Федеральное государственное автономное образовательное учреждение
высшего профессионального образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт математики

Базовая кафедра вычислительных и информационных технологий

УТВЕРЖДАЮ
Заведующий кафедрой
_____ В.В. Шайдуров
подпись
«____» _____ 2012 г.

# ОТЧЕТ ОБ УЧЕБНОЙ ПРАКТИКЕ

Институт математики СФУ
место прохождения практики

Модель колонии «Жизнь»
тема

Руководитель _____ доцент, канд. физ-мат. наук И.В. Баранова
подпись, дата

Студент _____ _____ _____ М.С. Снетков
номер группы номер зачетной книжки подпись, дата

**Красноярск 2012**

# Содержание

## 1. Постановка задачи

Целью курсовой работы является создание программы на языке объектно-ориентированного программирования C++, моделирующую жизнь колонии живых клеток. Жизнь − многоклеточное сообщество, населяющее пустыню. Под пустыней понимается квадратная решетка, в каждую ячейку которой вмещается одна клетка Жизни. Мерой течения времени служит смена поколений Жизни, приносящая в колонию клеток смерть и рождение. Если у клетки меньше двух соседей (из восьми возможных), она погибает от одиночества. Если количество соседей больше трех, клетка погибает от тесноты. Если рядом с пустой ячейкой оказывается ровно три клетки Жизни, то в этой ячейке рождается новая клетка Жизни. Исходными данными для программы служит начальное расположение клеток. В качестве результата нужно получить вид последовательности поколений колонии.

# 2. Описание программы

## 2.1 Среда разработки программы

Программа реализована в среде разработки Microsoft Visual Studio 2010 на языке объектно-ориентированного программирования C++ с применением библиотеки Microsoft Foundation Classes (MFC)

Пакет Microsoft Foundation Classes (MFC) — библиотека на языке C++, разработанная Microsoft и призванная облегчить разработку GUI-приложений для Microsoft Windows путем использования богатого набора библиотечных классов.

Библиотека MFC облегчает работу с GUI путем создания каркаса приложения — «скелетной» программы, автоматически создаваемой по заданному макету интерфейса и полностью берущей на себя рутинные действия по его обслуживанию (отработка оконных событий, пересылка данных между внутренними буферами элементов и переменными программы и т. п.). Программисту после генерации каркаса приложения необходимо только вписать код в места, где требуются специальные действия. Каркас должен иметь вполне определенную структуру, поэтому для его генерации и изменения в Visual C++ предусмотрены мастера.

Кроме того, MFC предоставляет объектно-ориентированный слой оберток (англ. wrappers) над множеством функций Windows API, делающий несколько более удобной работу с ними. Этот слой представляет множество встроенных в систему объектов (окна, виджеты, файлы и т. п.) в виде классов и опять же берет на себя рутинные действия вроде закрытия дескрипторов и выделения/освобождения памяти.

Добавление кода приложения к каркасу реализовано двумя способами. Первый использует механизм наследования: основные программные структуры каркаса представлены в виде классов, наследуемых от библиотечных. В этих классах предусмотрено множество виртуальных функций, вызываемых в определенные моменты работы программы. Путем

доопределения (в большинстве случаев необходимо вызвать функцию базового класса) этих функций программист может добавлять выполнение в эти моменты своего кода.

Второй способ используется для добавления обработчиков оконных событий. Мастер создает внутри каркасов классов, связанных с окнами, специальные массивы — карты (оконных) сообщений (англ. message map), содержащие пары «ИД сообщения — указатель на обработчик». При добавлении/удалении обработчика мастер вносит изменения в соответствующую карту сообщений.

## 2.2 Алгоритм решения задачи

### 2.2.1 Блок-схема алгоритма

На рис.1. приведена графическая блок-схема алгоритма смены поколения живых клеток. На рис. 2 приведена графическая блок-схема алгоритма работы программы.

