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.

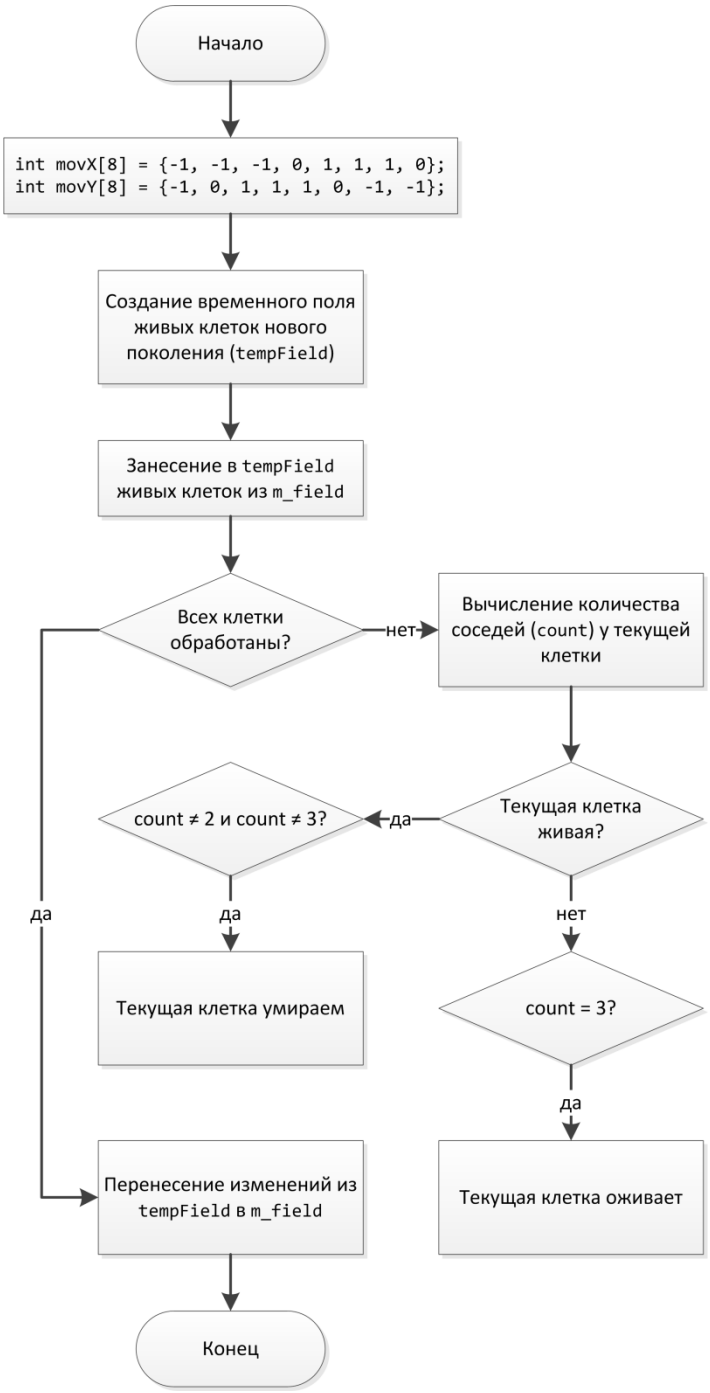

Рисунок 1 – Графическая блок-схема алгоритма смены поколения клеток

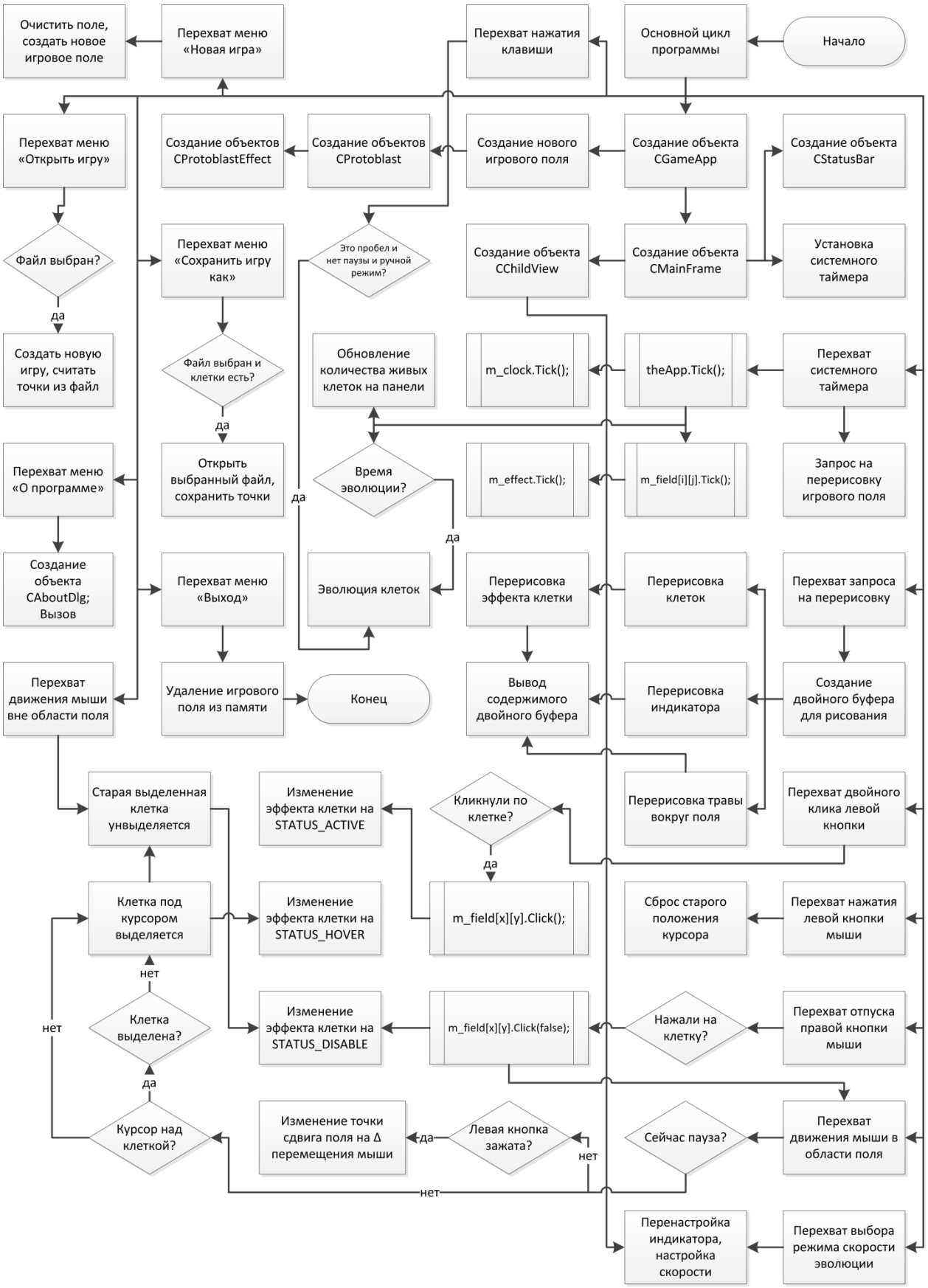

Рисунок 2 – Графическая блок-схема алгоритма работы программы

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.

