Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ

УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)

Кафедра автоматизации обработки информации (АОИ)

**Игра «Жизнь»**

Пояснительная записка к курсовой работе по дисциплине

«Практикум по объектно-ориентированному программированию»

Выполнил

студент гр. 429-3

\_\_\_\_\_\_\_\_\_\_\_\_\_ Бабец А.А.

«\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_2021 г.

Руководитель

доцент каф. АОИ, к.ф.-м.н.

\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_ Зариковская Н.В.

Оценка

«\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_2021 г.

Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)

Кафедра автоматизации и обработки информации (АОИ)\

УТВЕРЖДАЮ

Зав. кафедрой АОИ, к.э.н. А.А.Сидоров

00000000000000000000

0--0000000(подпись)

ЗАДАНИЕ

на курсовую работу по дисциплине

«Практикум по объектно-ориентированному программированию»

студенту группы 429-3 ФСУ Бабец Алексею Алексеевичу

1. Тема проекта: Тема проекта
2. Срок сдачи студентом законченной работы: «03» июля 2021 г.
3. Содержание курсовой работы:
   1. Постановка задачи
   2. Анализ задачи
   3. Алгоритм и блок-схема
   4. Выбор стека технологий
   5. Реализация основных процедур и функций
   6. Интерфейс приложения
   7. Тестирование приложения.

Дата выдачи задания: «12» февраля 2021 г.

Руководитель:

|  |  |  |
| --- | --- | --- |
| Доцент кафедры АОИ, к.ф.-м.н., |  | 0000000000000000 Н. В. Зариковская |

Задание принял к исполнению «12» февраля 2021 г.

|  |  |  |
| --- | --- | --- |
| Студент гр. 429-3 |  | 0000000000000000 А.А.Бабец |

**Оглавление**

[1 Постановка задачи 4](#_Toc75863687)

[2 Анализ задачи 5](#_Toc75863688)

[3 Алгоритм и блок-схема 6](#_Toc75863689)

[4 Стек технологий 8](#_Toc75863690)

[5 Описание реализации основных процедур и функций 10](#_Toc75863691)

[6 Описание интерфейса программы 13](#_Toc75863692)

[7 Тестирование программы 16](#_Toc75863693)

[8 Заключение 25](#_Toc75863694)

[Список использованных источников 26](#_Toc75863695)

# Постановка задачи

Реализовать игру «Жизнь». Разрабатываемое приложение представляет собой программную реализацию известной клеточной игры. Приложение должно обеспечивать возможность сохранения текущего состояния поля в файл, загрузки сохранений, редактирование поля, просмотр процесса жизни поля.

# Анализ задачи

Рассмотрим более подробно процесс игры «Жизнь»:

* существует поле жизни, состоящее из клеток, имеющих два состояния «живая» (зелёная) и «мёртвая» (белая);
* когда начинается «жизнь» генерируется следующее поколение «живых» клеток;
* в зависимости от количества «живых» соседей клетка может умереть или ожить;
* «жизнь» заканчивается, когда появится новое поколение, не отличающееся от текущего.

# Алгоритм и блок-схема

Далее представлена функциональная схема приложения.

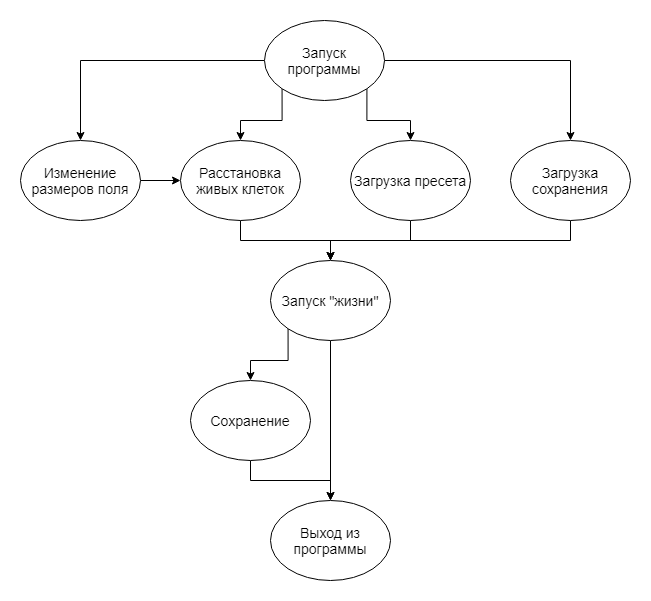


Рисунок 3.1 – Функциональная схема приложения

Основной функцией данной программы является функция, генерирующая следующее поколение клеток. Ниже описан её алгоритм.

Для каждой клетки происходит проверка всех восьми соседних клеток, при этом считается количество «живых» соседей. Клетки, находящиеся вне видимого поля (некоторые соседи крайних клеток) считаются мёртвыми. После подсчёта «живых» соседей идёт генерация по следующим правилам:

* если у «мёртвой» клетки ровно три живых соседа, то она будет «живой» в следующем поколении;
* если у «живой» клетки меньше двух или больше трёх «живых» соседей, то она будет «мёртвой» в следующем поколении [1].

Блок-схема представлена на рисунке 3.1.

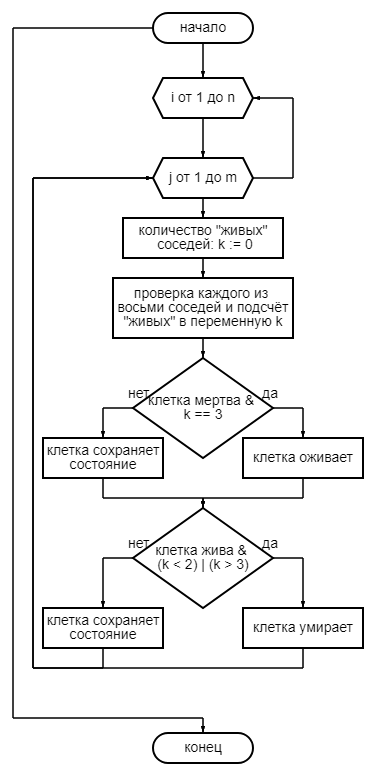


Рисунок 3.1 – Блок-схема метода генерации нового поколения

На основе созданной блок-схемы составлена функция генерации нового поколения.

# Стек технологий

Для написания программы был выбран язык C++. Далее рассмотрены его преимущества:

* поддержка объектно-ориентированного программирования (ООП), что помогает сделать код проще, и его быстрее писать;
* высокая скорость;
* возможности для работы с данными на низком уровне — то есть на уровне, близком к аппаратному, благодаря этому на**С++** -можно писать драйвера, микроконтроллеры;
* популярность:
  + для **С++** создано много библиотек и компиляторов;
  + **С++** используется практически везде;
* синтаксис **С++** похож насинтаксис **C, С#** и **Java,** так что переключаться между этими языками достаточно легко;
* совместимость с **C** благодаря тому, что **С++** создавался на его основе [3];
* более надежное освобождение ресурсов путем автоматического вызова деструкторов при уничтожении объектов;
* важным достоинством является предсказуемое выполнение программ, что является важным для систем реального времени;
* C++ - кросплатформенный язык, его стандарт накладывает минимальные требования на компьютер для запуска скомпилированной программы. На С++ разрабатывают программы для различных платформ и систем с помощью компиляторов для большого количества платформ;
* язык С++ спроектирован так, чтобы у программиста был максимальный контроль над всеми аспектами структуры и порядка выполнения программы [2].

Также выбрана среда разработки QtCreator. Её преимущества описаны ниже:

* это свободное ПО;
* бесплатность;
* кроссплатформенность, что позволяет работать с различными ОС;
* скорость работы;
* стабильность;
* размер установки (в сравнении с Visual Studio);
* функция автодополнения;
* возможность автоматического обновления через интернет [3].