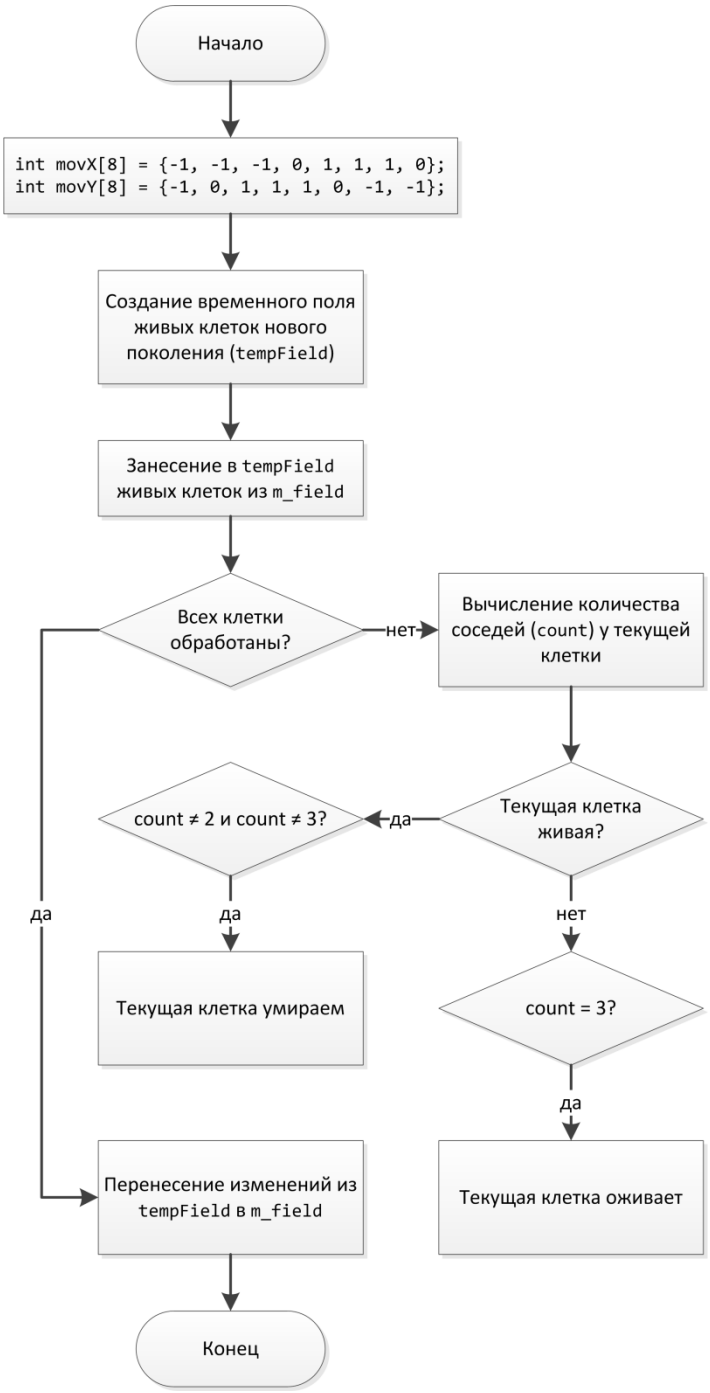


Рисунок 1 – Графическая блок-схема алгоритма смены поколения клеток

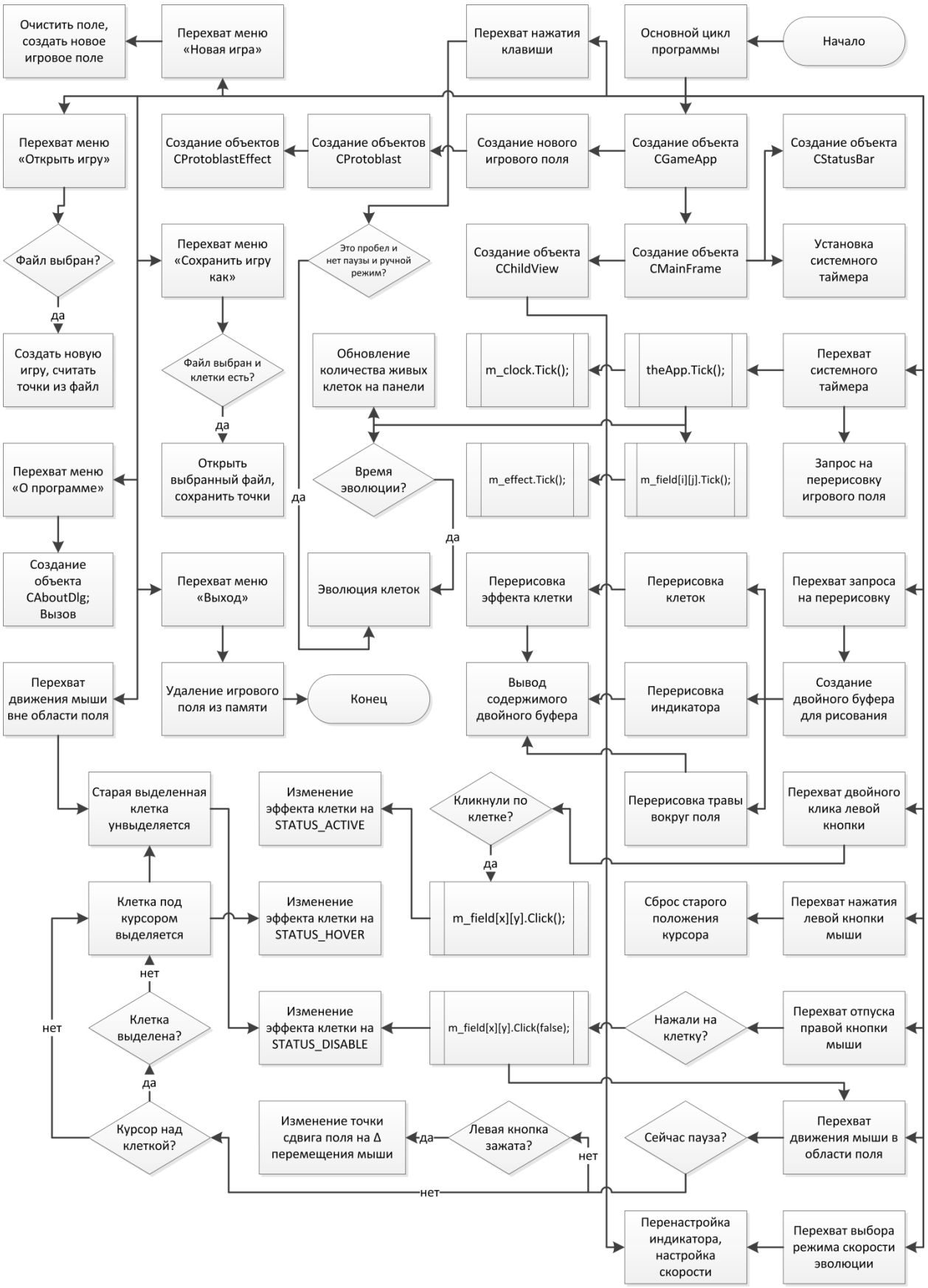


Рисунок 2 – Графическая блок-схема алгоритма работы программы

### 2.2.2 Описание алгоритма решения задачи

При запуске программы появляется окно с полем колонии «Жизнь». Изначально поле пустое, потому что ни одной клетки на поле нет. С помощью мыши пользователь может «оживить» или «убить» любую клетку, наведя на неё и кликнув дважды левой кнопкой мыши или нажав праву кнопку мыши соответственно. Для правильного понимания пользователем программы какая клетка на поле будет выбрана для действий, она подсвечивается желтым пульсирующим цветом. Белым цветом обозначены мертвые клетки, а синим – живые.

Для получения нового поколения клеток предусмотрены два режима работы программы, которые можно выбрать в главном меню программы:

1. Автоматическая смена поколений клеток через заданный временной промежуток (1 секунда, 3 секунды, 10 секунд, 42 минуты);
2. Ручной режим получения нового поколения при нажатии на кнопку Space (пробел).

Каждый режим получения нового поколения клеток отображен в правом нижнем углу игрового поля в виде часов с бегущей стрелкой и животного с часами соответственно.

Приведем последовательность действий алгоритма работы программы в автоматическом режиме смены поколений:

1. Основной цикл программы вызывает метод перехвата таймера «**`CMainFrame::OnTimer`**», который передаёт управление методу игрового класса «**`CGameApp::Tick`**», который отвечает за обновление игрового пространства.
2. Если с момента последнего получения нового поколения прошло установленное количество времени, то вызывается метод получения нового поколения клеток «**`CGameApp::Evolve`**».

3. В методе получения нового поколения создаётся массив нового поколения, куда записываются живые клетки текущего поля.
4. В цикле обходятся все клетки игрового поля, для каждой клетки считается количество её соседей. Если клетка живая, то если количество клеток меньше двух и больше трёх, то эта клетка умирает. Иначе если клетка была не живой, а количество соседей равно трём, то клетка оживает.
5. Все изменения временного поля переносятся на игровое поле.
6. Игровое поле выводится на экран.

Последовательность действий алгоритма работы программы в ручном режиме смены поколений:

1. При нажатии любой клавиши на клавиатуре основной цикл программы вызывает метод перехвата нажатия клавиши «**`CChildView::OnKeyDown`**».
2. Из метода перехвата нажатия клавиши управление передаётся методу игрового класса «**`CGameApp::KeyDown`**», который отвечает за нажатие клавиш.
3. Если нажатая клавиша является пробелом и игра не стоит на паузе, то вызывается метод «**`CGameApp::Evolve`**», в котором происходит получение нового поколения клеток.
4. Оставшаяся последовательность действий аналогична последовательности действий алгоритма в автоматическом режиме смены поколений начиная с пункта №3.

### 2.2.3 Описание основных функций программы

Ниже описываются методы класса игровой логики **`CGameApp`**, который обрабатывает все взаимодействия пользователя и основного цикла программы над игровым полем, а так же занимается выведением на экран игрового поля:

- Функция **`CGameApp()`** – конструктор.
    - Функция ничего не принимает.

  В данном методе происходит первоначальная инициализация класса при создании объекта данного класса. Так же в конструкторе создаётся новая игра.

- Функция **`~CGameApp()`** – деструктор.
    - Функция ничего не принимает.

  Игровое поле удаляется из памяти.

- Функция **`void OnEvolve(UINT id)`** – получение нового поколения.
    - **`UINT id`** – идентификатор нажатого пункта главного меню, отвечающего за режим получения нового поколения клеток на игровом поле;
    - Функция ничего не возвращает.

  В зависимости от идентификатора выбирается или ручной режим, или автоматический. В меню переставляется точка напротив выбранного пункта. Сбрасывается счётчик количества тиков таймера с момента последнего его сброса. Для индикатора режима получения поколения устанавливается соответствующий режим обновления.

- Функция **`void CreateNewGame()`** – создание новой игры.
    - Функция ничего не принимает;
    - Функция ничего не возвращает.

Если новая игра создаётся не в первый раз, то текущее игровое поле удаляется из памяти. Создаётся новое игровое поле, которое заполняется не живыми клетками.

- Функция **`void FreeField()`** – удаление игрового поля.
  - Функция ничего не принимает;
  - Функция ничего не возвращает.

  Если есть выделенный в памяти двухмерный массив игрового поля, то он освобождается из памяти.

- Функция **`void LeftButtonDoubleClick(UINT nFlags, CPoint point)`** – двойной клик левой кнопки мыши.
  - **`UINT nFlags`** – значение нажатых служебных клавиш и кнопок мыши;
  - **`CPoint point`** – координаты курсора, в которых произошёл двойной клик;
  - Функция ничего не возвращает.

  Если клик произошёл по клетке игрового поля, то эта клетка «оживает».

- Функция **`void LeftButtonDown(UINT nFlags, CPoint point)`** – нажатие левой кнопки.
  - **`UINT nFlags`** – значение нажатых служебных клавиш и кнопок мыши;
  - **`CPoint point`** – координаты курсора, в которых произошёл двойной клик;
  - Функция ничего не возвращает.

  Координатам точки, в которой последний раз находился курсор при нажатой левой кнопке, присваивается значение, соответствующее значению, что кнопка зажата впервые.

- Функция **`void RightButtonUp(UINT nFlags, CPoint point)`** – отпускание правой кнопки мыши.
    - **`UINT nFlags`** – значение нажатых служебных клавиш и кнопок мыши;
    - **`CPoint point`** – координаты курсора, в которых произошёл двойной клик;
    - Функция ничего не возвращает.

  Если кнопку отпустили над клеткой поля, то эта клетка становится мертвой. Вызывается функция нахождения курсора над этой клеткой, чтобы клетка стала выделенной.

- Функция **`void NcMouseMove()`** – передвижение курсора вне игрового поля.
    - Функция ничего не принимает;
    - Функция ничего не возвращает.

  Если курсор вышел за пределы игрового поля, то выделенная на поле клетка перестанет выделяться.