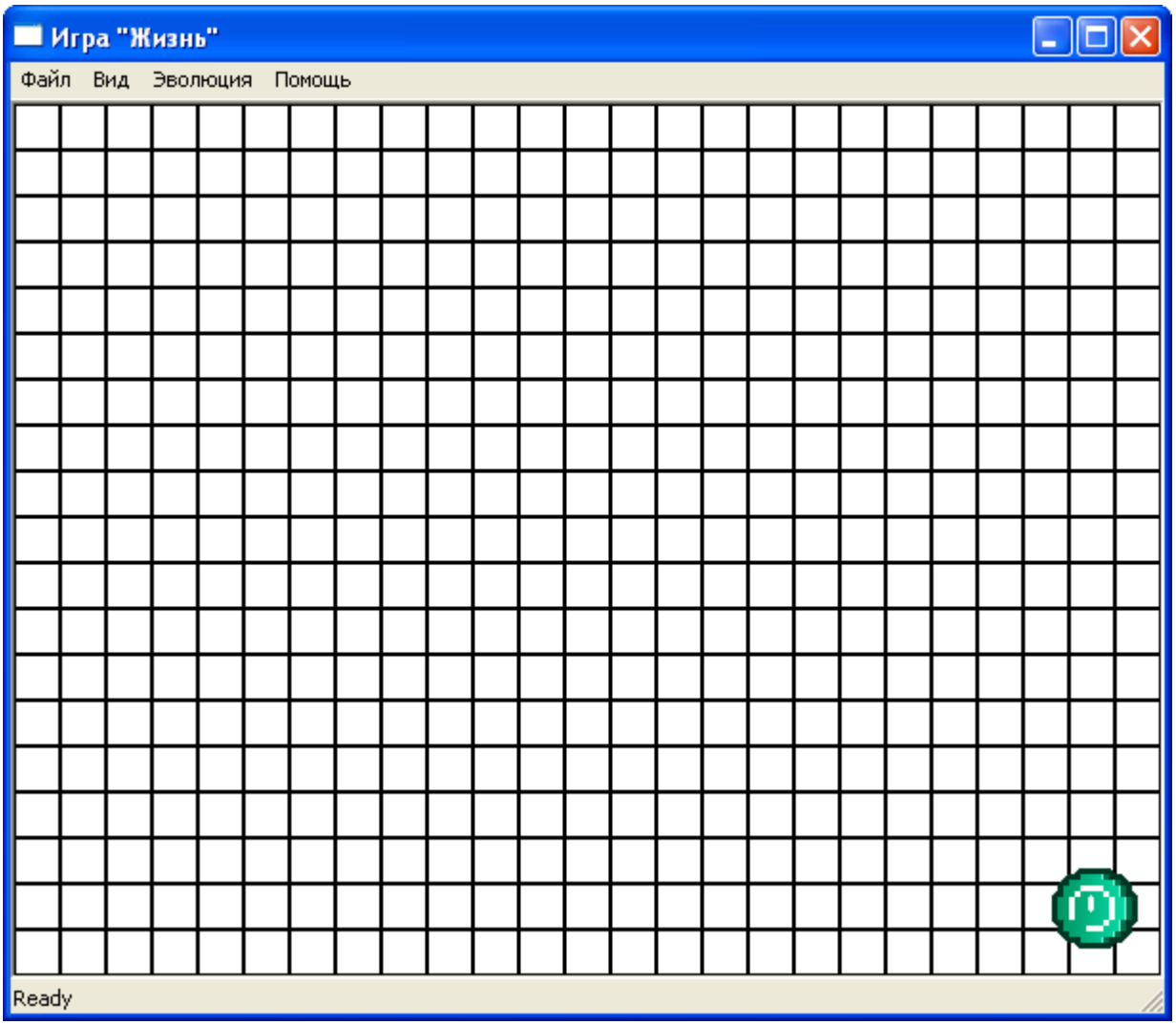

Рисунок 1 – Первоначальный вид запущенного приложения

При переходе в ручной режим циферблат сменится 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).

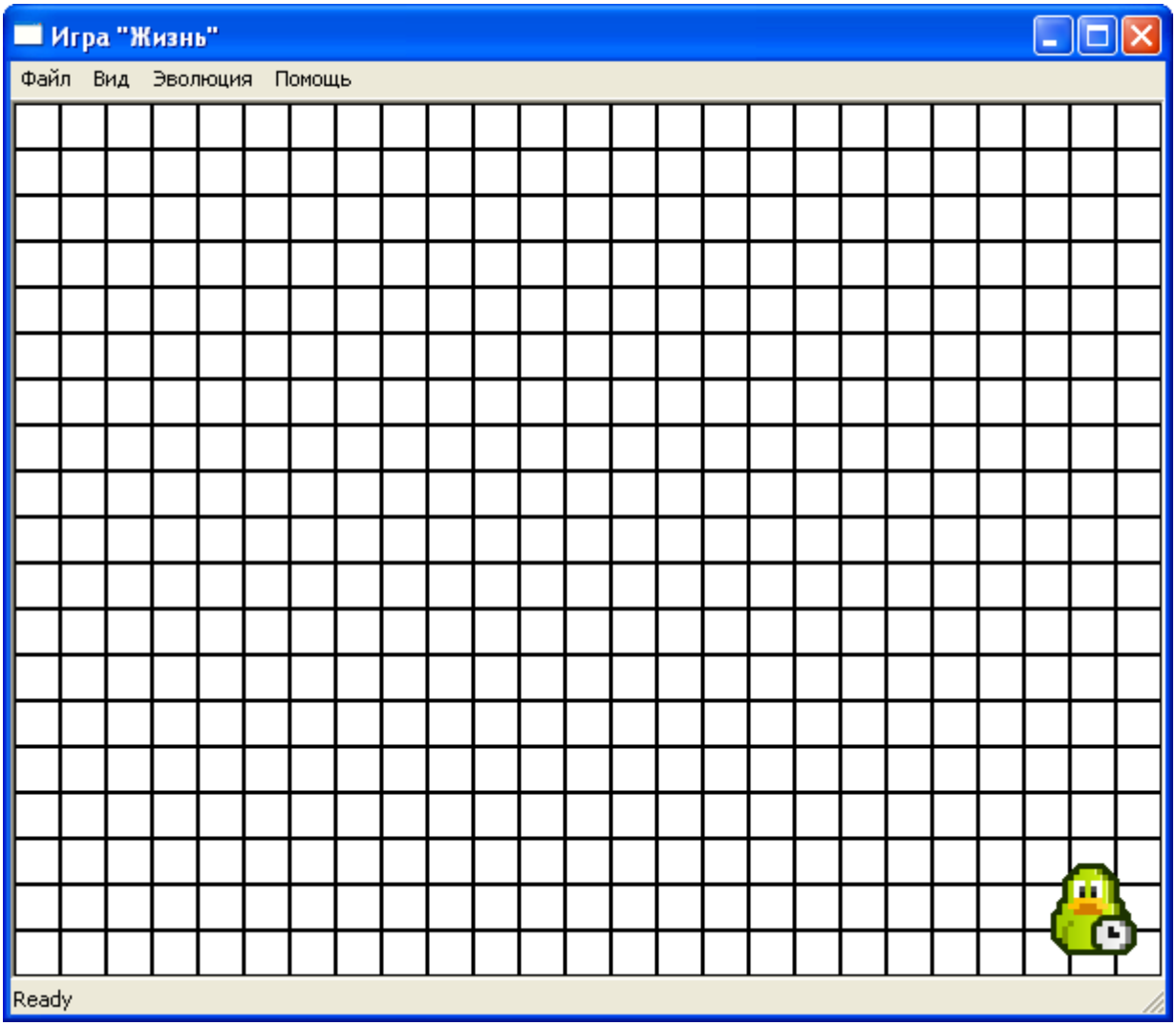

Рисунок 2 – Переключение в режим ручного обновления поколения

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.

