси x
51      * @param a размер поля по оси x
52      */
53     void set_x(const int a);
54     /**
55      * @brief Установить значение поля y, равное размеру поля по оси y
56      * @param b размер поля по оси y
57      */
58     void set_y(const int b);
59     /**
60      * @brief Задать данные новой клетки
61      * @param x_cell координата клетки по x
62      * @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

```
1 #include <QString>
2 #include <QtTest>
3 #include "api.h"
4 #include "field.h"
5
6 class TestTest : public QObject
7 {
8     Q_OBJECT
9
10 public:
11     TestTest();
12
13 private Q_SLOTS:
14     void test_initialize_field();
15     void test_change_field();
16     void test_initialize_settings();
17 };
18
19 TestTest::TestTest()
20 {
21 }
22
23 void TestTest::test_initialize_settings()
24 {
25     Field f;
26     Api model;
27     model.initialize_settings(f,5,5);
28     QCOMPARE(f.get_x(),5);
29     QCOMPARE(f.get_y(),5);
30 }
31
32 void TestTest::test_initialize_field()
33 {
34     Field f(5,5);
35     Api model;
36     bool flag = true;
37     model.initialize_field(f);
38     for (int i = 0; i < 5; ++i){
39         for (int j = 0; j < 5; ++j){
40             Cell c = f.get_cell(i,j);
41             if ((c.get_status() != 0) && (c.get_status() != 1)){
42                 flag = false;
43             }
44         }
45     }
46     QCOMPARE(flag, true);
47 }
48
49 void TestTest::test_change_field(){
50     Field f(3,3);
51     Api model;
52     f.set_cell(0,0,1);
53     f.set_cell(0,1,1);
54     f.set_cell(0,2,1);
55     f.set_cell(1,0,1);
56     f.set_cell(1,1,0);
57     f.set_cell(1,2,1);
58     f.set_cell(2,0,0);
59     f.set_cell(2,1,1);
60     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"

```

My Project

Создано системой Doxygen 1.8.8

Чт 2 Июн 2016 21:37:17

Оглавление

1	Иерархический список классов	1
1.1	Иерархия классов	1
2	Алфавитный указатель классов	3
2.1	Классы	3
3	Классы	5
3.1	Класс <code>Api</code>	5
3.1.1	Подробное описание	5
3.1.2	Методы	5
3.1.2.1	<code>change_field</code>	5
3.1.2.2	<code>initialize_field</code>	6
3.1.2.3	<code>initialize_settings</code>	6
3.1.2.4	<code>load_field</code>	6
3.1.2.5	<code>print_field</code>	6
3.1.2.6	<code>save_field</code>	6
3.1.2.7	<code>set_cell</code>	6
3.2	Класс <code>Cell</code>	7
3.2.1	Подробное описание	7
3.2.2	Конструктор(ы)	7
3.2.2.1	<code>Cell</code>	7
3.2.3	Методы	8
3.2.3.1	<code>check</code>	8
3.2.3.2	<code>get_status</code>	8
3.2.3.3	<code>get_x</code>	8
3.2.3.4	<code>get_y</code>	8
3.2.3.5	<code>search_living</code>	8
3.2.3.6	<code>set_status</code>	8
3.2.3.7	<code>set_x</code>	9
3.2.3.8	<code>set_y</code>	9
3.3	Класс <code>console_ui</code>	9
3.3.1	Подробное описание	9