- Функция **`void MouseMove(UINT nFlags, CPoint point)`** – передвижение курсора мыши в области игрового поля.
    - **`UINT nFlags`** – значение нажатых служебных клавиш и кнопок мыши;
    - **`CPoint point`** – координаты курсора, в которых произошёл двойной клик;
    - Функция ничего не возвращает.

  Если игра на паузе, то ничего не происходит. Иначе если зажата левая кнопка мыши, то вычисляется дельта передвижения и суммируется в переменную сдвига положения игрового поля. Если не нажата левая кнопка мыши, то если во время передвижения курсора перескочили с

одной клетку на другую, то надо перестаёт выделяться старая клетка и начинает – текущая.

- Функция **`void Evolve()`** – получение нового поколения.
    - Функция ничего не принимает;
    - Функция ничего не возвращает.

  В памяти создаётся массив живых клеток нового поколения. В него переносятся живые клетки текущего поколения. Проходя по всем клеткам текущего игрового поля, для каждой клетки вычисляется количество соседей. Исходя из условий получения нового поколения, некоторые клетки умирают, а некоторые – рождаются. После все изменения с временного игрового поля переносятся на текущее поле.
- Функция **`void Tick()`** – срабатывание таймера.
    - Функция ничего не принимает;
    - Функция ничего не возвращает.

  Если приложение на паузе, то ничего делать не надо. Иначе увеличивается счетчик срабатываний таймера. Для каждой клетки игрового поля вызывается метод срабатывания таймера. Индикатор получает вызов игрового поля тоже. Если пришло время получения нового поколения, и пользователь установил автоматический режим, то происходит получение нового поколения клеток.

- Функция **`void Render(CMemoryDC &dc, CRect screen)`** – перерисовка игрового поля.
    - **`CMemoryDC &dc`** – ссылка на контекст рисования в памяти;
    - **`CRect screen`** – текущая область рисования;
    - Функция ничего не возвращает.

  Определяется размер одной клетки. Для каждой игровой клетки вызывается метод рисования клетки с заданными координатами. Если индикатор помещается в текущую область рисования, то он получает

запрос на рисование в правом нижнем углу. Если поле клеток не полностью занимает игровое поле, то пустые области вокруг поля заполняются тайтлами (циклическая зеленая трава).

- Функция **`int GetProtoblastCount()`** – количество живых клеток на поле.
    - Функция ничего не принимает;
    - Функция возвращает **`int`** – количество живых клеток.

  Пробегая по всем клеткам игрового поля, в счетчик записывается количество живых клеток.

- Функция **`CSize GetFieldSize()`** – размеры поля с клетками.
    - Функция ничего не принимает;
    - Функция возвращает **`CSize`** – количество клеток по горизонтами и по вертикали.

  Значения количества клеток по вертикали и по горизонтами берутся из статических констант данного класса.

- Функция **`int GetFPS()`** – количество кадров в секунду.
    - Функция ничего не возвращает;
    - Функция возвращает **`int`** – количество кадров в секунду.

  Возвращаемое значение берется из статической константы данного класса.

- Функция **`bool MousePositionToXY(CPoint position, CPoint &cell)`** – преобразование координат мыши в координаты клетки на поле.
    - **`CPoint position`** – координаты курсора мыши;
    - **`CPoint &cell`** – ссылка получившиеся координаты клетки;
    - Возвращает **`bool`** – истина, если координаты принадлежат клетке.

Положение курсора мыши преобразуется с учетом сдвига игрового поля. Если хоть одна из преобразованных координат точки отрицательная, то курсор точно не над точкой. С помощью деления без остатка вычисляются координаты точки, над которой находится курсор. Если курсор вышел за область массива с точками, то возвращается ложь. Иначе возвращается истина, а в результат записываются координаты точки под курсором.

- Функция **`void OnAppAbout()`** – перехват нажатия «О программе».
    - Функция ничего не принимает;
    - Функция ничего не возвращает.

  Программа ставится на паузу. Создаётся диалог о программе. При закрытии диалога приложение снимается с паузы.

- Функция **`void OnOpenGame()`** – перехват нажатия пункта «Загрузить игру».
    - Функция ничего не принимает;
    - Функция ничего не возвращает.

  Программа ставится на паузу. Создаётся диалог открытия файла заданного формата. Если файл выбран, то он открывается для чтения. Игровое поле очищается. Из файла загружаются координаты точек. Появляется уведомление о завершении загрузки игры. Включается ручной режим получения нового поколения. Приложение снимается с паузы.

- Функция **`void OnSaveGame()`** – нажатие на пункт «Сохранить игру как».
    - Функция ничего не принимает;
    - Функция ничего не возвращает.

Приложение ставится на паузу. Если на поле есть живые клетки, то создается диалог сохранения файла заданного формата. Если выбран файл, куда стоит сохранить расположение клеток игрового поля, то этот файл открывается для записи. В файл записывается количество живых клеток и их координаты. Файл закрывается, создаётся диалог об успешной операции. Приложение снимается с паузы.

- Функция **`void KeyDown(UINT nChar, UINT nRepCount, UINT nFlags)`** – нажатие на клавишу.
    - **`UINT nChar`** – код нажатой клавиши;
    - **`UINT nRepCount`** – количество последовательных нажатий;
    - **`UINT nFlags`** – значение нажатых служебных клавиш;
    - Функция ничего не возвращает.

  Если нажали на пробел и игра не на паузе, то вызывается функция получения нового поколения.

- Функция **`void OnNewGame()`** – нажатие на пункт «Новая игра».
    - Функция ничего не получаем;
    - Функция ничего не возвращает.

  Если живых клеток нет, то ничего не происходит. Иначе после подтверждения действия вызывается функция создания новой игры.

## 3. Примеры результатов работы программы

Приведем тестовые примеры работы программы в различных её состояниях.

На рис. 1 показан первоначальный вид запущенного приложения. Изначально установлен режим автоматической смены поколений каждые 3 секунды. Как только стрелка дойдет до 12 часов, смена поколений произойдет.

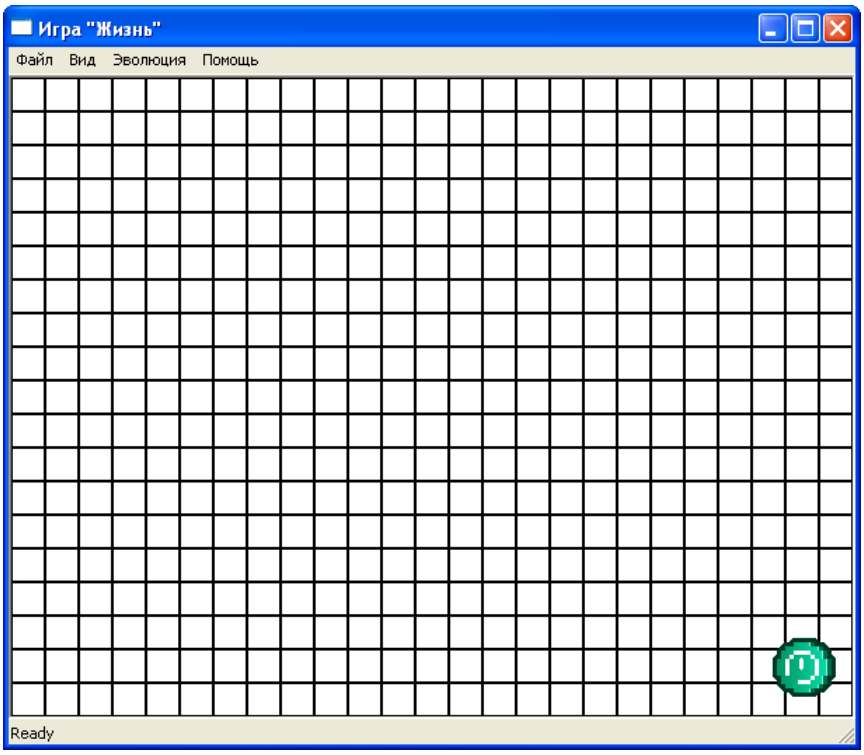


Рисунок 1 – Первоначальный вид запущенного приложения

При переходе в ручной режим циферблат сменится на изображение животного и стоящих часов, как указано на рис. 2. Начиная с этого момента и до переключения в автоматический режим, обновление поколений происходит при нажатии клавиши Space (пробел).