# Описание реализации основных процедур и функций

Для решения поставленной задачи опишем реализацию следующих процедур и функций:

* генерация следующего поколения public int nextGen(), класса life\_place;
* перерисовка поля по текущему состоянию поля void drawplace(bool new), класса mainwindow;

int life\_place**::**nextGen**()** **{**

int k **=** **-**1**;**

QVector**<**QVector**<**int**>>** newplace**;**

newplace**.**resize**(**height**+**2**);**

**for(**int i **=** 0**;** i**<**height**+**2**;** i**++)** **{**

newplace**[**i**].**resize**(**width**+**2**);**

**for(**int j **=** 0**;** j**<**width**+**2**;** j**++)**

newplace**[**i**][**j**]** **=** 0**;**

**}**

**for(**int i **=** 1**;** i**<=**height**;** i**++)**

**for(**int j **=** 1**;** j**<=**width**;** j**++)** **{**

int nb **=** 0**;**

**if(**life\_place**::**place**[**i**-**1**][**j**-**1**]** **>** 0**)** nb**++;**

**if(**life\_place**::**place**[**i**-**1**][**j**]** **>** 0**)** nb**++;**

**if(**life\_place**::**place**[**i**-**1**][**j**+**1**]** **>** 0**)** nb**++;**

**if(**life\_place**::**place**[**i**][**j**-**1**]** **>** 0**)** nb**++;**

**if(**life\_place**::**place**[**i**][**j**+**1**]** **>** 0**)** nb**++;**

**if(**life\_place**::**place**[**i**+**1**][**j**-**1**]** **>** 0**)** nb**++;**

**if(**life\_place**::**place**[**i**+**1**][**j**]** **>** 0**)** nb**++;**

**if(**life\_place**::**place**[**i**+**1**][**j**+**1**]** **>** 0**)** nb**++;**

**if(**life\_place**::**place**[**i**][**j**]==**0**)** **{**

**if(**nb **==** 3**)** **{**

newplace**[**i**][**j**]=** 1**;**

k**++;**

**}**

**}**

**else** **{**

**if(**nb**<**2 **||** nb**>**3**)** **{**

newplace**[**i**][**j**]** **=** 0**;**

k**++;**

**}**

**else**

newplace**[**i**][**j**]** **=** 1**;**

**}**

**}**

life\_place**::**place **=** newplace**;**

**return** k**;**

**}**

Генерация следующего поколения происходит с помощью двух вложенных циклов for следующим образом:

При первом и единственном полном проходе по всему полю для каждой клетки считается количество её «живых» соседей, далее, в зависимости от количества соседей и того, «жива» ли клетка на данный момент, принимается решение о её состоянии в следующем поколении. Тут действуют следующие правила:

* если у «мёртвой» клетки ровно три живых соседа, то она будет «живой» в следующем поколении;
* если у «живой» клетки меньше двух или больше трёх «живых» соседей, то она будет «мёртвой» в следующем поколении.

Так в процессе пересчёта нового поколения происходит подсчёт клеток, состояние которых изменилось, это количество является выводом функции. Как только будет сгенерированное новое поколение без изменений, обработчик события таймера остановит пересчёт и переотрисовку.

void MainWindow**::**drawplace**(**bool first**)**

**{**

int n **=** place**.**getHeight**(),** m **=** place**.**getWidth**(),** nh **=** lh**/**n**,** mh **=** lw**/**m**;**

lh **=** nh**\***n**;**

lw **=** mh**\***m**;**

QVector**<**QVector**<**int**>>** nowplace **=** place**.**getPlace**();**

QPainter pen**;**

QPixmap lplace**;**

**if(**first**)** lplace **=** QPixmap**(**lw**,** lh**);**

**else** lplace **=** QPixmap**(**ui**->**life\_place**->**pixmap**());**

pen**.**begin**(&**lplace**);**

pen**.**eraseRect**(**0**,**0**,**lw**,**lh**);**

pen**.**setPen**(**QPen**(**Qt**::**black**,** 1**,** Qt**::**SolidLine**));**

**for(**int i **=** 0**;** i**<**lw**;** i**+=**mh**)**

pen**.**drawLine**(**i**,**0**,**i**,**lh**);**

**for(**int i **=** 0**;** i**<**lh**;** i**+=**nh**)**

pen**.**drawLine**(**0**,**i**,**lw**,**i**);**

pen**.**setBrush**(**QBrush**(**Qt**::**green**));**

**for(**int i **=** 1**;** i**<=**n**;** i**++)**

**for(**int j **=** 1**;** j**<=**m**;** j**++)**

**if(**nowplace**[**i**][**j**]** **==** 1**)**

pen**.**drawRect**(**QRect**((**j**-**1**)\***mh**,** **(**i**-**1**)\***nh**,** mh**,** nh**));**

pen**.**end**();**

ui**->**life\_place**->**setPixmap**(**lplace**);**

**}**

Переотрисовка поля начинается с копирования объекта класса QPixmap с объекта класса QExLabel или с создания нового объекта класса QPixmap, в зависимости от значения входной переменной, далее идёт инициализация объекта класса QPainter, с помощью которого и будет производиться вся отрисовка. Затем происходит очистка полученного QPixmap методом eraseRect(int x, int y, int width, int height) класса QPainter. Далее с помощью двух циклов for и метода drawLine(int x1, int y1, int x2, int y2) класса QPainter рисуется сетка поля. Затем с помощью двух вложенных циклов for производится обход матрицы поля и отрисовка зелёных «живых» клеток на соответствующих координатах сетки с помощью метода drawRect(QRect rectangle) класса QPainter. В конце функции объекту класса QExLabel находящемуся в главном окне программы присваивается изменённый QPixmap, который и отображается.

# Описание интерфейса программы

При нажатии на исполняемый файл «life\_game» в директории «build-life\_game-Desktop-Release» открывается окно программы, скриншот которого представлен на рисунке 6.1.

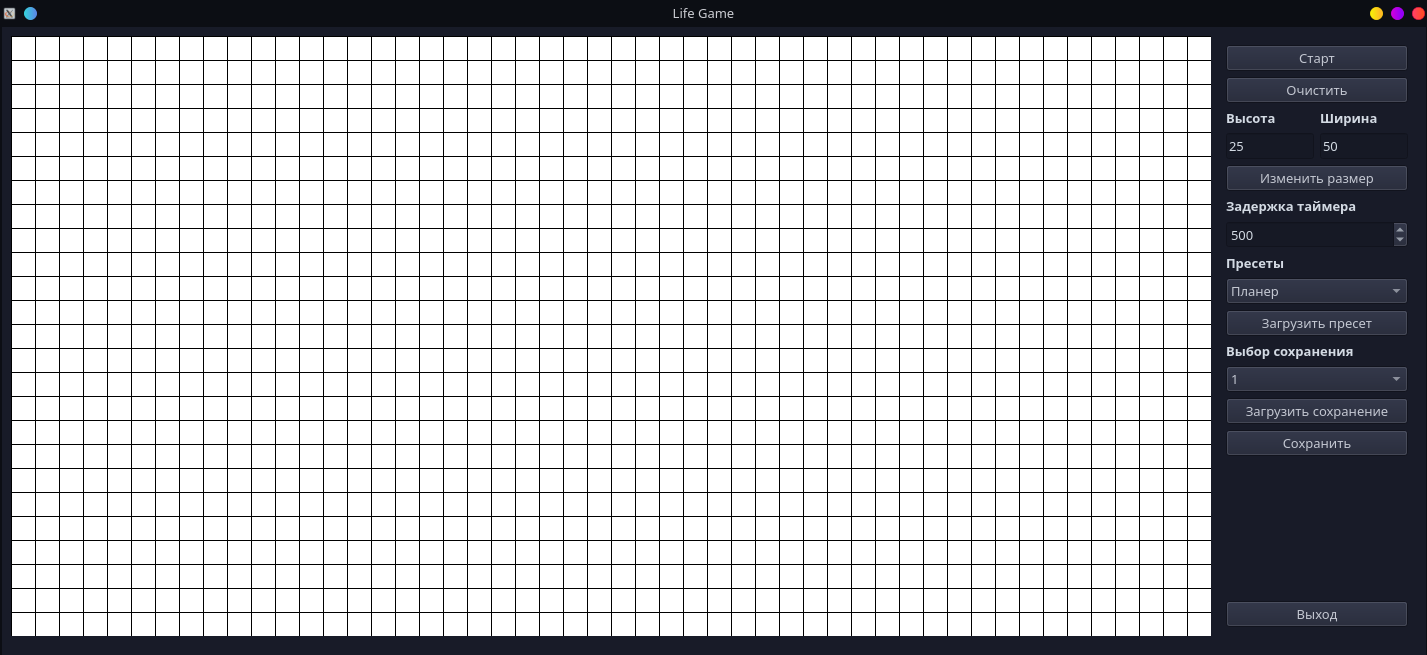


Рисунок 6.1 – Интерфейс программы

Слева находится само «поле» жизни, на котором и отрисовываются все клетки. Так же на этом поле пользователь может сам расставлять живые клетки с помощью оптического манипулятора типа мышь.

Далее рассмотрим правую часть более подробно.

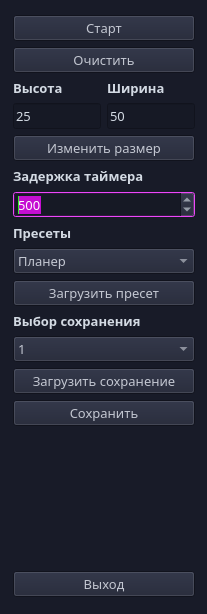


Рисунок 6.2 – Правая часть интерфейса.

Кнопка «Старт» (она же «Стоп») запускает или останавливает таймер, с помощью которого, реализован процесс «жизни», то есть просчёта следующего поколения и переотрисовки поля по таймеру.

Кнопка «Очистить» очищает поле от живых клеток.

С помощью объектов QTextEdit, подписанных с помощью объектов QLabel, и кнопки «Изменить размер» пользователь может изменить размер поля. Путём ввода в соответствующие поля желаемого количества клеток в высоту и в ширину.