3.3.2	Методы	9
3.3.2.1	input_menu	9
3.3.2.2	secondary_menu	10
3.4	Класс exit_window	10
3.5	Класс Field	10
3.5.1	Подробное описание	11
3.5.2	Конструктор(ы)	11
3.5.2.1	Field	11
3.5.3	Методы	11
3.5.3.1	get_cell	11
3.5.3.2	get_cells	11
3.5.3.3	get_x	11
3.5.3.4	get_y	12
3.5.3.5	set_cell	12
3.5.3.6	set_cells	12
3.5.3.7	set_x	12
3.5.3.8	set_y	12
3.6	Класс field_window	12
3.7	Класс figure_window	13
3.8	Класс initialize_window	13
3.9	Класс MainWindow	13
3.10	Структура qt_meta_stringdata_exit_window_t	14
3.11	Структура qt_meta_stringdata_field_window_t	14
3.12	Структура qt_meta_stringdata_figure_window_t	14
3.13	Структура qt_meta_stringdata_initialize_window_t	14
3.14	Структура qt_meta_stringdata_MainWindow_t	15
3.15	Структура qt_meta_stringdata_size_window_t	15
3.16	Класс size_window	15
3.17	Класс TestTest	15
	Алфавитный указатель	16

Глава 1

Иерархический список классов

1.1 Иерархия классов

Иерархия классов.

Api	5
Cell	7
console_ui	9
Field	10
QDialog	
exit_window	10
figure_window	13
initialize_window	13
QObject	
TestTest	15
qt_meta_stringdata_exit_window_t	14
qt_meta_stringdata_field_window_t	14
qt_meta_stringdata_figure_window_t	14
qt_meta_stringdata_initialize_window_t	14
qt_meta_stringdata_MainWindow_t	15
qt_meta_stringdata_size_window_t	15
QWidget	
field_window	12
MainWindow	13
size_window	15

Глава 2

Алфавитный указатель классов

2.1 Классы

Классы с их кратким описанием.

Api		
	Класс Api	5
Cell		
	Класс Клетка	7
console_ui		
	Класс консольного меню	9
exit_window		10
Field		
	Класс Поле	10
field_window		12
figure_window		13
initialize_window		13
MainWindow		13
qt_meta_stringdata_exit_window_t		14
qt_meta_stringdata_field_window_t		14
qt_meta_stringdata_figure_window_t		14
qt_meta_stringdata_initialize_window_t		14
qt_meta_stringdata_MainWindow_t		15
qt_meta_stringdata_size_window_t		15
size_window		15
TestTest		15

Глава 3

Классы

3.1 Класс Api

Класс [Api](#).

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

Открытые члены

- void [initialize_settings](#) ([Field](#) &f, int f_x, int f_y)
Задать размеры поля
- void [initialize_field](#) ([Field](#) &f)
Инициализировать поле случайными клетками
- void [save_field](#) ([Field](#) &f)
Сохранить поле в файл
- void [load_field](#) ([Field](#) &f)
Загрузить поле из файла
- void [print_field](#) ([Field](#) &f)
Вывести поле в консоль
- void [change_field](#) ([Field](#) &f)
Сменить поколение и изменить поле
- void [set_cell](#) ([Field](#) &f, int x, int y, std::string status)
Задать клетку поля

3.1.1 Подробное описание

Класс [Api](#).

Этот класс предоставляет методы ядра.

3.1.2 Методы

3.1.2.1 void Api::change_field (Field & f)

Сменить поколение и изменить поле

Аргументы

f	поле
---	------

3.1.2.2 void Api::initialize_field (Field & f)

Инициализировать поле случайными клетками

Аргументы

f	поле
---	------

3.1.2.3 void Api::initialize_settings (Field & f, int f_x, int f_y)

Задать размеры поля

Аргументы

f	поле
f_x	размер поля по оси x
f_y	размер поля по оси y

3.1.2.4 void Api::load_field (Field & f)

Загрузить поле из файла

Аргументы

f	поле
---	------

3.1.2.5 void Api::print_field (Field & f)

Вывести поле в консоль

Аргументы

f	поле
---	------

3.1.2.6 void Api::save_field (Field & f)

Сохранить поле в файл

Аргументы

f	поле
---	------

3.1.2.7 void Api::set_cell (Field & f, int x, int y, std::string status)

Задать клетку поля

Аргументы

f	поле
x	координата клетки по оси x
y	координата клетки по оси y
status	статус клетки