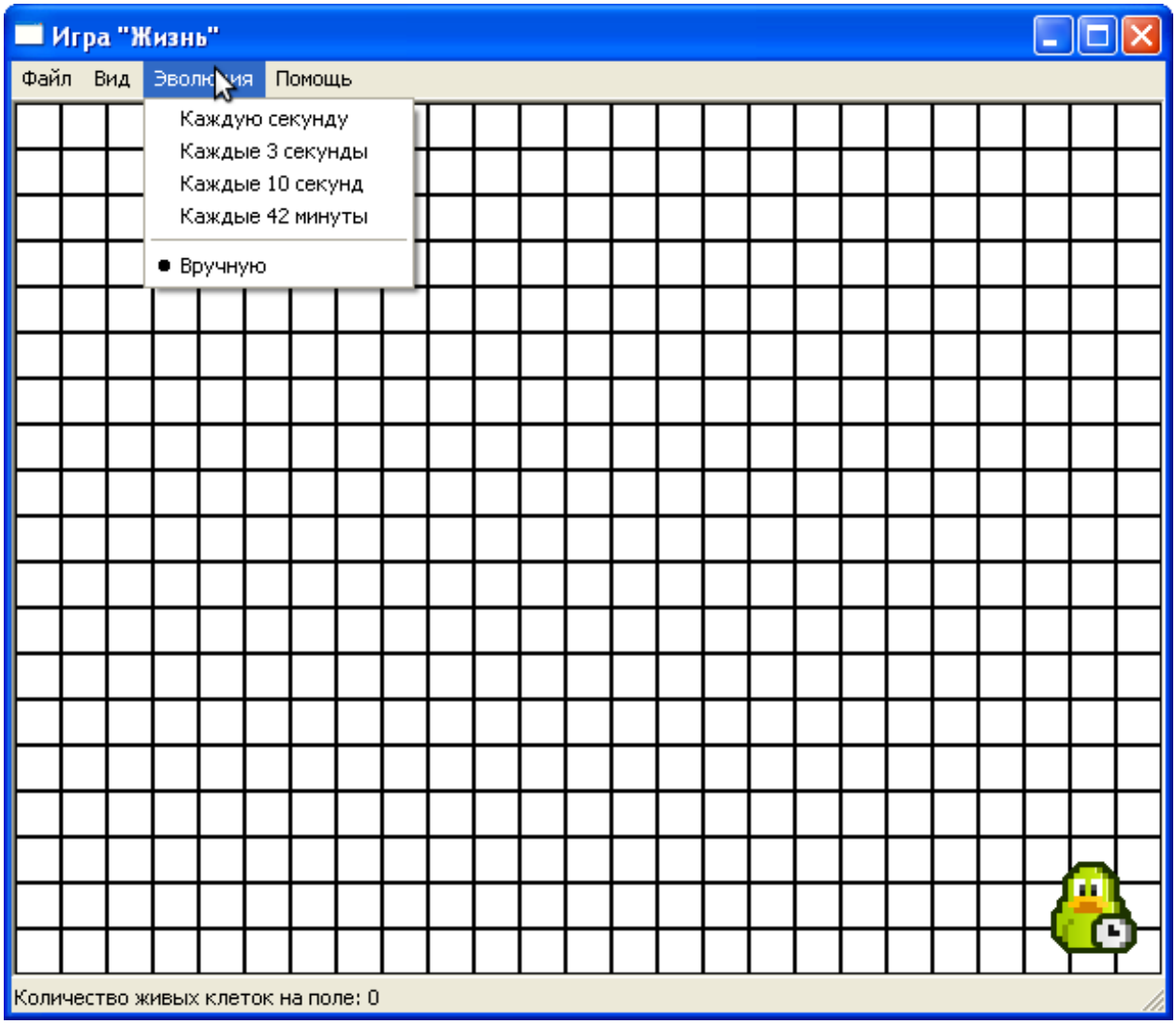

Рисунок 3 – Выбор режима обновления поколения

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.