С помощью объекта QComboBox, подписанного как «Пресеты», можно выбрать один из имеющихся пресетов (заготовленных файлов с фигурами) поля.

Кнопка «Загрузить пресет» загружает выбранный пресет из соответствующего файла и отрисовывает его.

Второй объект QComboBox позволяет выбрать одно из имеющихся сохранений для загрузки.

Кнопка «Загрузить сохранение» загружает из файла и отрисовывает выбранное сохранение.

Кнопка «Сохранить» сохраняет текущее состояние поля в файл.

# Тестирование программы

Тестирование проводилось с различными входными данными с целью выявления и предотвращения ошибок в работе программы. При представлении тестов были пропущены некоторые состояния поля для уменьшения количества картинок.

Первый тест: входные данные – рисунок 7.1. Дальнейшие состояния поля рисунки 7.2–7.6.

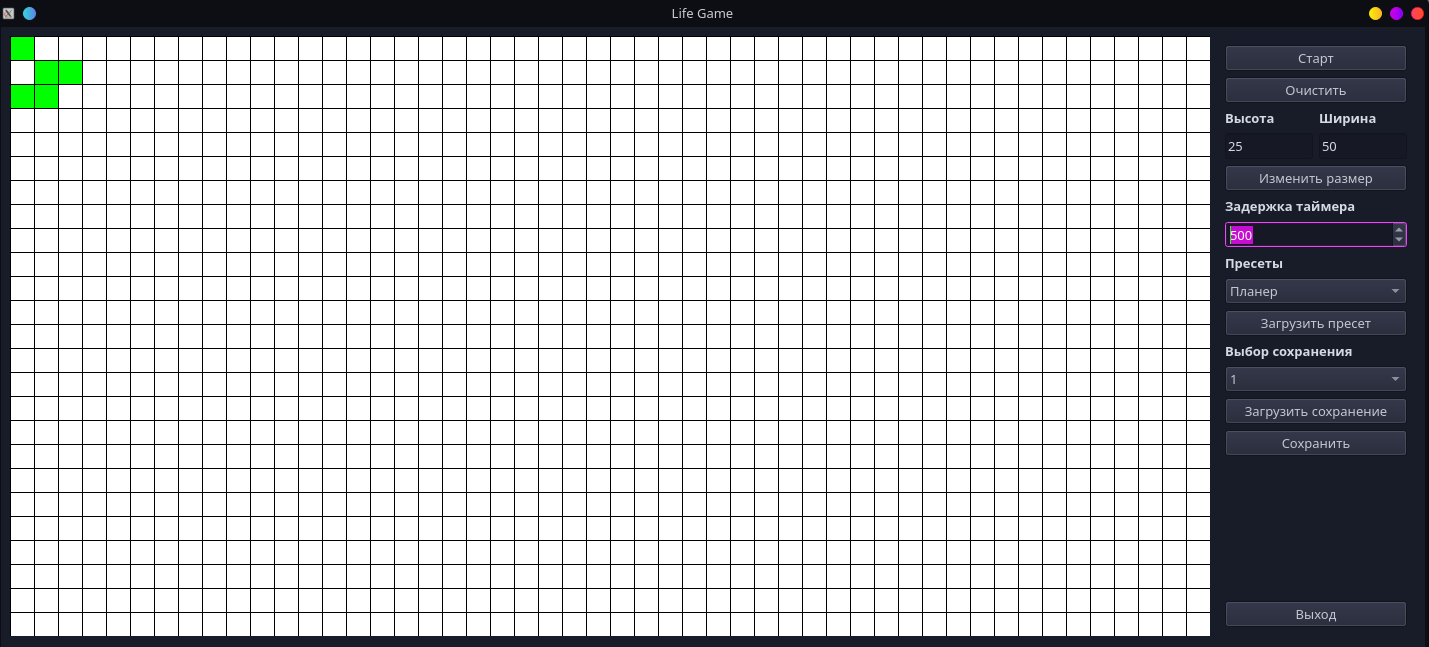


Рисунок 7.1 – Скриншот программы с «планером»

Планер – фигура, движущаяся по диагонали, в данной ситуации, вправо-вниз.