Объявления и описания членов классов находятся в файлах:

- /home/user/LivingCells/sources/livingcells/core/api.h
- /home/user/LivingCells/sources/livingcells/core/api.cpp

3.2 Класс Cell

Класс Клетка

```
#include <cell.h>
```

Открытые члены

- `Cell` (const int a=0, const int b=0, const bool st=false)
Конструктор
- int `get_x` ()
Получить координату по оси x.
- int `get_y` ()
Получить координату по оси y.
- bool `get_status` ()
Получить состояние клетки
- void `set_x` (const int a)
Установить значение поля x, равное координате по оси x.
- void `set_y` (const int b)
Установить значение поля y, равное координате по оси y.
- void `set_status` (const bool st)
Установить значение поля status, равное состоянию клетки
- int `search_living` (const std::vector< std::vector< `Cell` > > c)
Подсчёт количества соседних живых клеток
- int `check` (`Cell`)
Проверка состояние клетки

3.2.1 Подробное описание

Класс Клетка

Этот класс моделирует объект клетка из которых будет состоять поле.

3.2.2 Конструктор(ы)

3.2.2.1 `Cell::Cell (const int a = 0, const int b = 0, const bool st = false) [inline]`

Конструктор

Аргументы

a	координата по оси x
b	координата по оси y
st	состояние клетки

3.2.3 Методы

3.2.3.1 int Cell::check (Cell c)

Проверка состояние клетки

Аргументы

проверяемая	клетка
-------------	--------

3.2.3.2 bool Cell::get_status () [inline]

Получить состояние клетки

Возвращает

состояние клетки

3.2.3.3 int Cell::get_x () [inline]

Получить координату по оси x.

Возвращает

координата по оси x

3.2.3.4 int Cell::get_y () [inline]

Получить координату по оси y.

Возвращает

координата по оси y

3.2.3.5 int Cell::search_living (const std::vector< std::vector< Cell > > c)

Подсчёт количества соседних живых клеток

Аргументы

c	двумерный массив клеток
---	-------------------------

3.2.3.6 void Cell::set_status (const bool st) [inline]

Установить значение поля status, равное состоянию клетки

Аргументы

st	статус клетки
----	---------------

3.2.3.7 void Cell::set_x (const int a) [inline]

Установить значение поля x, равное координате по оси x.

Аргументы

a	координата по оси x
---	---------------------

3.2.3.8 void Cell::set_y (const int b) [inline]

Установить значение поля y, равное координате по оси y.

Аргументы

b	координата по оси y
---	---------------------

Объявления и описания членов классов находятся в файлах:

- /home/user/LivingCells/sources/livingcells/core/cell.h
- /home/user/LivingCells/sources/livingcells/core/cell.cpp

3.3 Класс console_ui

Класс консольного меню

```
#include <console_ui.h>
```

Открытые члены

- void [print_menu](#) ()
Вывести главное меню
- void [secondary_menu](#) (Field &f, [Api](#) a)
Вывести промежуточное меню
- void [input_menu](#) (Field &f, [Api](#) a, int x, int y)
Вывести меню задания клеток поля

3.3.1 Подробное описание

Класс консольного меню

Этот класс, посредством консоли, взаимодействует с пользователем.

3.3.2 Методы

3.3.2.1 void console_ui::input_menu (Field &f, [Api](#) a, int x, int y)

Вывести меню задания клеток поля

Аргументы

f	поле
a	объект класса Api
x	размер поля по оси x
y	размер поля по оси y

3.3.2.2 void console_ui::secondary_menu (Field & f, Api a)

Вывести промежуточное меню

Аргументы

f	поле
a	объект класса Api

Объявления и описания членов классов находятся в файлах:

- /home/user/LivingCells/sources/livingcells/app/console_ui.h
- /home/user/LivingCells/sources/livingcells/app/console_ui.cpp

3.4 Класс exit_window

Граф наследования:exit_window:

Граф связей класса exit_window:

Открытые члены

- exit_window (QWidget *parent)

Объявления и описания членов классов находятся в файлах:

- /home/user/LivingCells/sources/livingcells/gui/exit_window.h
- /home/user/LivingCells/sources/livingcells/gui/exit_window.cpp

3.5 Класс Field

Класс Поле

```
#include <field.h>
```

Открытые члены

- [Field](#) (const int a=0, const int b=0)
Конструктор
- int [get_x](#) ()
Получить размер поля по оси x.
- int [get_y](#) ()
Получить размер поля по оси y.
- [Cell](#) [get_cell](#) (const int x_cell, const int y_cell)
Получить клетку из поля
- std::vector< std::vector< [Cell](#) > > [get_cells](#) ()

- Получить все клетки поля
- void `set_x` (const int a)
Установить значение поля x, равное размеру поля по оси x.
- void `set_y` (const int b)
Установить значение поля y, равное размеру поля по оси y.
- void `set_cell` (const int x_cell, const int y_cell, const int status)
Задать данные новой клетки
- void `set_cells` (const std::vector< std::vector< `Cell` > > c)
Задать массив клеток

3.5.1 Подробное описание

Класс Поле

Этот класс моделирует объект поле на котором располагаются клетки.

3.5.2 Конструктор(ы)

3.5.2.1 `Field::Field (const int a = 0, const int b = 0) [inline]`

Конструктор

Аргументы

a	размер поля по оси x
b	размер поля по оси y

3.5.3 Методы

3.5.3.1 `Cell Field::get_cell (const int x_cell, const int y_cell) [inline]`

Получить клетку из поля

Аргументы

x_cell	координата клетки по x
y_cell	координата клетки по y

Возвращает

клетки из поля

3.5.3.2 `std::vector<std::vector<Cell> > Field::get_cells () [inline]`

Получить все клетки поля

Возвращает

двумерный массив клеток

3.5.3.3 `int Field::get_x () [inline]`

Получить размер поля по оси x.

Возвращает

размер поля по оси x

3.5.3.4 `int Field::get_y () [inline]`

Получить размер поля по оси y.

Возвращает

размер поля по оси y

3.5.3.5 `void Field::set_cell (const int x_cell, const int y_cell, const int status)`

Задать данные новой клетки

Аргументы

<code>x_cell</code>	координата клетки по x
<code>y_cell</code>	координата клетки по y
<code>status</code>	состояние клетки

3.5.3.6 `void Field::set_cells (const std::vector< std::vector< Cell > > c) [inline]`

Задать массив клеток

Аргументы

<code>c</code>	двумерный массив клеток
----------------	-------------------------

3.5.3.7 `void Field::set_x (const int a)`

Установить значение поля x, равное размеру поля по оси x.

Аргументы

<code>a</code>	размер поля по оси x
----------------	----------------------

3.5.3.8 `void Field::set_y (const int b)`

Установить значение поля y, равное размеру поля по оси y.

Аргументы

<code>b</code>	размер поля по оси y
----------------	----------------------

Объявления и описания членов классов находятся в файлах:

- `/home/user/LivingCells/sources/livingcells/core/field.h`
- `/home/user/LivingCells/sources/livingcells/core/field.cpp`

3.6 Класс `field_window`

Граф наследования:`field_window`:

Граф связей класса `field_window`:

Открытые члены

- field_window (QWidget *parent, [Api](#) a, [Field](#) f)
- void print_field ()

Объявления и описания членов классов находятся в файлах:

- /home/user/LivingCells/sources/livingcells/gui/field_window.h
- /home/user/LivingCells/sources/livingcells/gui/field_window.cpp

3.7 Класс figure_window

Граф наследования:figure_window:

Граф связей класса figure_window:

Открытые члены

- figure_window (QWidget *parent)

Объявления и описания членов классов находятся в файлах:

- /home/user/LivingCells/sources/livingcells/gui/figure_window.h
- /home/user/LivingCells/sources/livingcells/gui/figure_window.cpp

3.8 Класс initialize_window

Граф наследования:initialize_window:

Граф связей класса initialize_window:

Открытые члены

- initialize_window (QWidget *parent, [Api](#) *a, [Field](#) *f, int i, int j)

Объявления и описания членов классов находятся в файлах:

- /home/user/LivingCells/sources/livingcells/gui/initialize_window.h
- /home/user/LivingCells/sources/livingcells/gui/initialize_window.cpp

3.9 Класс MainWindow

Граф наследования:MainWindow:

Граф связей класса MainWindow:

Открытые члены

- MainWindow (QWidget *parent)

Объявления и описания членов классов находятся в файлах:

- /home/user/LivingCells/sources/livingcells/gui/mainwindow.h
- /home/user/LivingCells/sources/livingcells/gui/mainwindow.cpp

3.10 Структура qt_meta_stringdata_exit_window_t

Открытые атрибуты

- QByteArrayData data [4]
- char stringdata0 [41]

Объявления и описания членов структуры находятся в файле:

- /home/user/LivingCells/sources/build-livingcells-Desktop_Qt_5_5_0_GCC_32bit-Debug/gui/moc↔_exit_window.cpp

3.11 Структура qt_meta_stringdata_field_window_t

Открытые атрибуты

- QByteArrayData data [5]
- char stringdata0 [59]

Объявления и описания членов структуры находятся в файле:

- /home/user/LivingCells/sources/build-livingcells-Desktop_Qt_5_5_0_GCC_32bit-Debug/gui/moc↔_field_window.cpp

3.12 Структура qt_meta_stringdata_figure_window_t

Открытые атрибуты

- QByteArrayData data [4]
- char stringdata0 [46]

Объявления и описания членов структуры находятся в файле:

- /home/user/LivingCells/sources/build-livingcells-Desktop_Qt_5_5_0_GCC_32bit-Debug/gui/moc↔_figure_window.cpp

3.13 Структура qt_meta_stringdata_initialize_window_t

Открытые атрибуты

- QByteArrayData data [4]
- char stringdata0 [59]

Объявления и описания членов структуры находятся в файле:

- /home/user/LivingCells/sources/build-livingcells-Desktop_Qt_5_5_0_GCC_32bit-Debug/gui/moc↔_initialize_window.cpp

3.14 Структура qt_meta_stringdata_MainWindow_t

Открытые атрибуты

- QByteArrayData data [7]
- char stringdata0 [79]

Объявления и описания членов структуры находятся в файле:

- /home/user/LivingCells/sources/build-livingcells-Desktop_Qt_5_5_0_GCC_32bit-Debug/gui/moc_↵_mainwindow.cpp

3.15 Структура qt_meta_stringdata_size_window_t

Открытые атрибуты

- QByteArrayData data [4]
- char stringdata0 [43]

Объявления и описания членов структуры находятся в файле:

- /home/user/LivingCells/sources/build-livingcells-Desktop_Qt_5_5_0_GCC_32bit-Debug/gui/moc_↵_size_window.cpp

3.16 Класс size_window

Граф наследования: size_window:

Граф связей класса size_window:

Открытые члены

- size_window (QWidget *parent, int mode)

Объявления и описания членов классов находятся в файлах:

- /home/user/LivingCells/sources/livingcells/gui/size_window.h
- /home/user/LivingCells/sources/livingcells/gui/size_window.cpp

3.17 Класс TestTest

Граф наследования: TestTest:

Граф связей класса TestTest:

Объявления и описания членов класса находятся в файле:

- /home/user/LivingCells/sources/livingcells/test/tst_testtest.cpp

Предметный указатель

Api, [5](#)

Cell, [7](#)

Cell, [7](#)

check, [8](#)

check

Cell, [8](#)

Field, [10](#)

Field, [11](#)