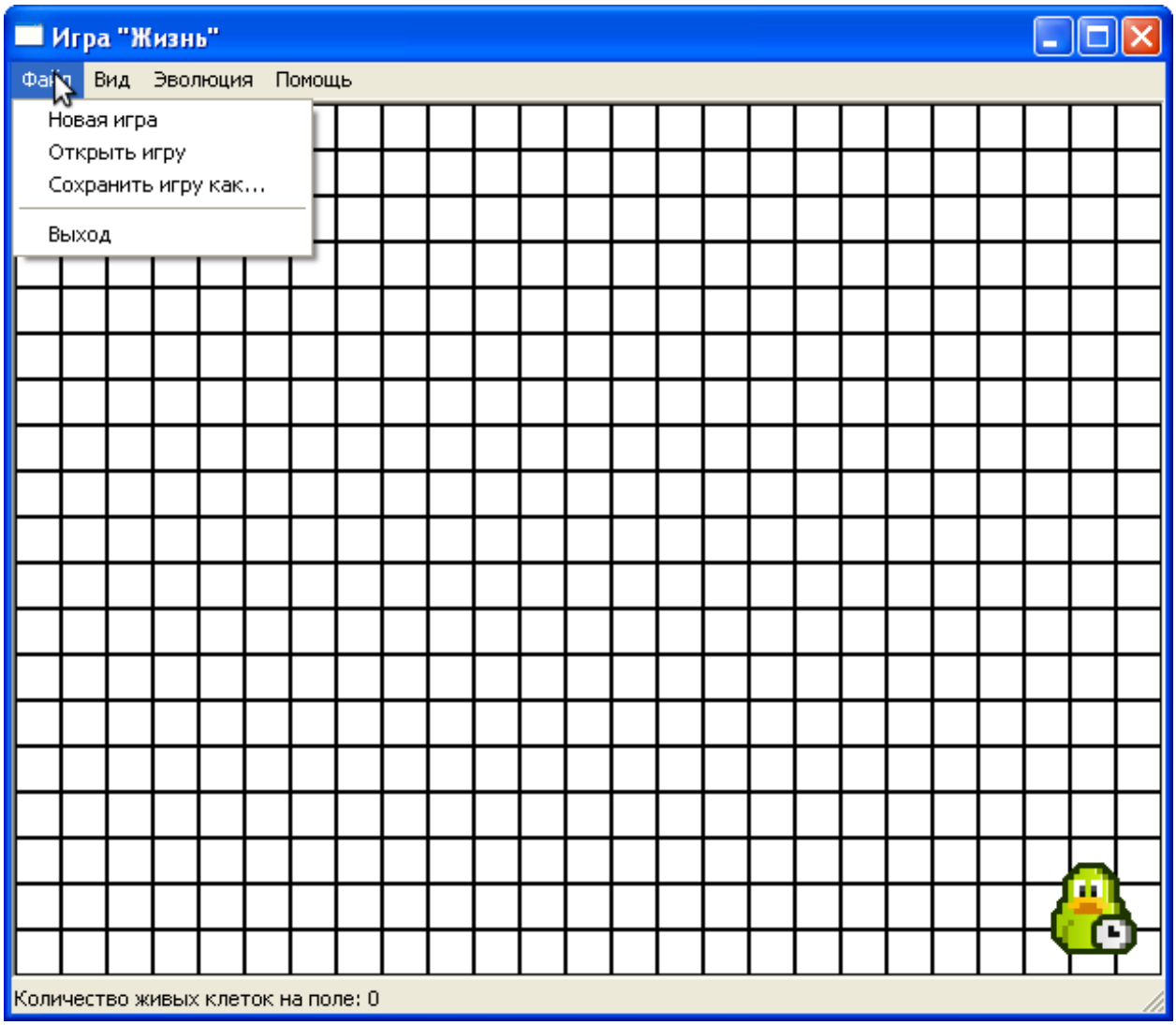

Рисунок 4 – Пункты меню «Файл»

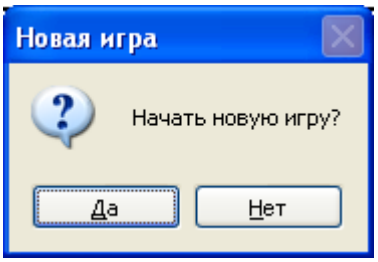

Рисунок 5 – Диалог создания новой игры, если на поле есть клетки

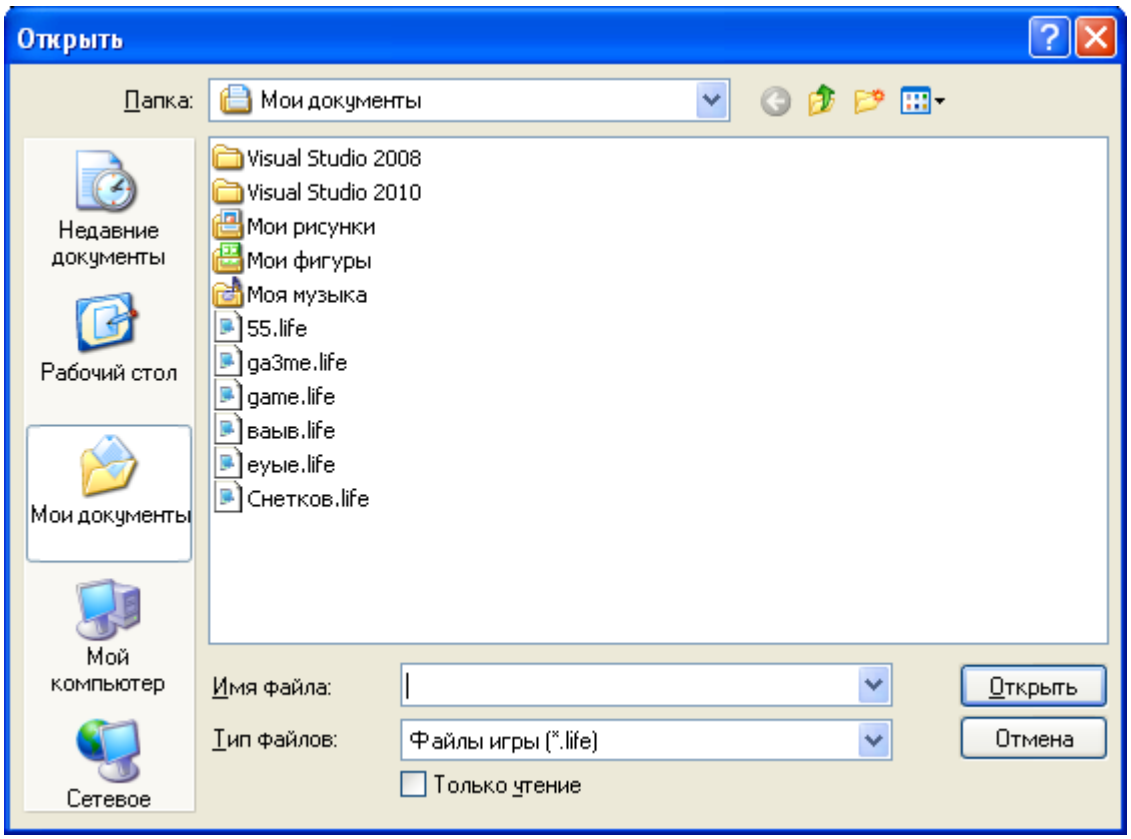

Рисунок 6 – Диалог открытия сохранённой игры с заданным расширением

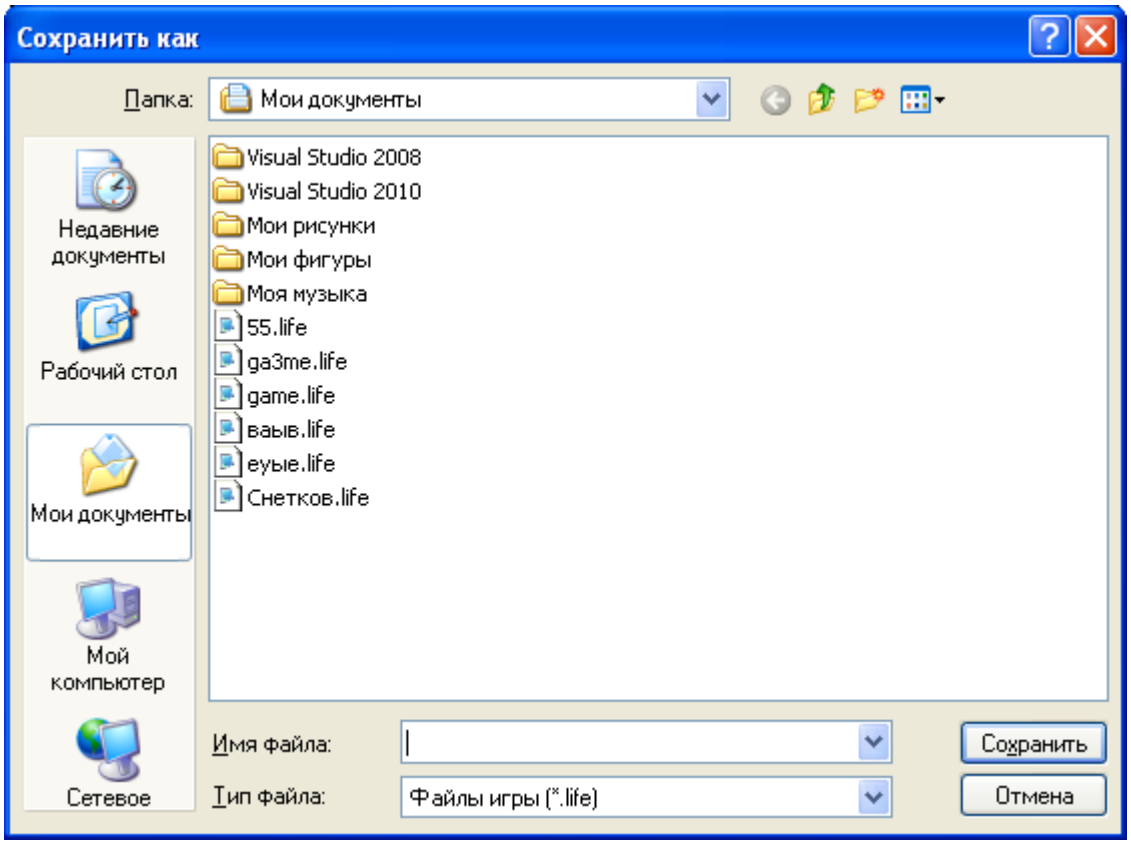

Рисунок 7 – Диалог сохранения текущей игры

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.