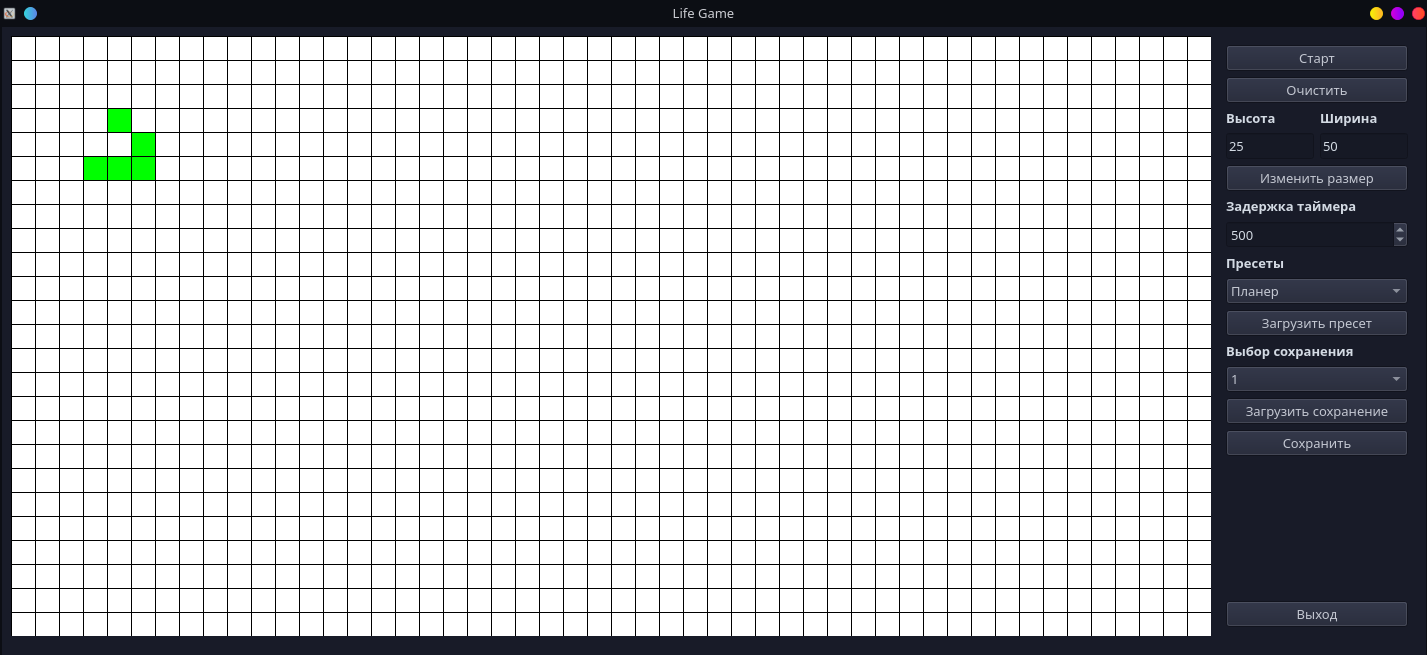


Рисунок 7.2 - Скриншот процесса работы программы

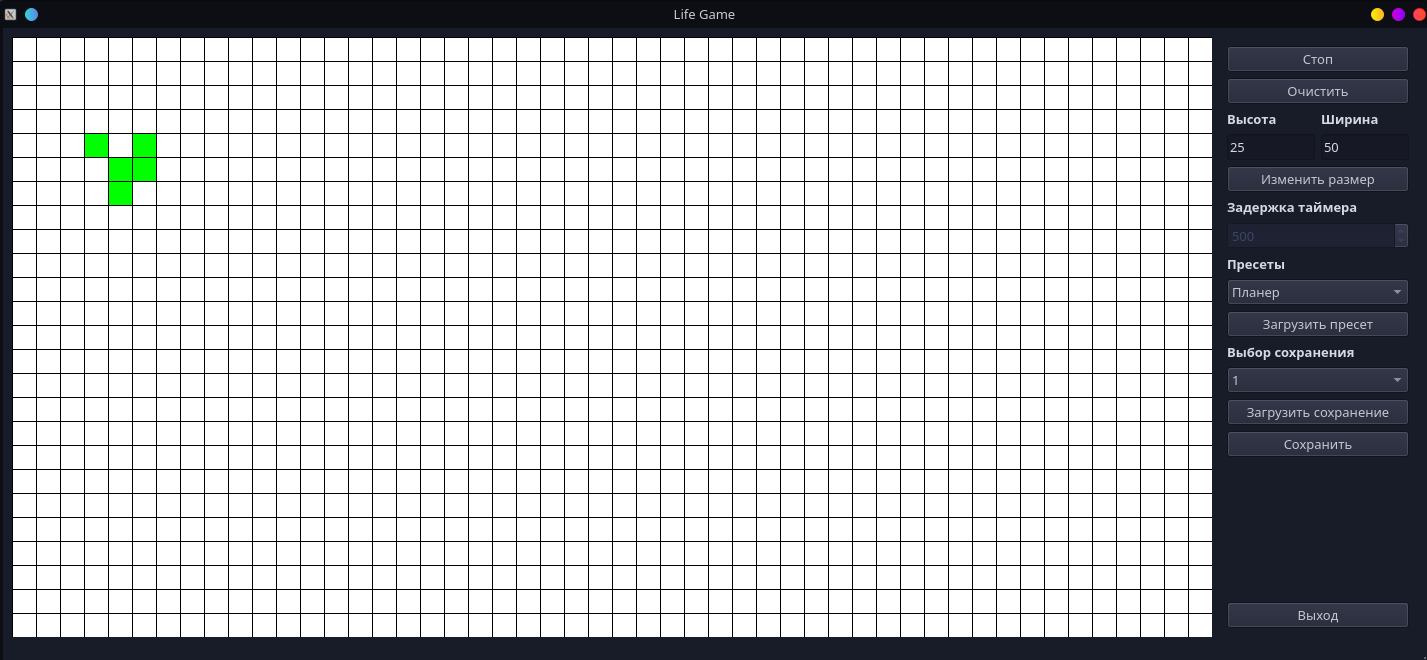


Рисунок 7.3 – Скриншот процесса работы программы

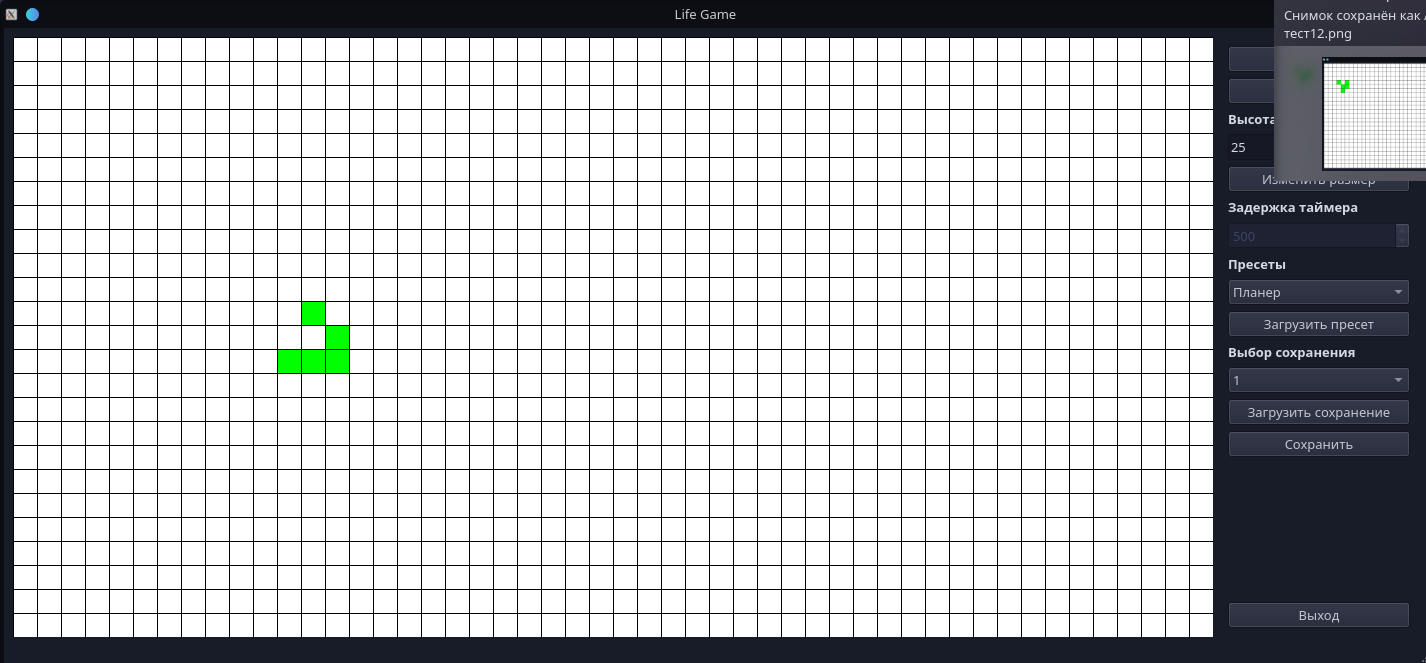


Рисунок 7.4 - Скриншот процесса работы программы

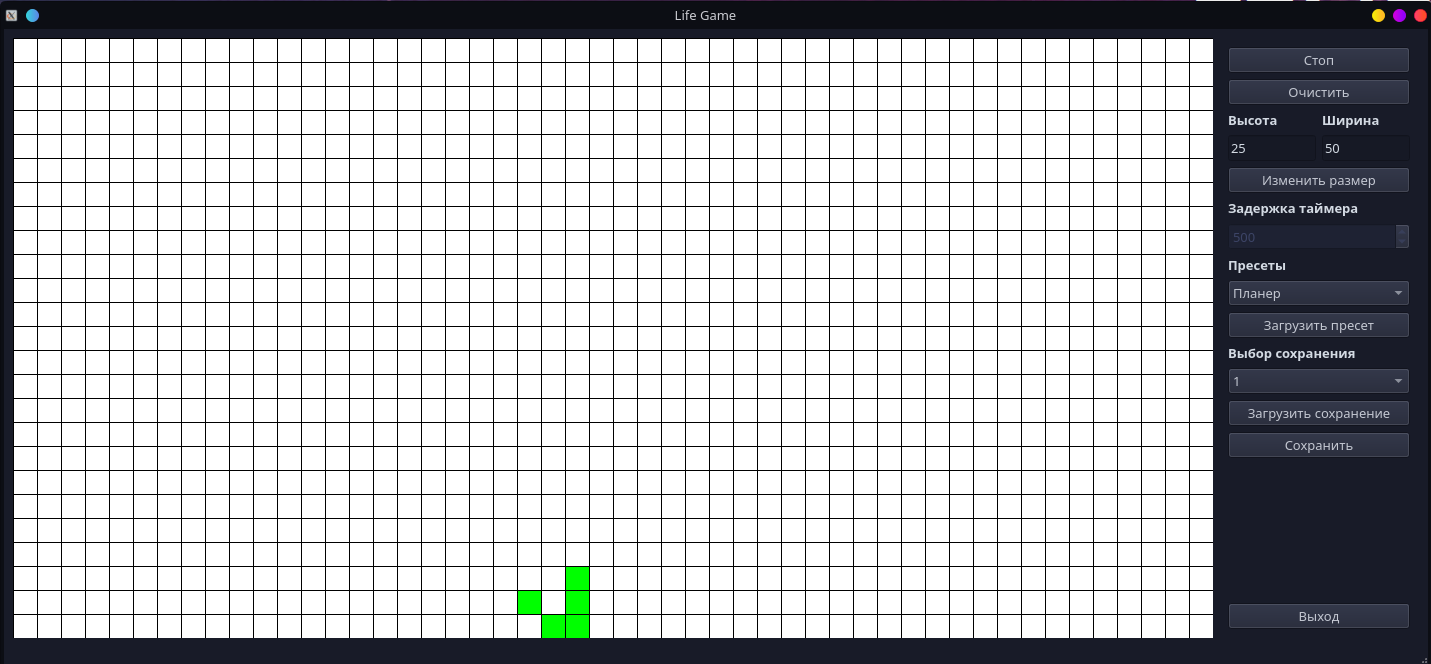


Рисунок 7.5 - Скриншот процесса работы программы