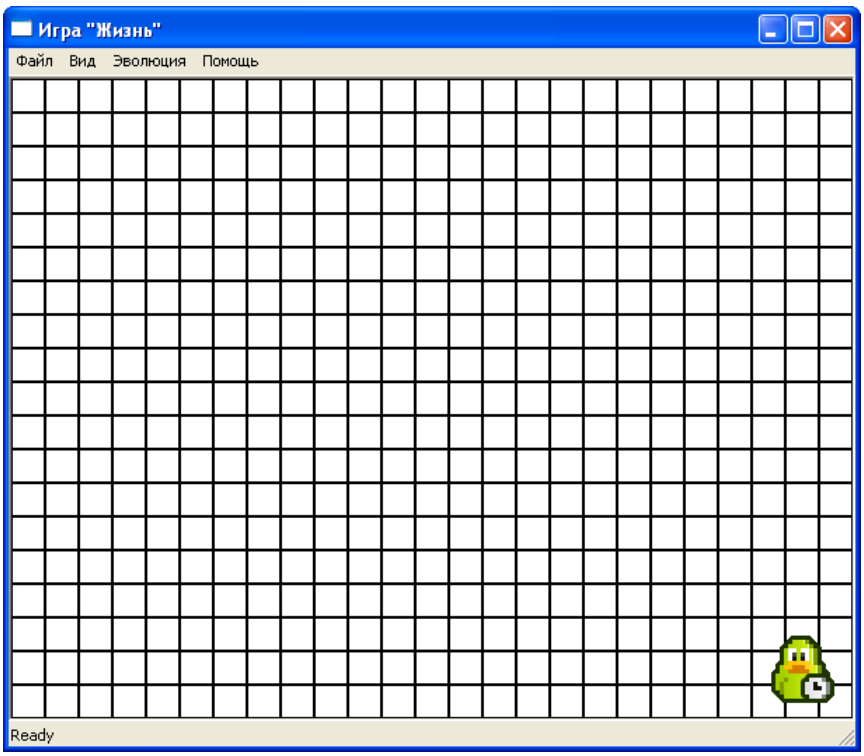

Рисунок 2 – Переключение в режим ручного обновления поколения

В главном меню программы (рис. 3) есть возможность установить желаемый режим обновления поколений, причем текущий выбранный режим выделен точкой возле себя. При повторном выборе одного и того же режима счетчик сбросится и режим будет работать ровно столько, сколько им запланированно.

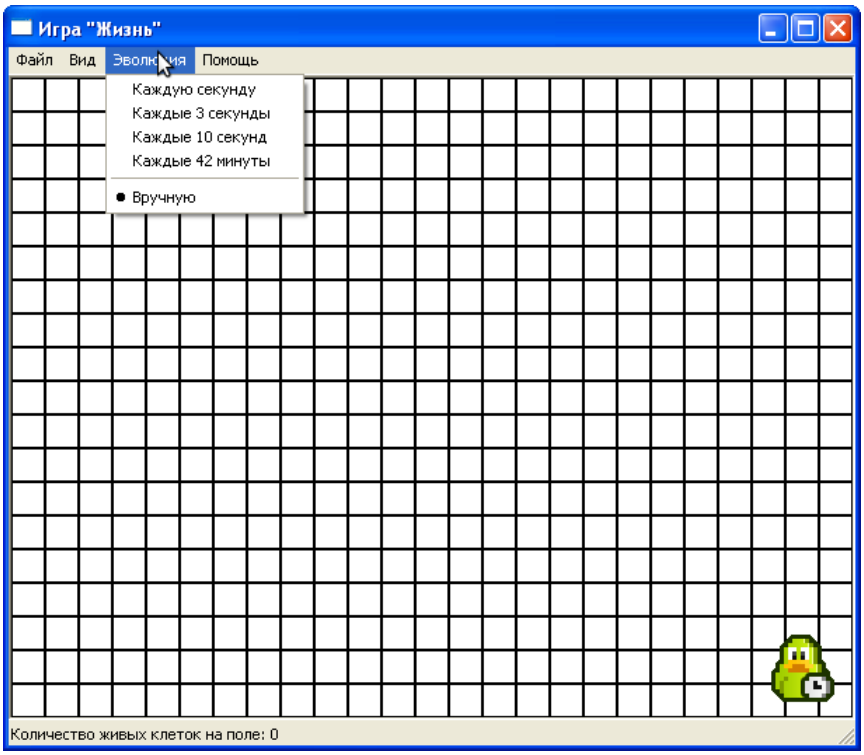

Рисунок 3 – Выбор режима обновления поколения

В пункте «Файл» (рис. 4) главного меню можно создать новую игру, сохранить игру, а так же сохранить. Если при создании новой игры на поле есть живые клетки, то изначально появится диалог с предупреждением (рис. 5). При положительном ответе поле будет очищено. При нажатии на пункты «Открыть игру» и «Сохранить игру как…» будут выведены диалоги открытия (рис. 6) и сохранения (рис. 7) соответственно. Причем при сохранении пустого поля будет выведено сообщение, что сохранять нечего.

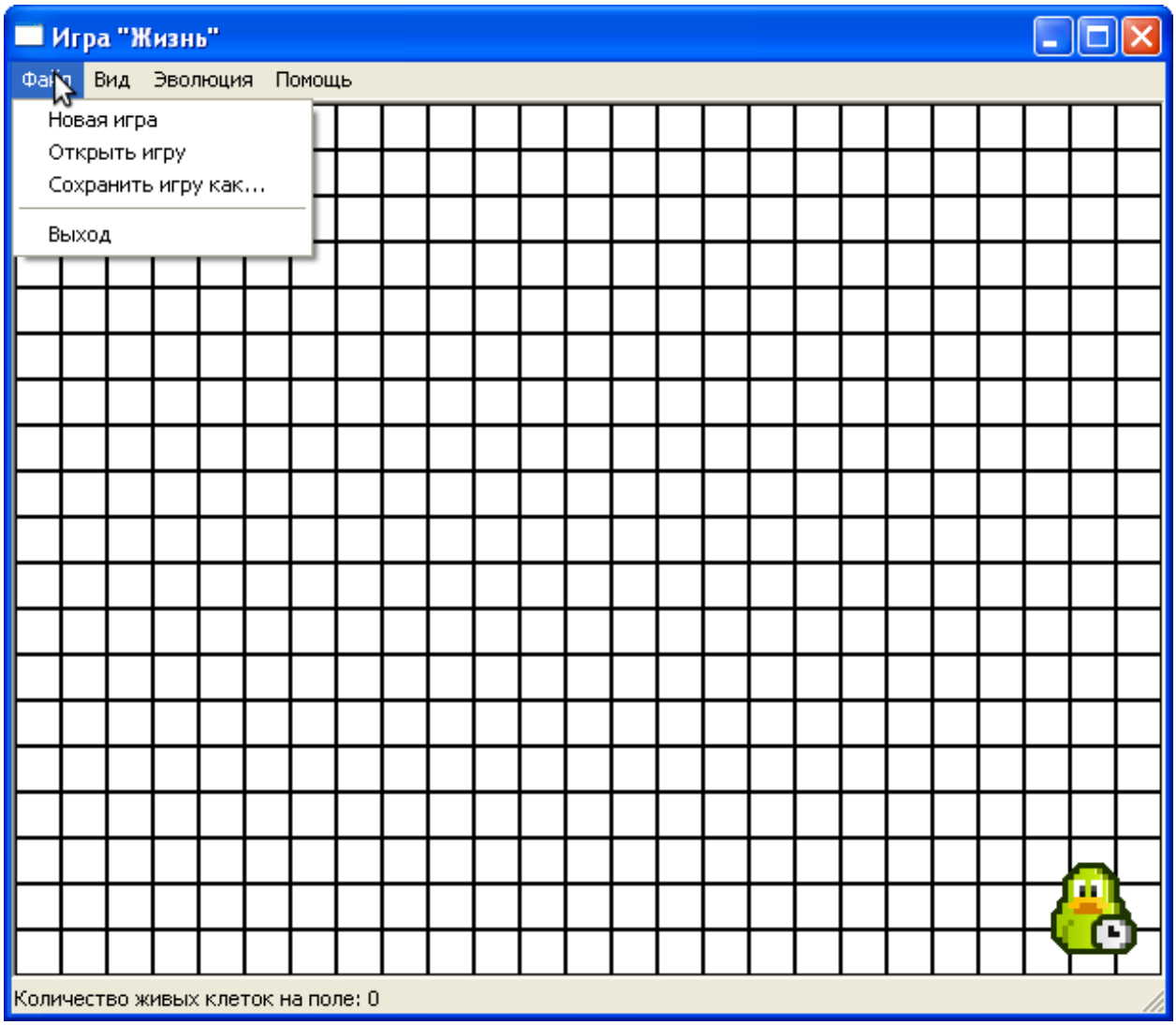


Рисунок 4 – Пункты меню «Файл»