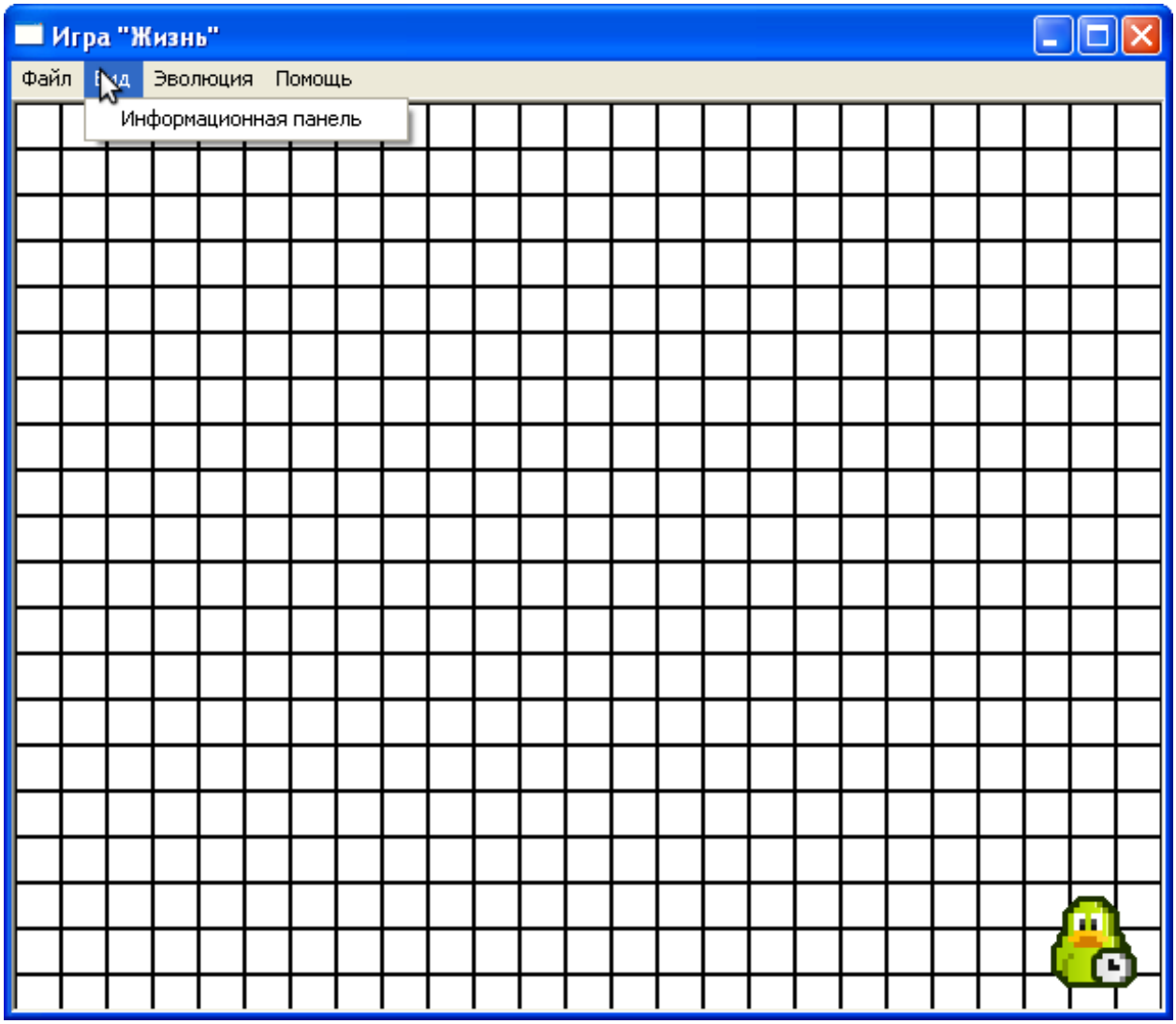

Рисунок 8 – Пункт меню «Вид» после выключения информационной панели

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

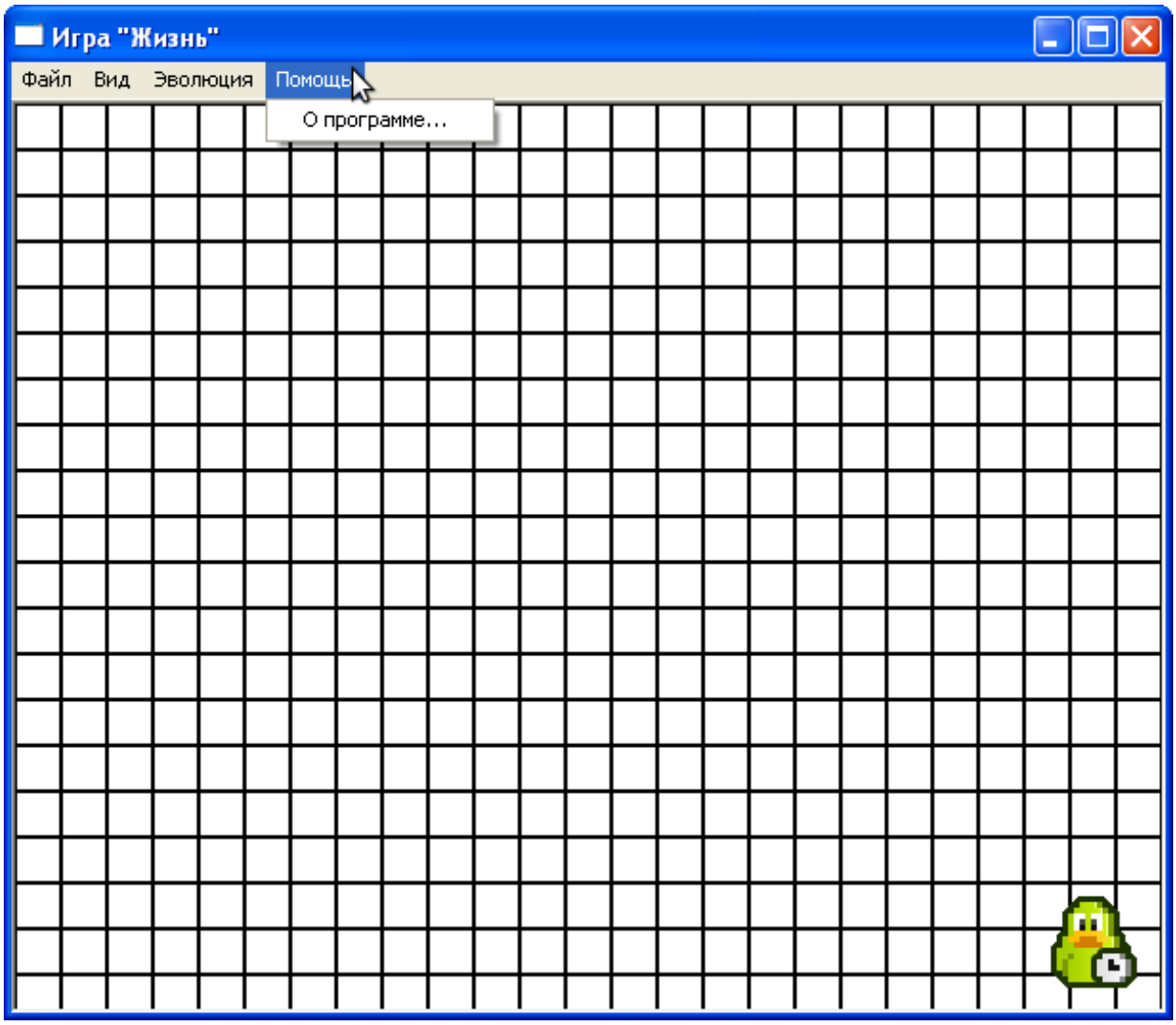

Рисунок 9 – Пункт меню «Помощь»

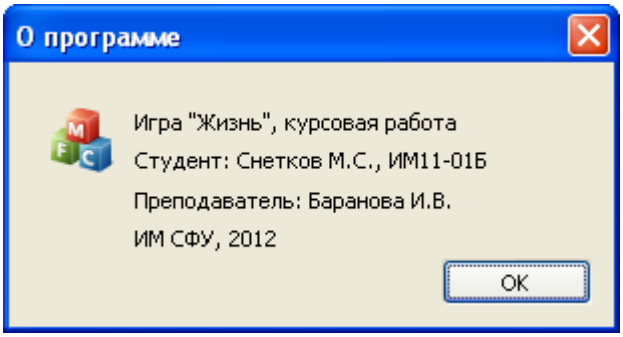