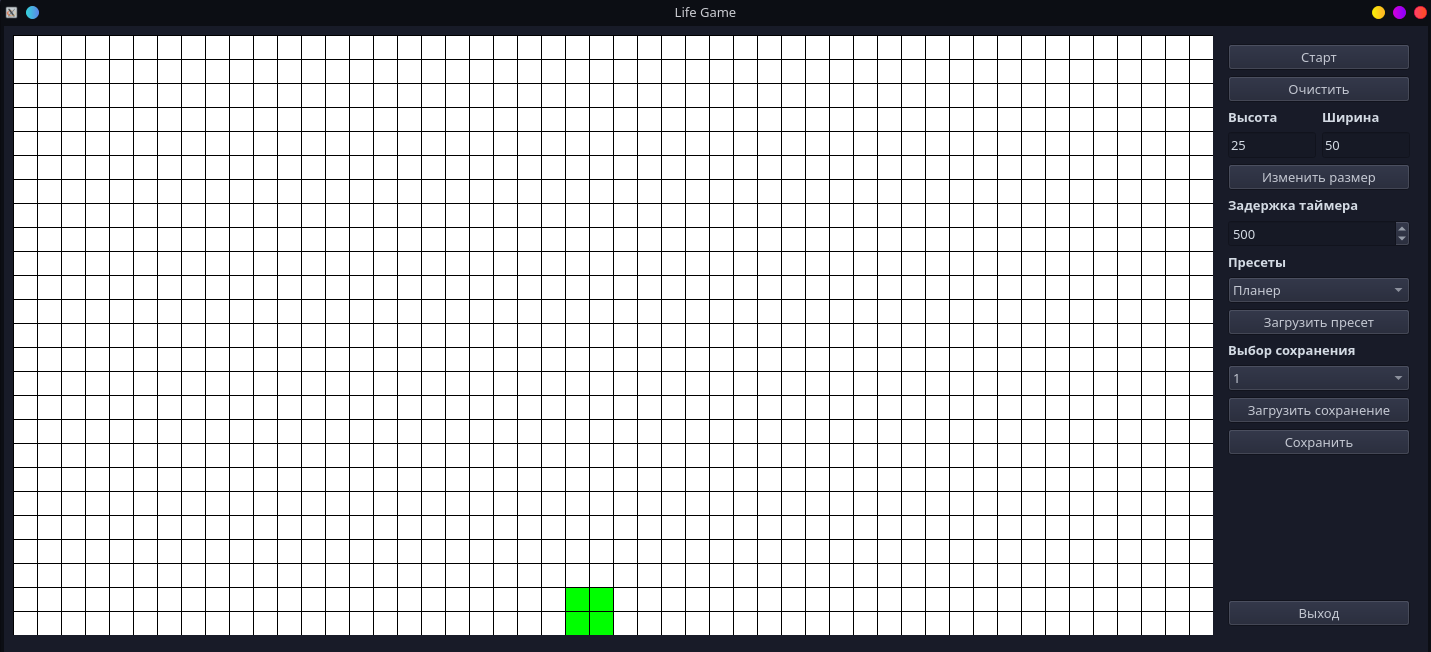


Рисунок 7.6 – Конец первого теста

В первом тесте наблюдается перемещение планера вниз и вправо, как и должно происходить.

Тест второй: использован пресет, содержащий планерное ружьё Госпера. Данная фигура бесконечно генерирует планеры из первого теста. Рисунок 7.7 – входные данные. Дальнейшие состояния поля - рисунки 7.8 и 7.9.