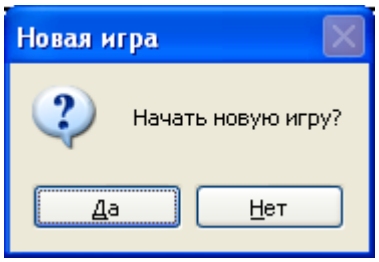


Рисунок 5 – Диалог создания новой игры, если на поле есть клетки

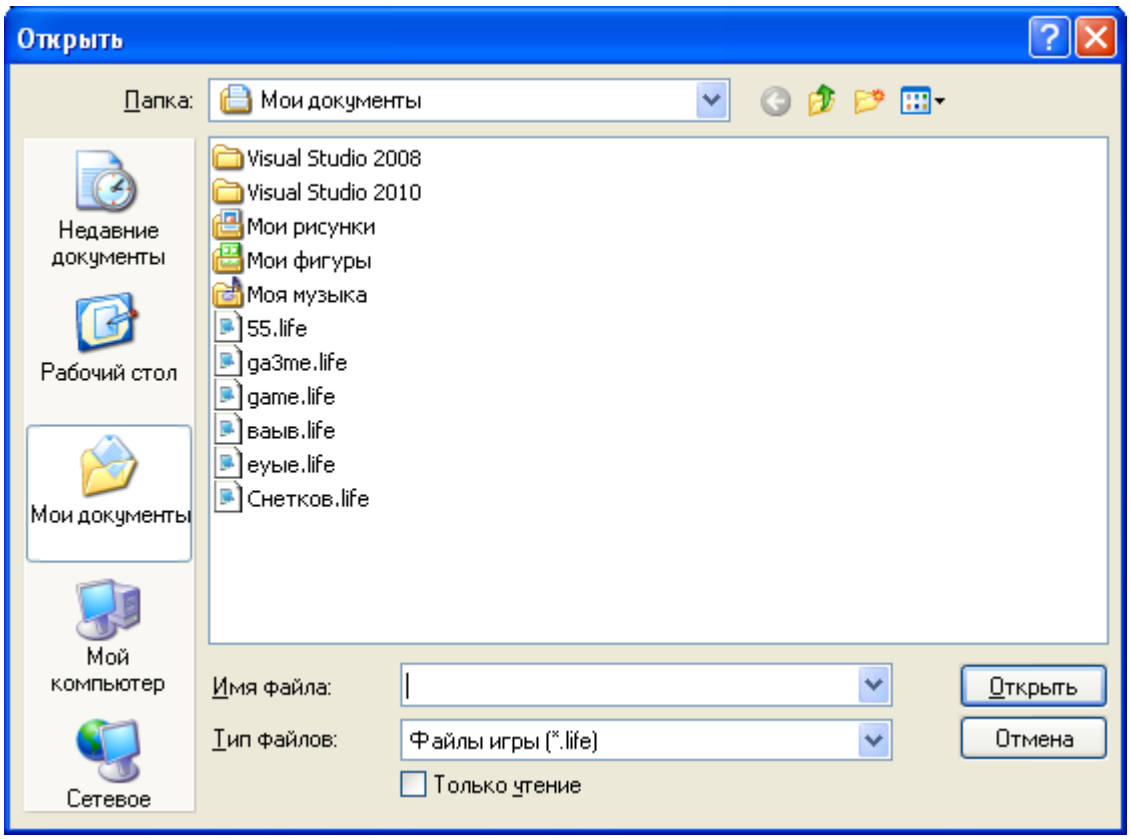


Рисунок 6 – Диалог открытия сохранённой игры с заданным расширением

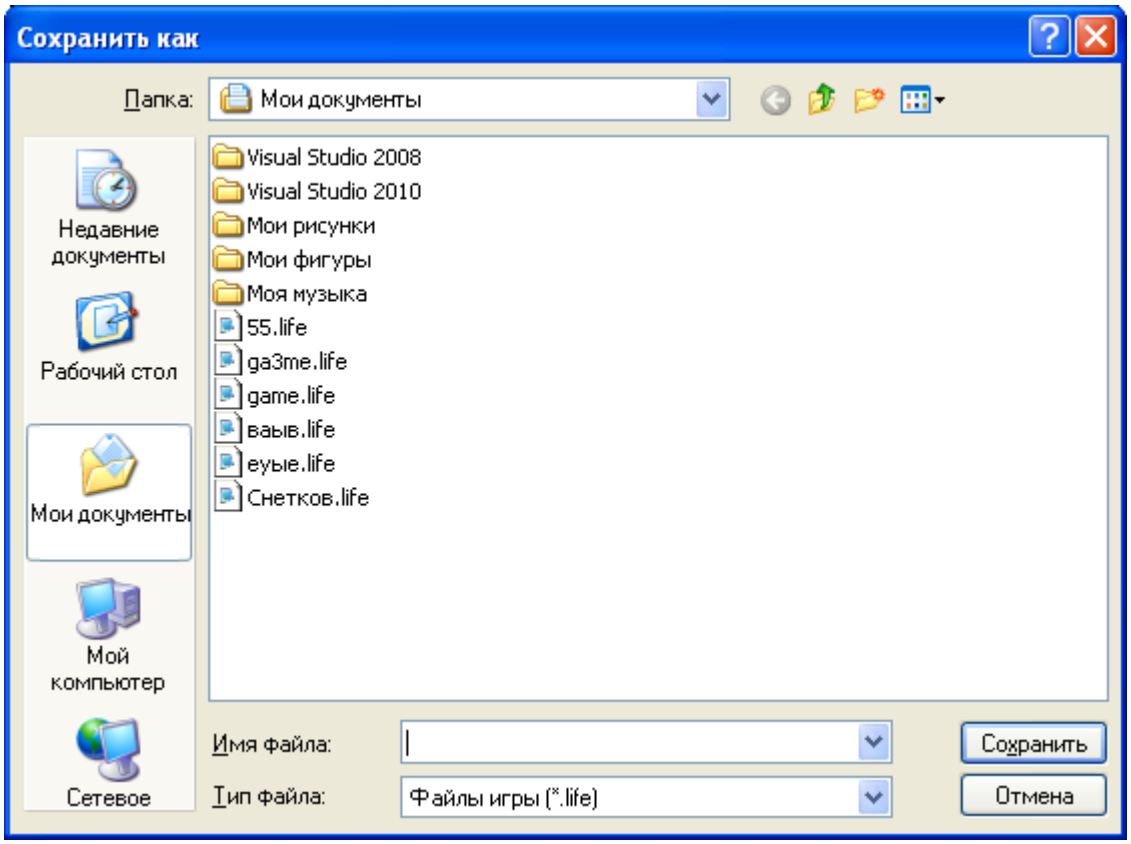


Рисунок 7 – Диалог сохранения текущей игры

В пункте меню «Вид» главного меню (рис. 8) есть возможность отключить панель состояния, которая располагается внизу и выводит количество живых клеток или текущее состояние приложения к дальнейшим действиям.