Рисунок 10 – Вызов диалога «О программе» из пункта «Помощь»

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).

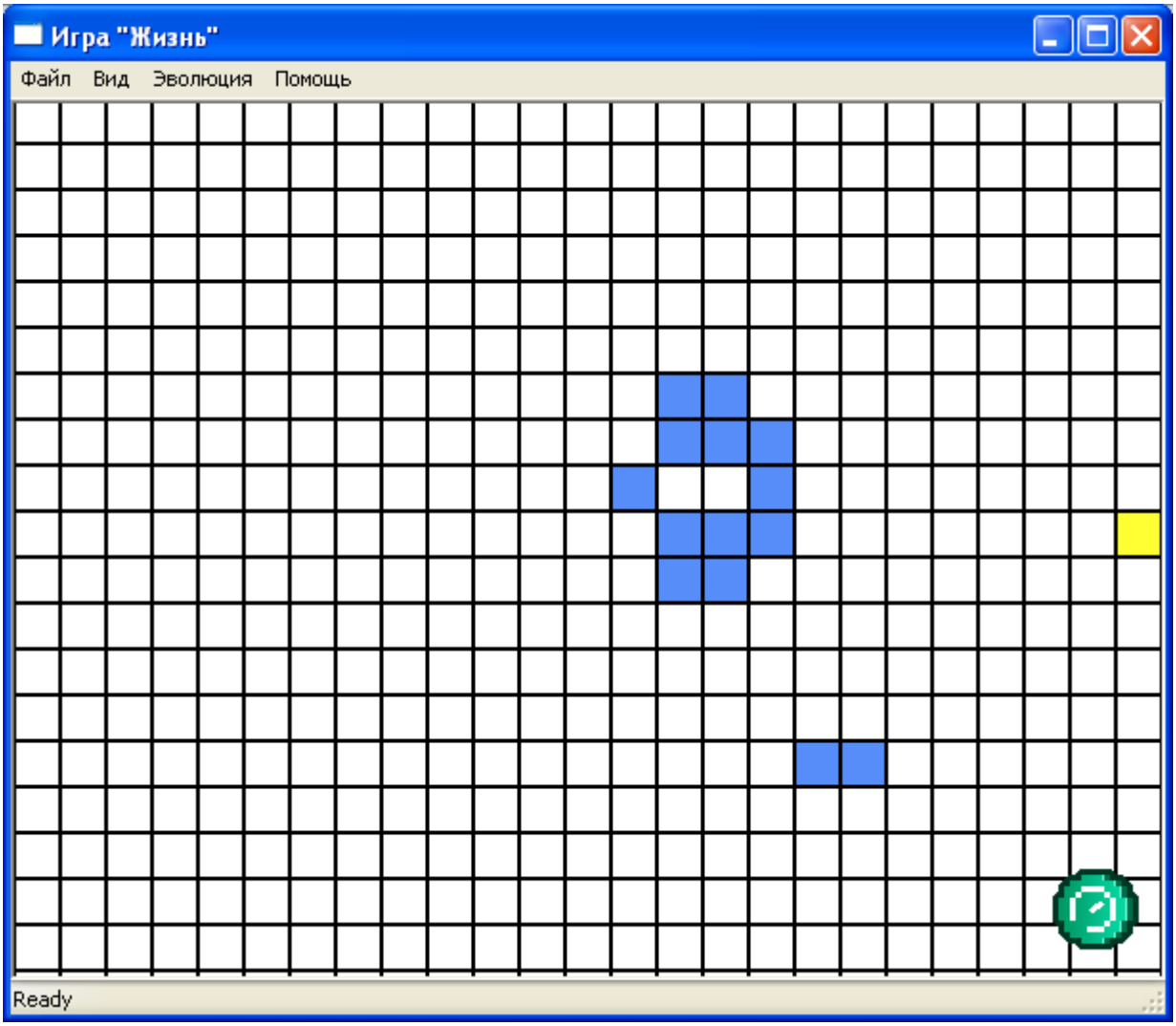

Рисунок 11 – Возможность расставлять и удалять клетки на игровом поле, клетка под курсором подсвечивается желтым пульсирующим цветом