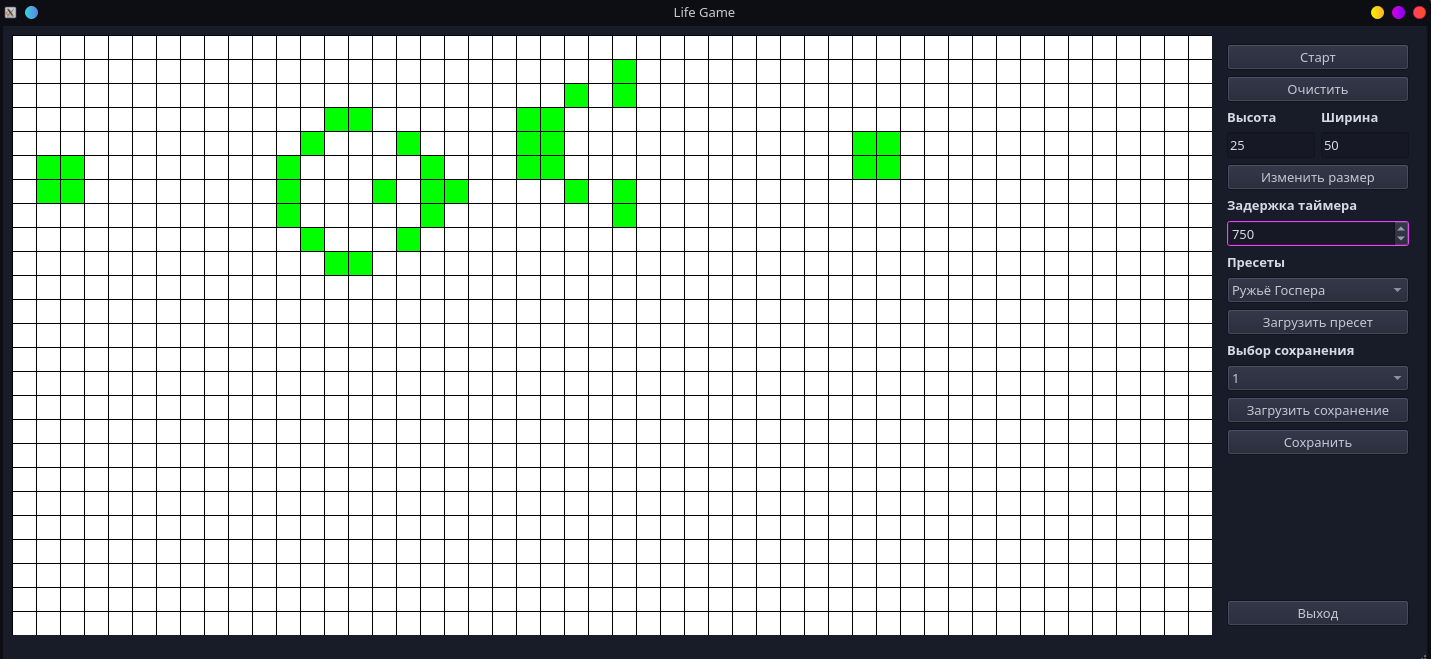


Рисунок 7.7 – Скриншот программы с ружьём Госпера

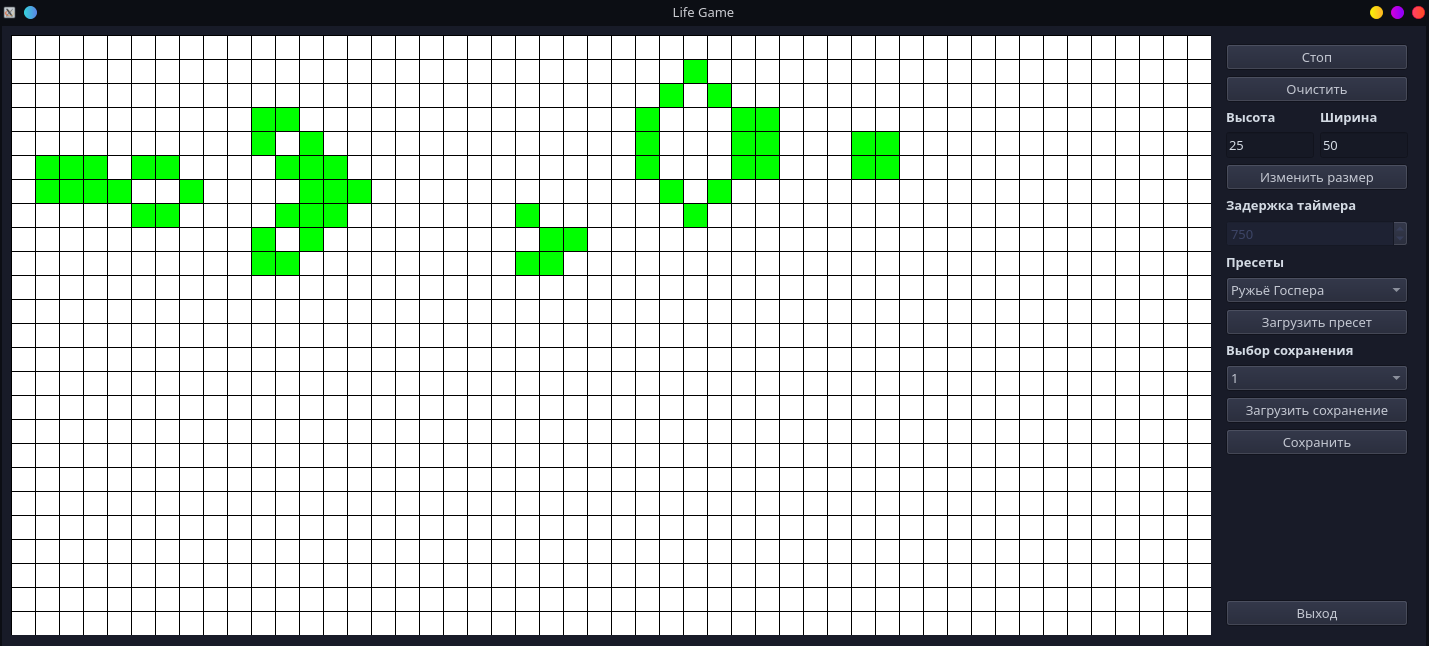


Рисунок 7.8 - Скриншот процесса работы программы

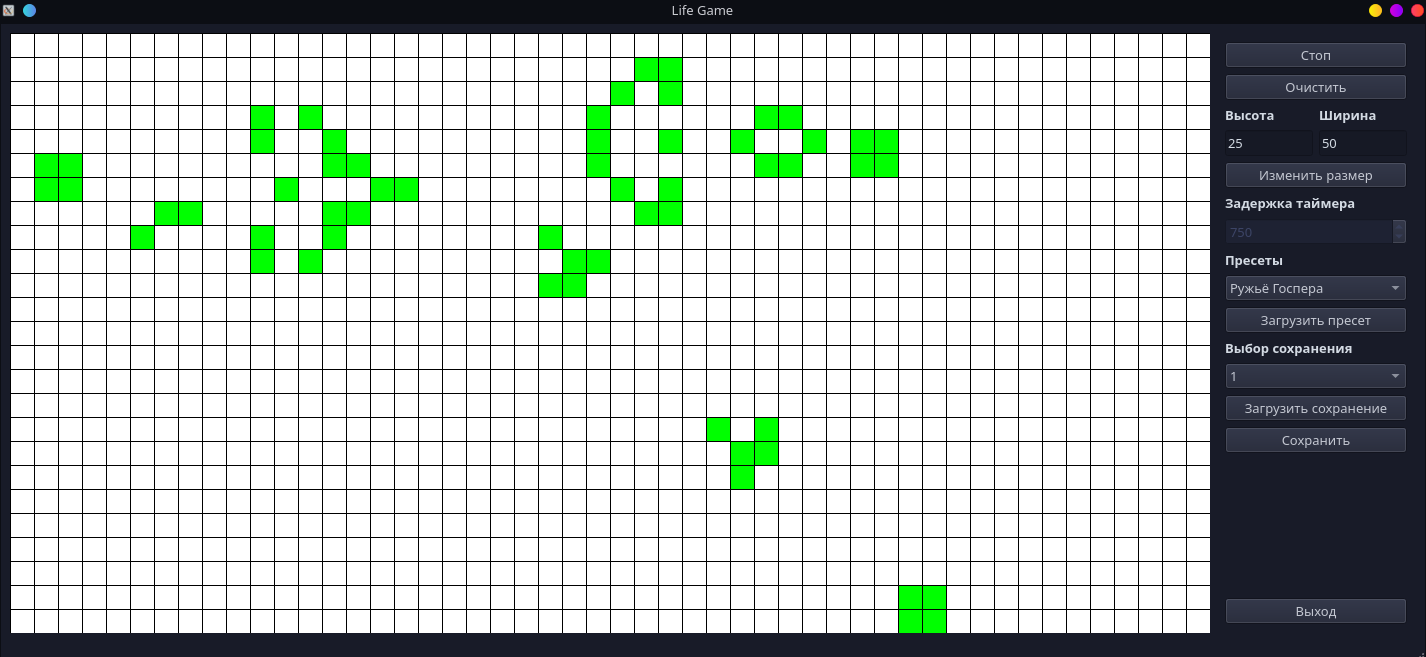


Рисунок 7.9 – Конец второго теста

Так как планерное ружьё Госпера может работать бесконечно, тест был остановлен, когда первый планер «разбился». По итогам теста мы наблюдаем ожидаемое поведение фигуры.

Тест третий: использована фигура, обнаруженная при тестировании во время написания программы. Рисунок 7.10 – входные данные. Дальнейшие состояния поля - рисунки 7.11 – 7.16.