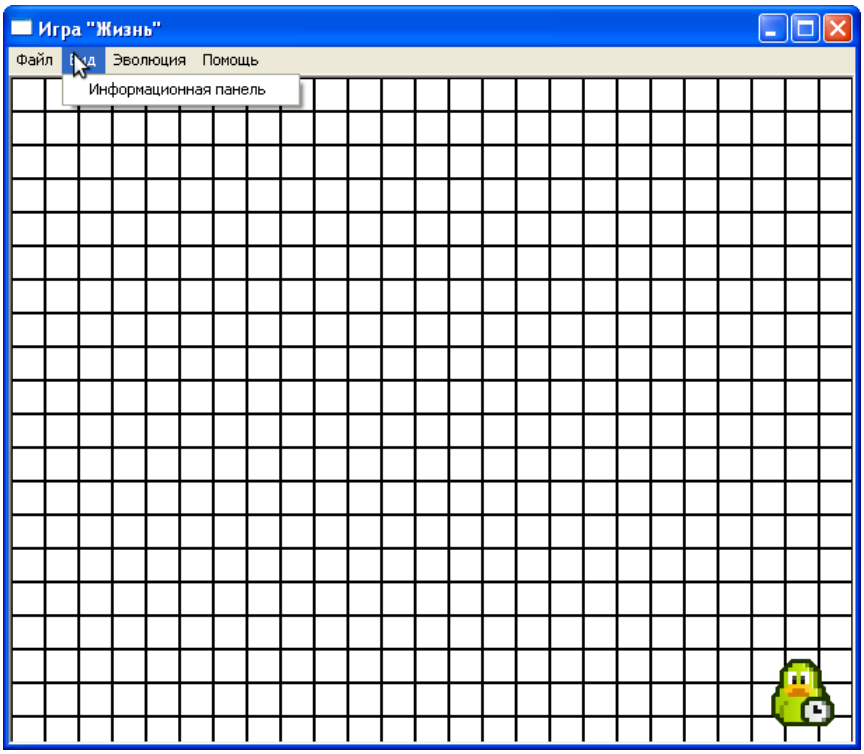


Рисунок 8 – Пункт меню «Вид» после выключения информационной панели

В пункте меню «Помощь» (рис. 9) можно посмотреть информацию в пункте «О программе» (рис. 10), где есть краткая информация об авторе приложения, а так же тонкий намек о причине создания приложения.

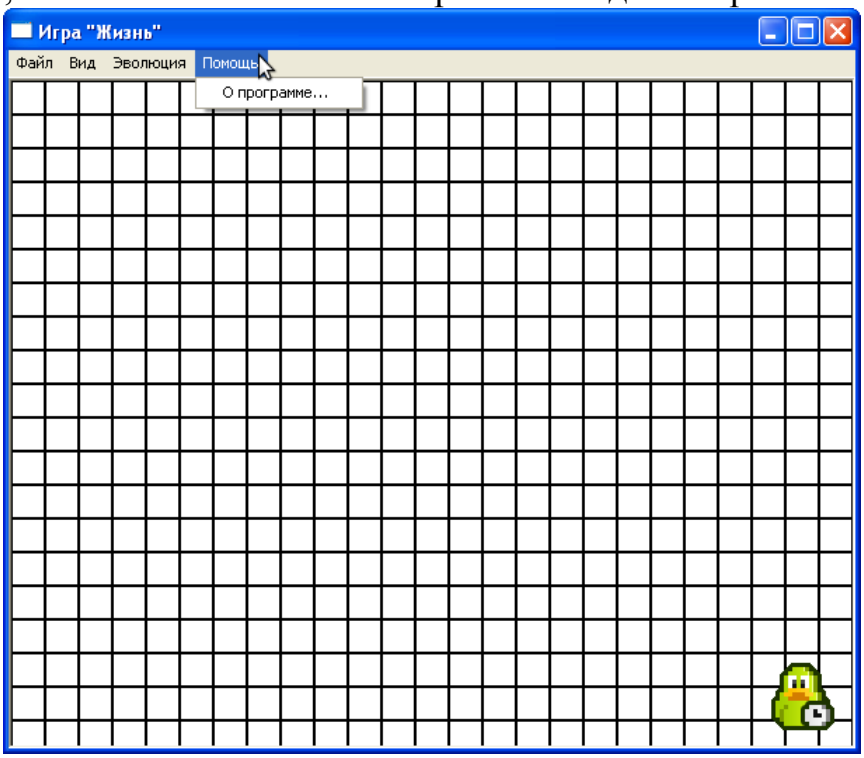


Рисунок 9 – Пункт меню «Помощь»

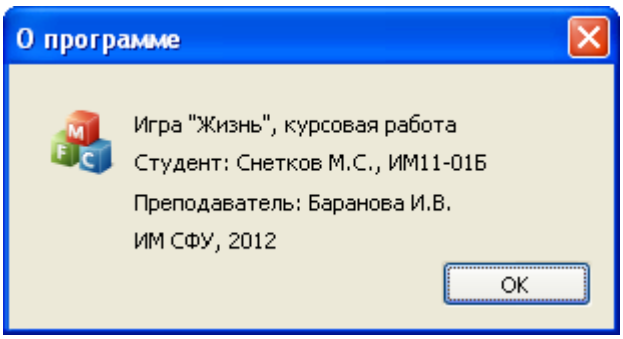

Рисунок 10 – Вызов диалога «О программе» из пункта «Помощь»

С помощью мыши можно расставлять и уничтожать игровые клетки на поле двойным щелчком левой кнопки и правым кликом правой кнопки соответственно. На рис. 11 указан один из возможных вариантов расстановки живых клеток. Через некоторое время в автоматическом режиме эти клетки сменятся новым поколением по законам колонии «Жизнь». Очень интересно наблюдать за сменой поколений, потому что в уме тяжело предсказать во что может превратиться та или иная колония. В большинстве случаев смена поколений происходит с волнообразным изменением количества живых клеток на поле. Для примера, над начальной колонией (рис. 12) была приезведена смена поколений несколько раз. (рис. 13, 14, 15, 16, 17).

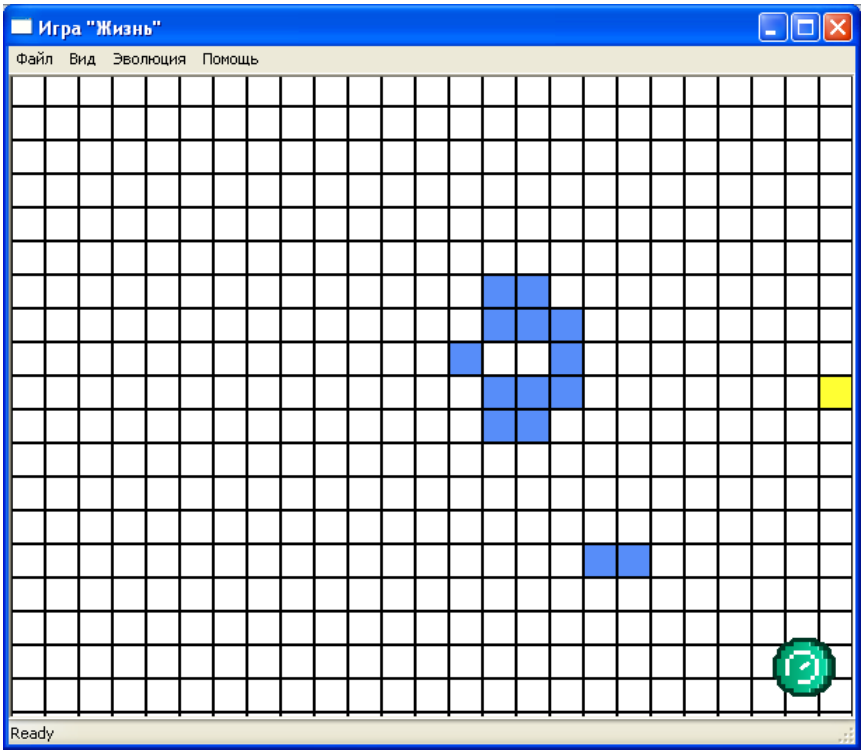

Рисунок 11 – Возможность расставлять и удалять клетки на игровом поле, клетка под курсором подсвечивается желтым пульсирующим цветом

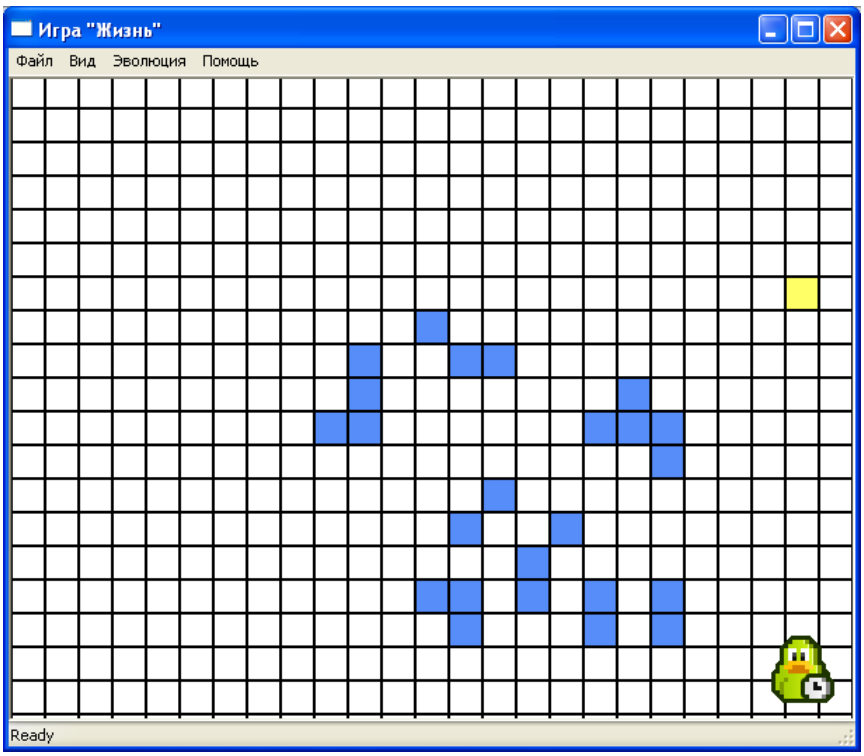


Рисунок 12 – Пример смены поколений (первоначальная колония)