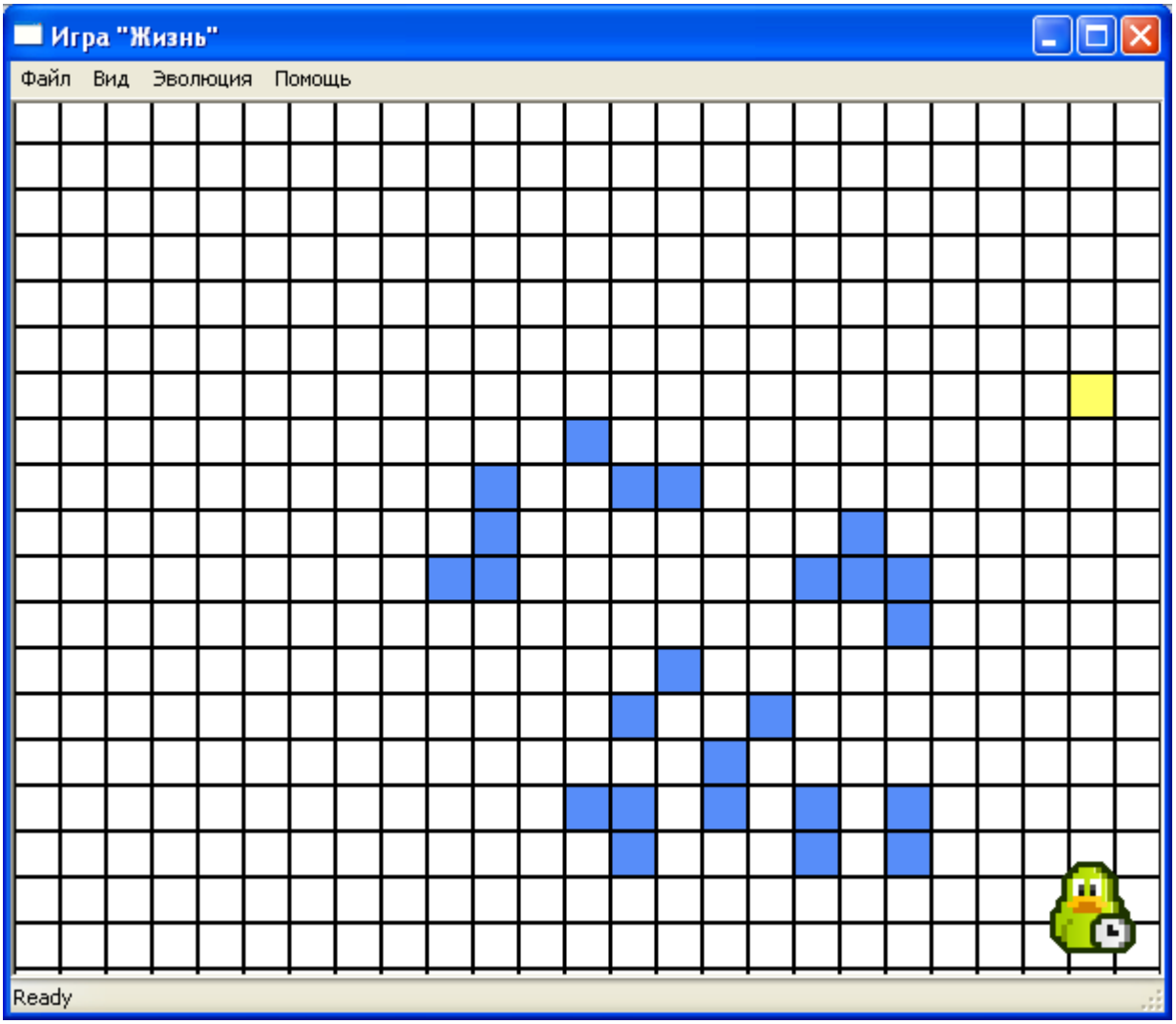

Рисунок 12 – Пример смены поколений (первоначальная колония)

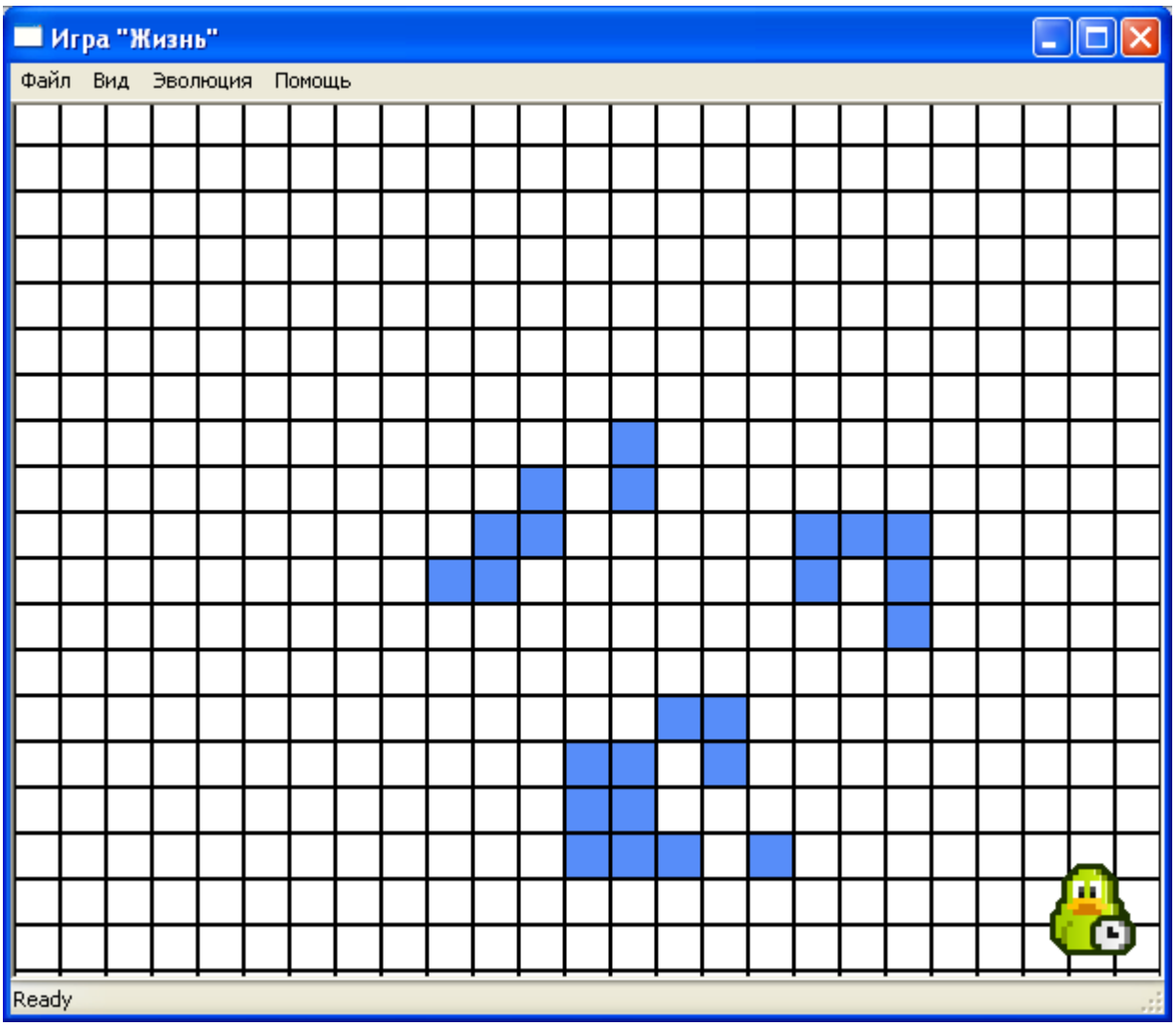

Рисунок 13 – Пример смены поколений (1-е поколение)