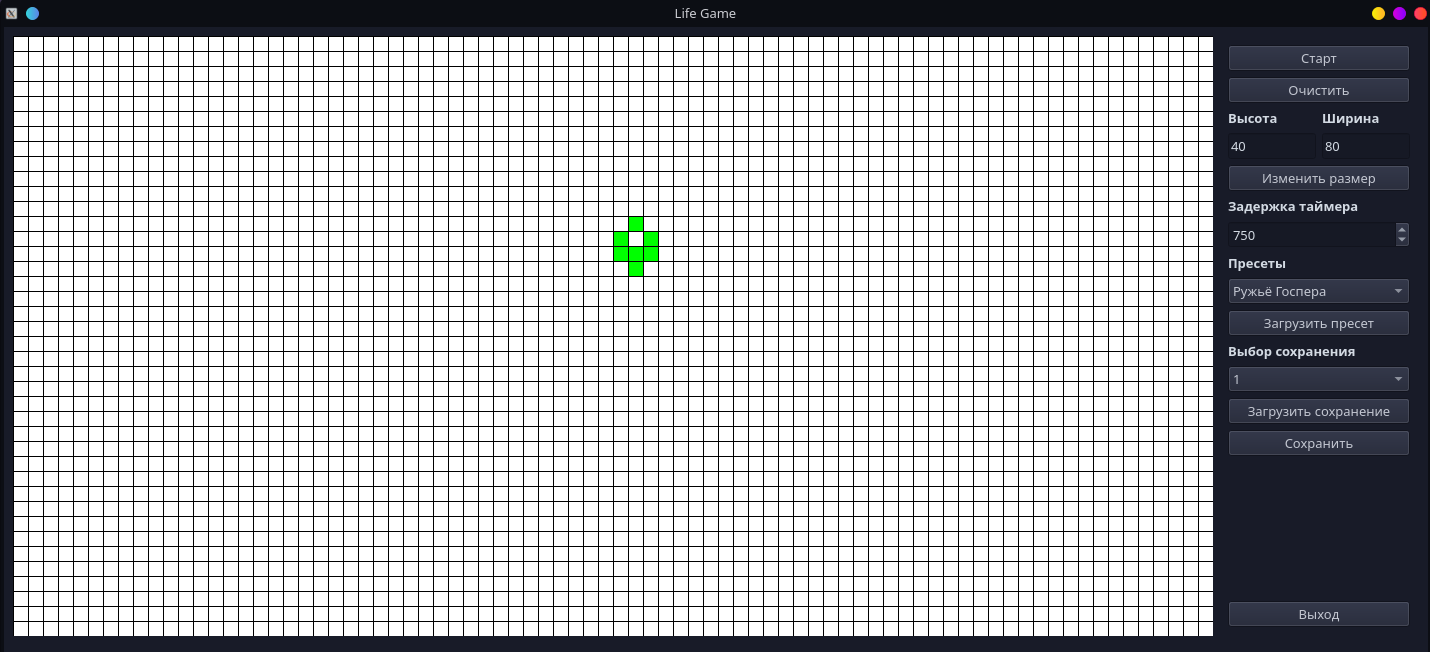


Рисунок 7.10 – Скриншот программы с фигурой

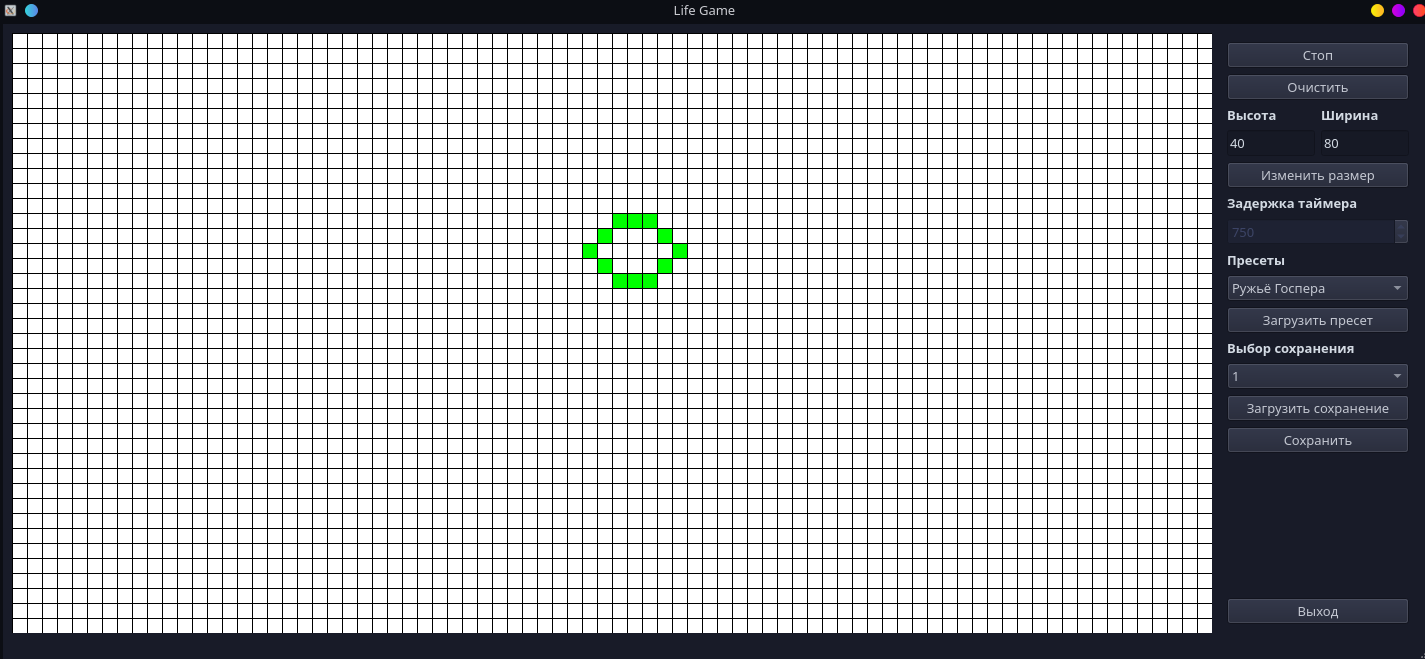


Рисунок 7.11 - Скриншот процесса работы программы

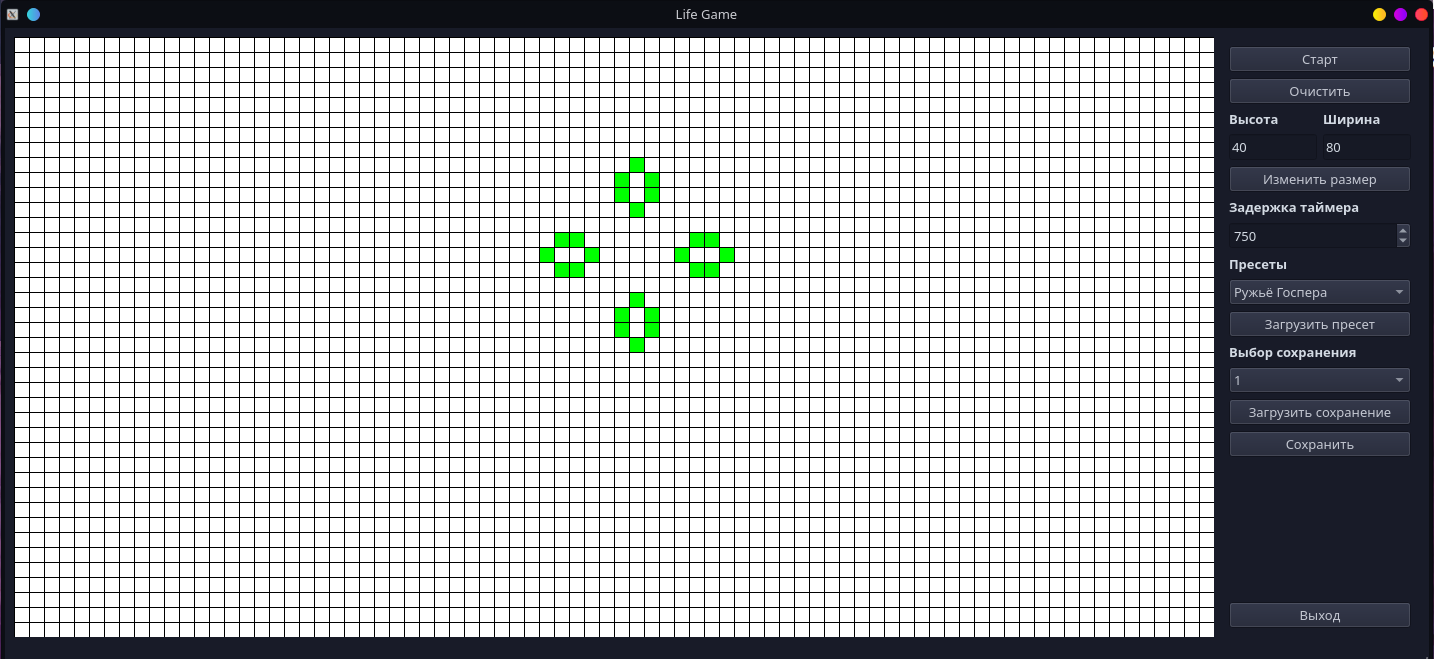


Рисунок 7.12 - Скриншот процесса работы программы

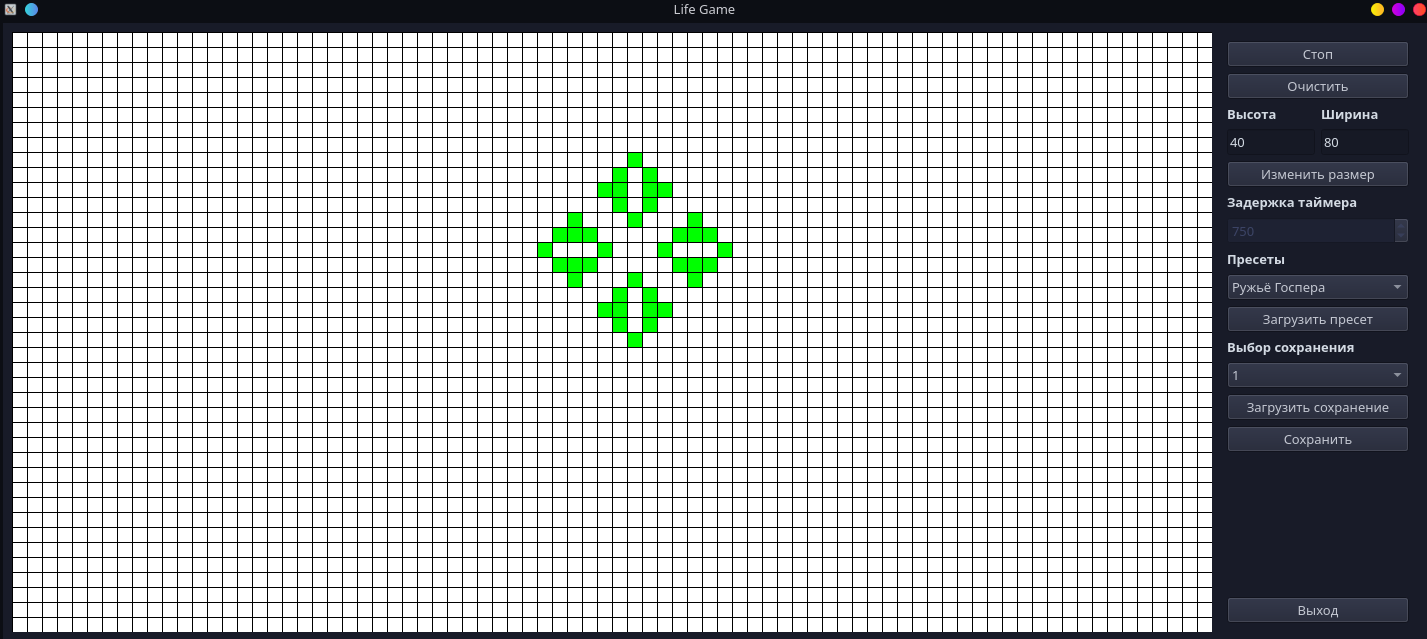


Рисунок 7.13 - Скриншот процесса работы программы

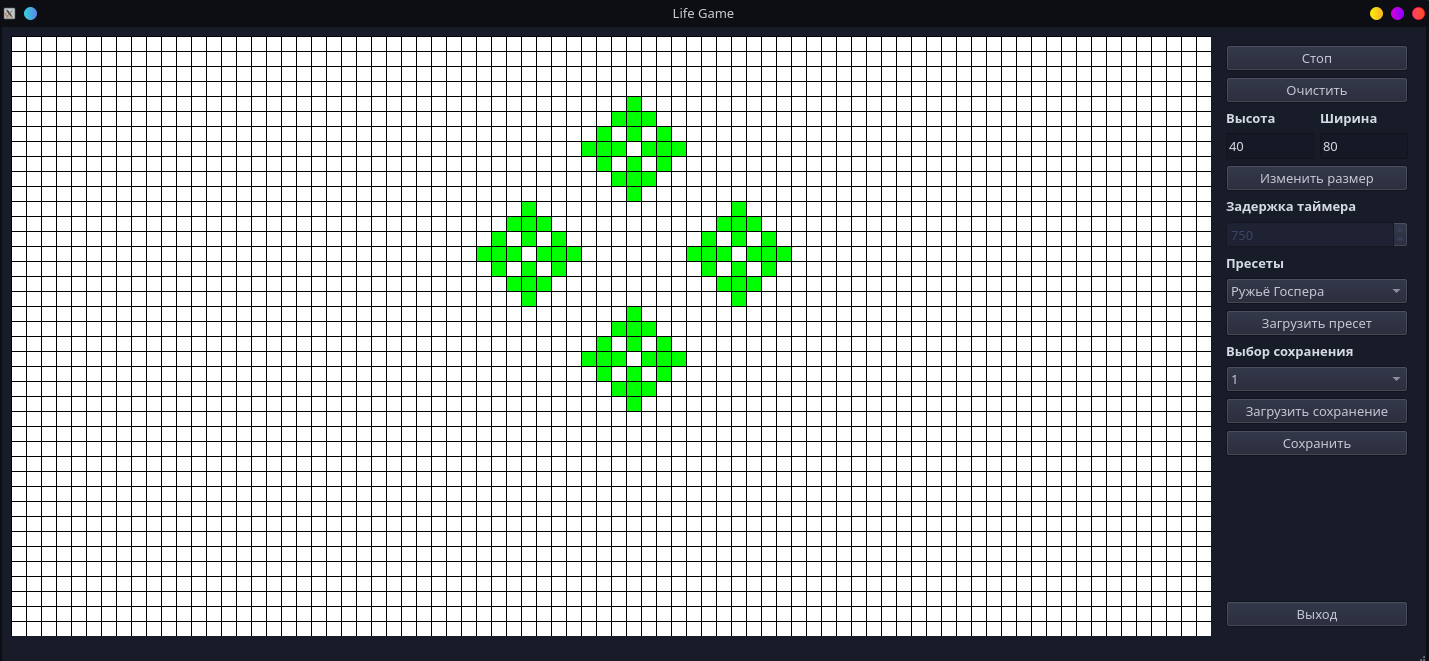


Рисунок 7.14 - Скриншот процесса работы программы

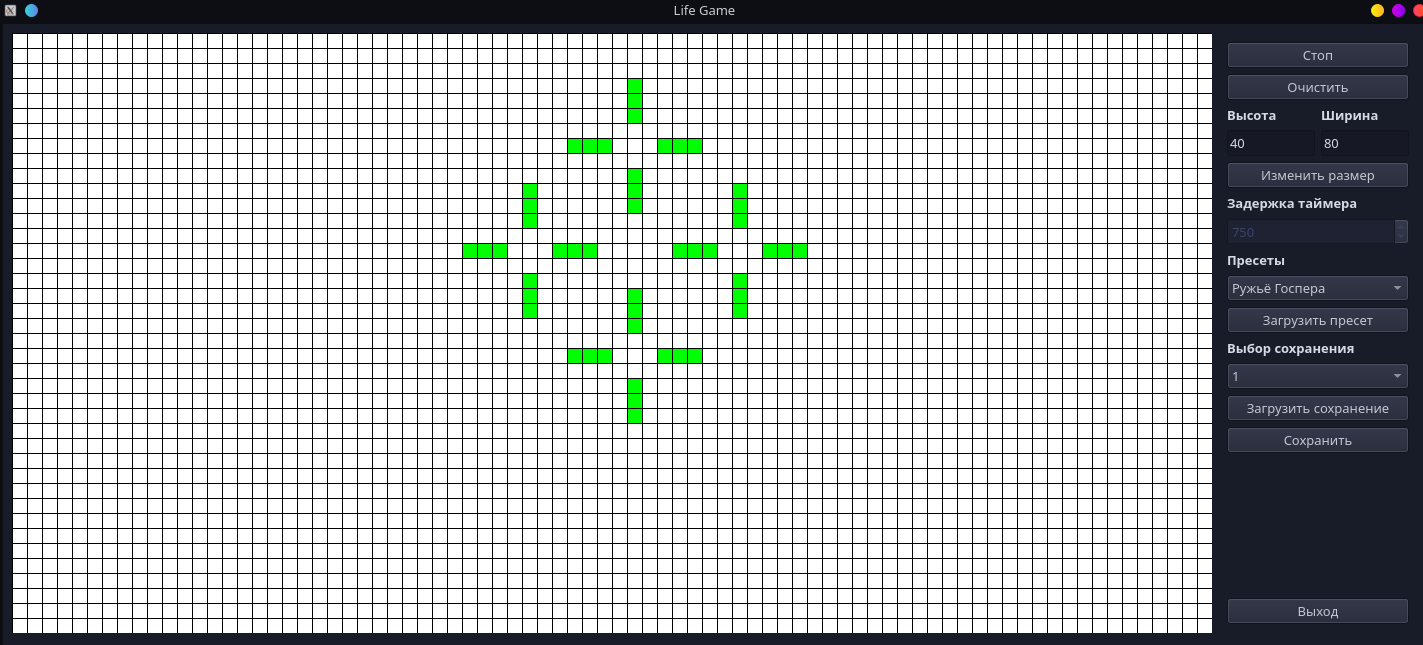


Рисунок 7.15 - Скриншот процесса работы программы

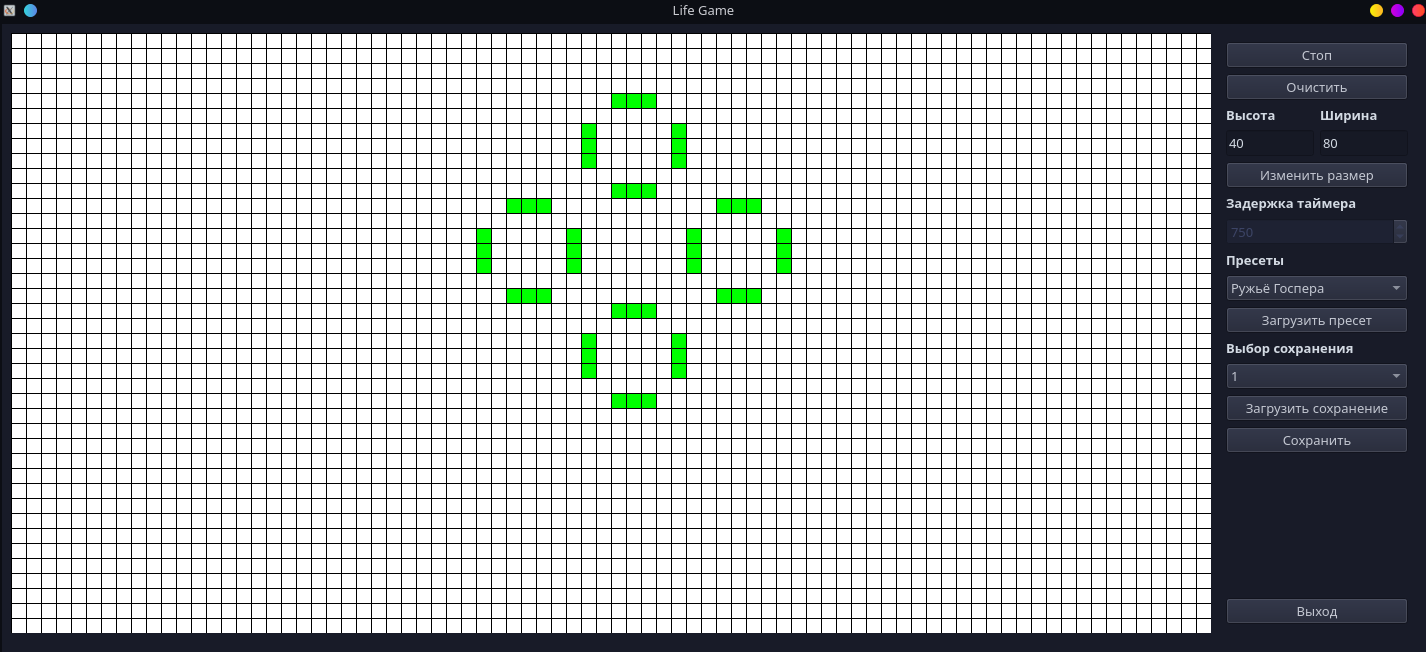


Рисунок 7.16 – Конец третьего теста

При множественных запусках с одинаковыми фигурами результаты не изменяются и совпадают с ожидаемыми.

Судя по тестам, программа работает правильно, все поставленные задачи выполнены.

# Заключение

В ходе выполнения курсовой работы по дисциплине «Практикум по объектно-ориентированному программированию» была реализована игра «Жизнь».

В процессе реализации были выполнены следующие задачи:

* проведён анализ задачи;
* составлен алгоритм программы;
* выполнен выбор стека технологий;
* проведена реализация программы;
* создан пользовательский интерфейс;
* проведено тестирование.