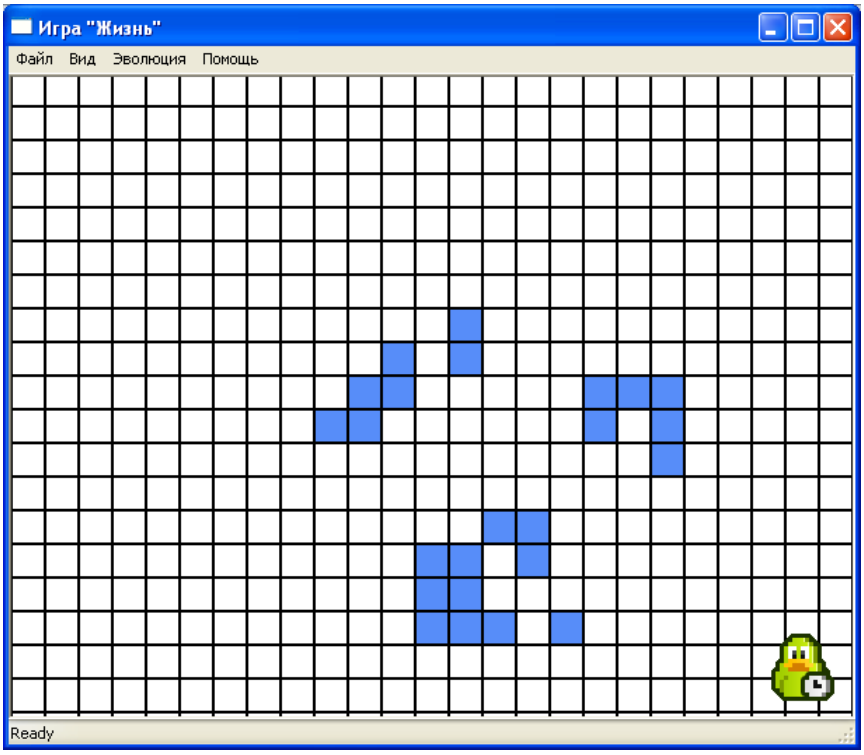


Рисунок 13 – Пример смены поколений (1-е поколение)

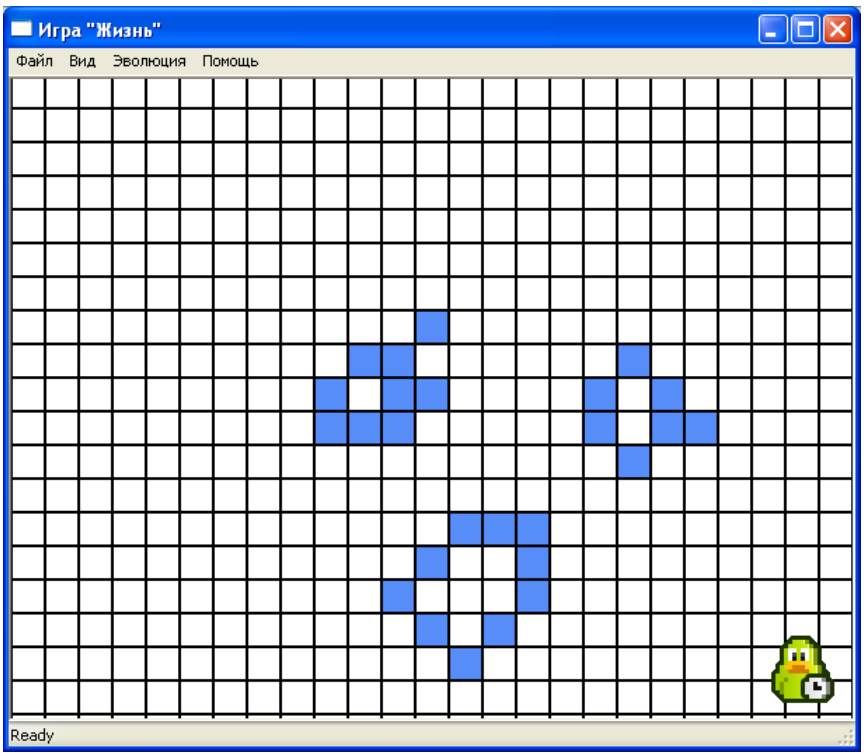


Рисунок 14 – Пример смены поколений (2-е поколение)

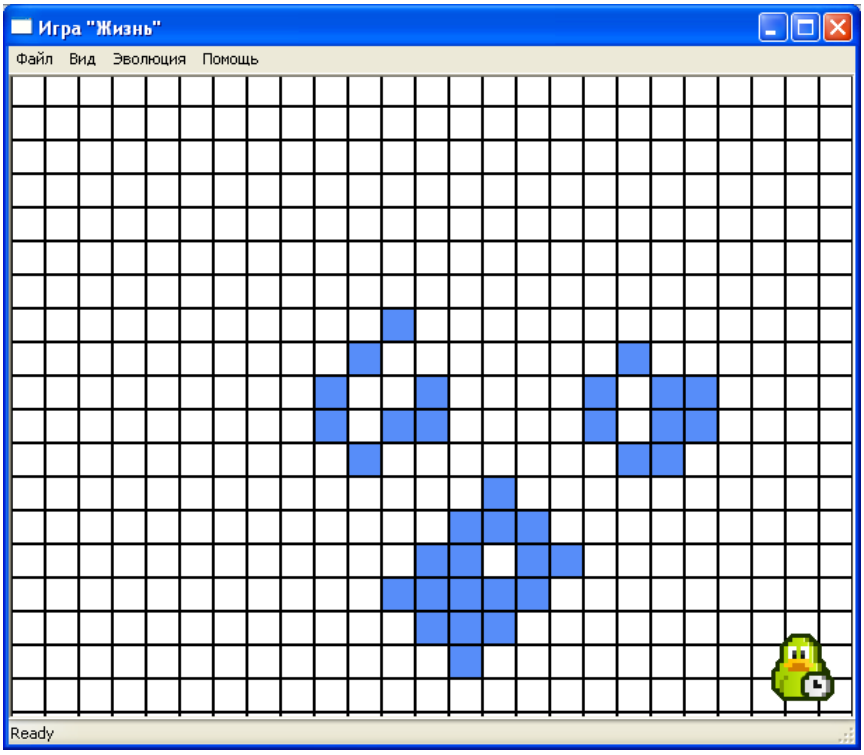


Рисунок 15 – Пример смены поколений (3-е поколение)

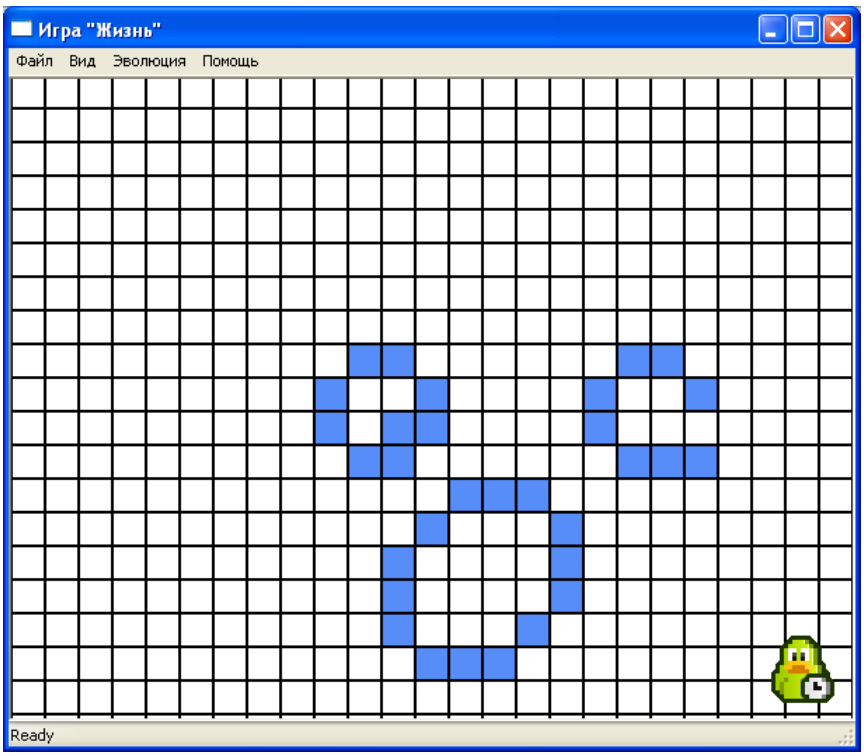


Рисунок 16 – Пример смены поколений (4-е поколение)