В процессе выполнения были изучены возможности языка программирования C++, [кроссплатформенная](https://ru.wikipedia.org/wiki/%D0%9A%D1%80%D0%BE%D1%81%D1%81%D0%BF%D0%BB%D0%B0%D1%82%D1%84%D0%BE%D1%80%D0%BC%D0%B5%D0%BD%D0%BD%D0%BE%D1%81%D1%82%D1%8C) [среда разработки](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D0%B3%D1%80%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D1%80%D0%B5%D0%B4%D0%B0_%D1%80%D0%B0%D0%B7%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B8) QtCreator, а также требования к оформлению курсовой работы

# Список использованных источников

1. Архив номеров научно-популярного физико-математического журнал «Квант» [Электронный ресурс] – Режим доступа: <http://kvant.mccme.ru/1974/09/igra_zhizn.htm> (дата обращения 10.03.2021)
2. Vuzlit архив студенческих работ [Электронный ресурс] – Режим доступа: [https://vuzlit.ru/948592/preimuschestva\_nedostatki\_yazyka\_programmirovaniya](https://vuzlit.ru/948592/preimuschestva_nedostatki_yazyka_programmirovaniya%20) (дата обращения 10.05.2021)
3. Моя библиотека электронная библиотека книг и статей [Электронный ресурс] – Режим доступа: <https://mybiblioteka.su/8-100076.html> (дата обращения 10.05.2021)
4. QT Documentation [Электронный ресурс] – Режим доступа: <https://doc.qt.io/qt-5.15/> (дата обращения 13.04.2021)
5. Cplusplus.com [Электронный ресурс] – Режим доступа: <http://cplusplus.com/reference/> (дата обращения 10.04.2021)
6. Стивен Прата. Язык программирования C++ (C++11). Лекции и упражнения, 6-е издание — М.: Вильямс, 2012. — 1248 с.
7. Qt 5.10. Профессиональное программирование на C++. — СПб.: БХВ-Петербург, 2018. — 1072 с.