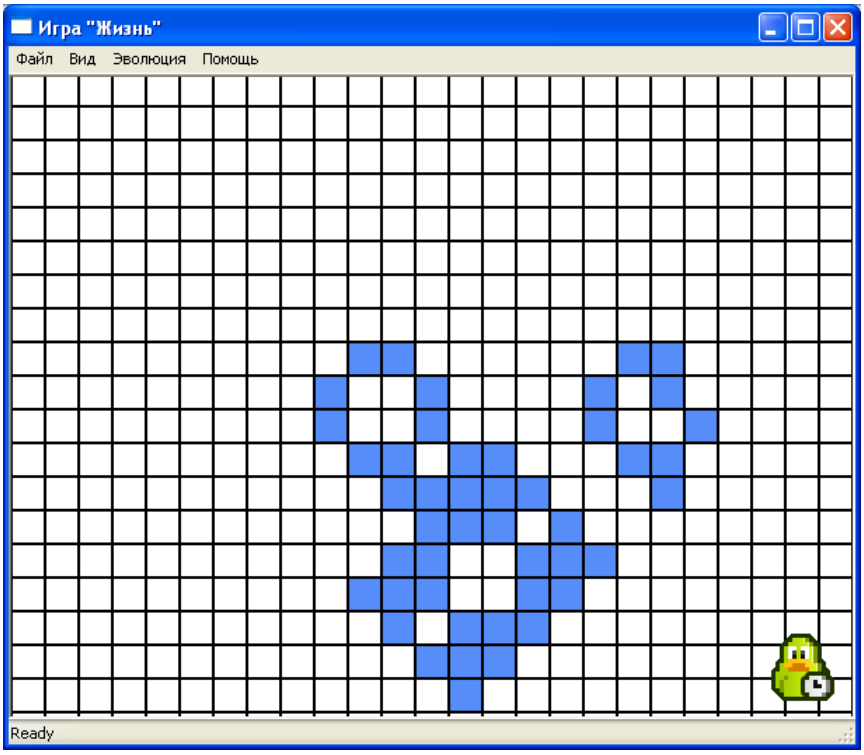


Рисунок 17 – Пример смены поколений (5-е поколение)

В приложении есть возможность навигации по полю с помощью зажатой левой кнопки мыши и передвижением курсора. Пустые области, где отсутствуют клетки колонии, заполняются тайтлами (рис. 18). Заполнение происходит рациональным образом только пустой области. Таким образом можно сколько угодно передвигаться по полю и наблюдать просторы зелени.

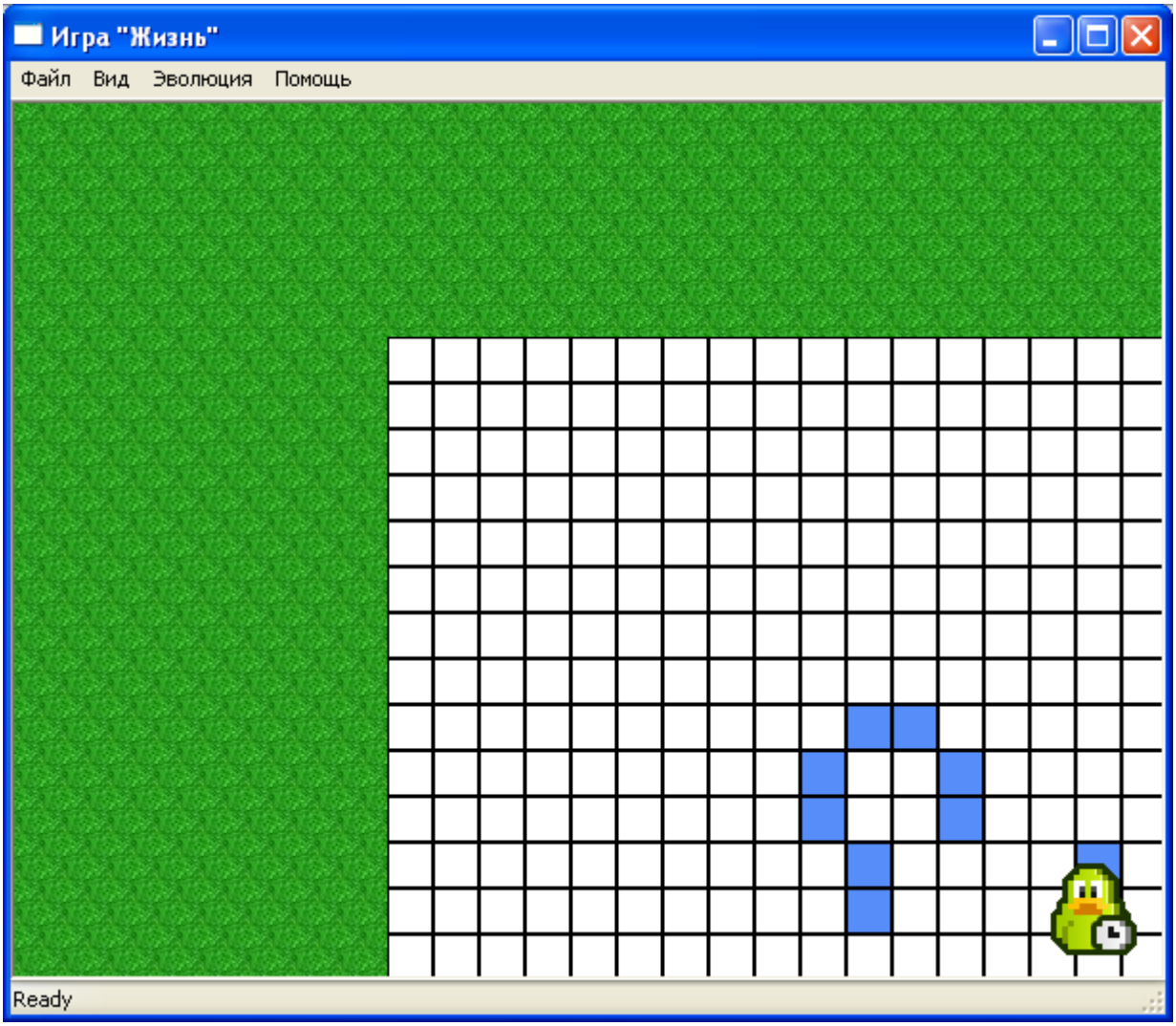


Рисунок 18 – Область вокруг поля с клетками заполняется тайтлами

# Список использованных источников

1. Баранов С.Н. Программирование на языке С++: учеб. пособие / С.Н. Баранов, И.В. Баранова. – Красноярск: ИПК СФУ, 2010. – 112 с.

2. Глушаков, С. В. Язык программирования С++ / С. В. Глушаков, А. В. Коваль, С. В. Смирнов. – М.: АСТ, 2004. – 500 с.

3. Лафоре Р. Объектно-ориентированное программирование в С++ / Р. Лафоре. – Спб.: Питер, 2004. – 923 с.

4. Павловская, Т. А. С/С++. Программирование на языке высокого уровня / Т. А. Павловская. – СПб.: Питер, 2010. – 461 с.

5. Прата, С. Язык программирования С++. Лекции и упражнения / С. Прата. – М.: Вильямс, 2006. – 1184 с.

6. MFC Desktop Application [Электронный ресурс]: база данных содержит описание классов библиотеки. — Электрон. дан. (4,5 тыс. записей). — Microsoft, [199—]. — Режим доступа: http://msdn.microsoft.com/en-us/library/d06h2x6e(v=vs.100).aspx. — Загл. с экрана.