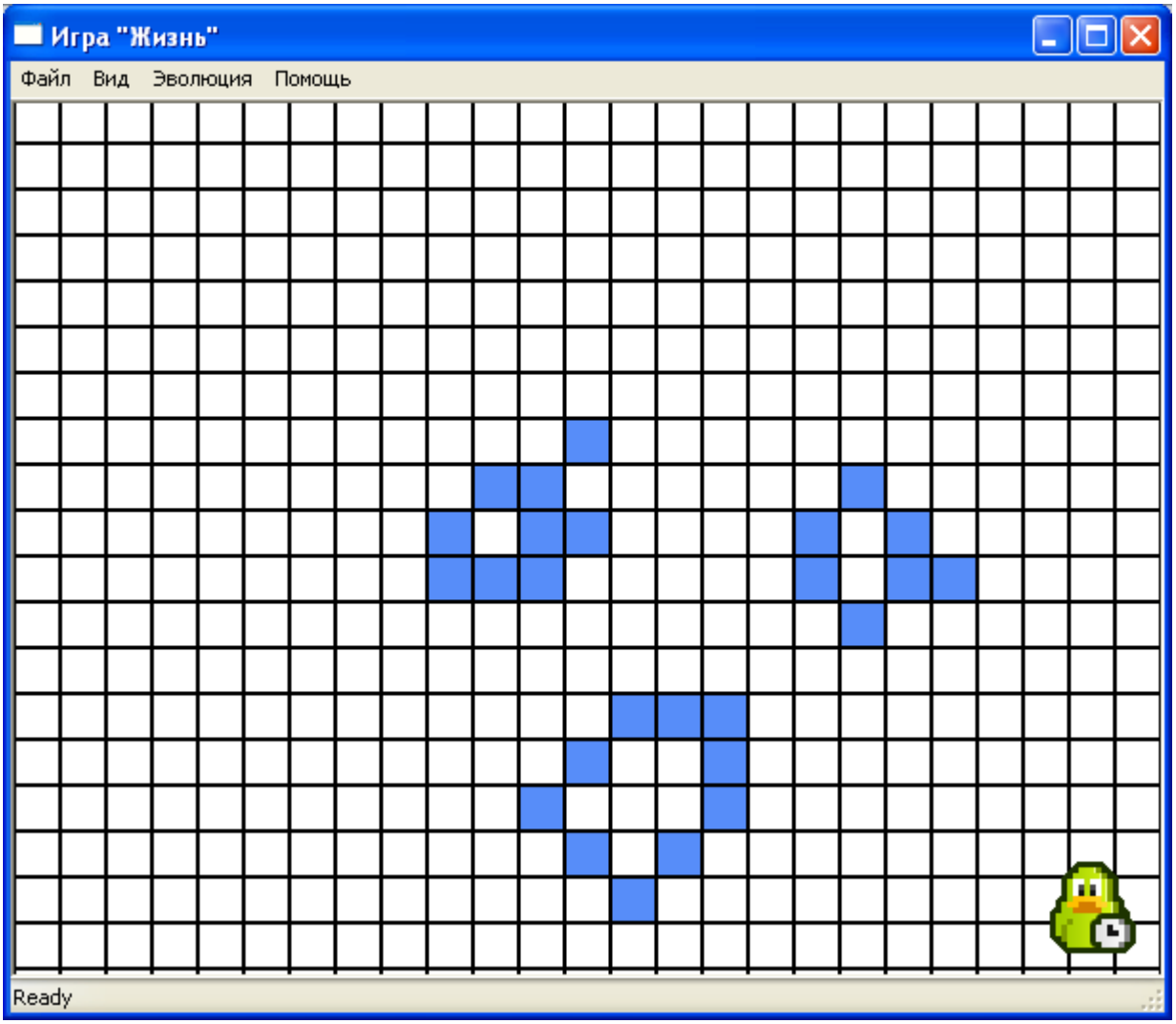

Рисунок 14 – Пример смены поколений (2-е поколение)

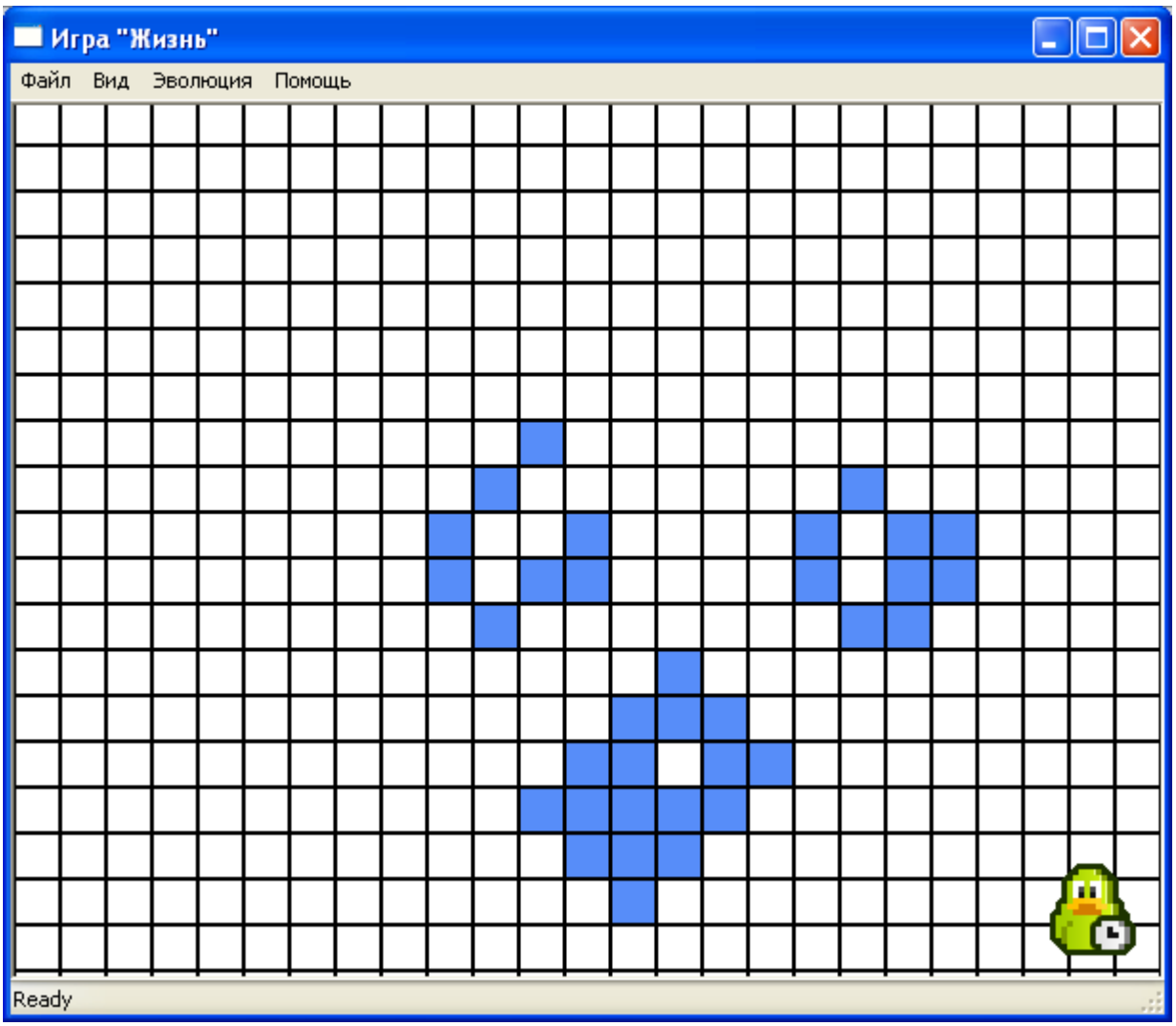

Рисунок 15 – Пример смены поколений (3-е поколение)

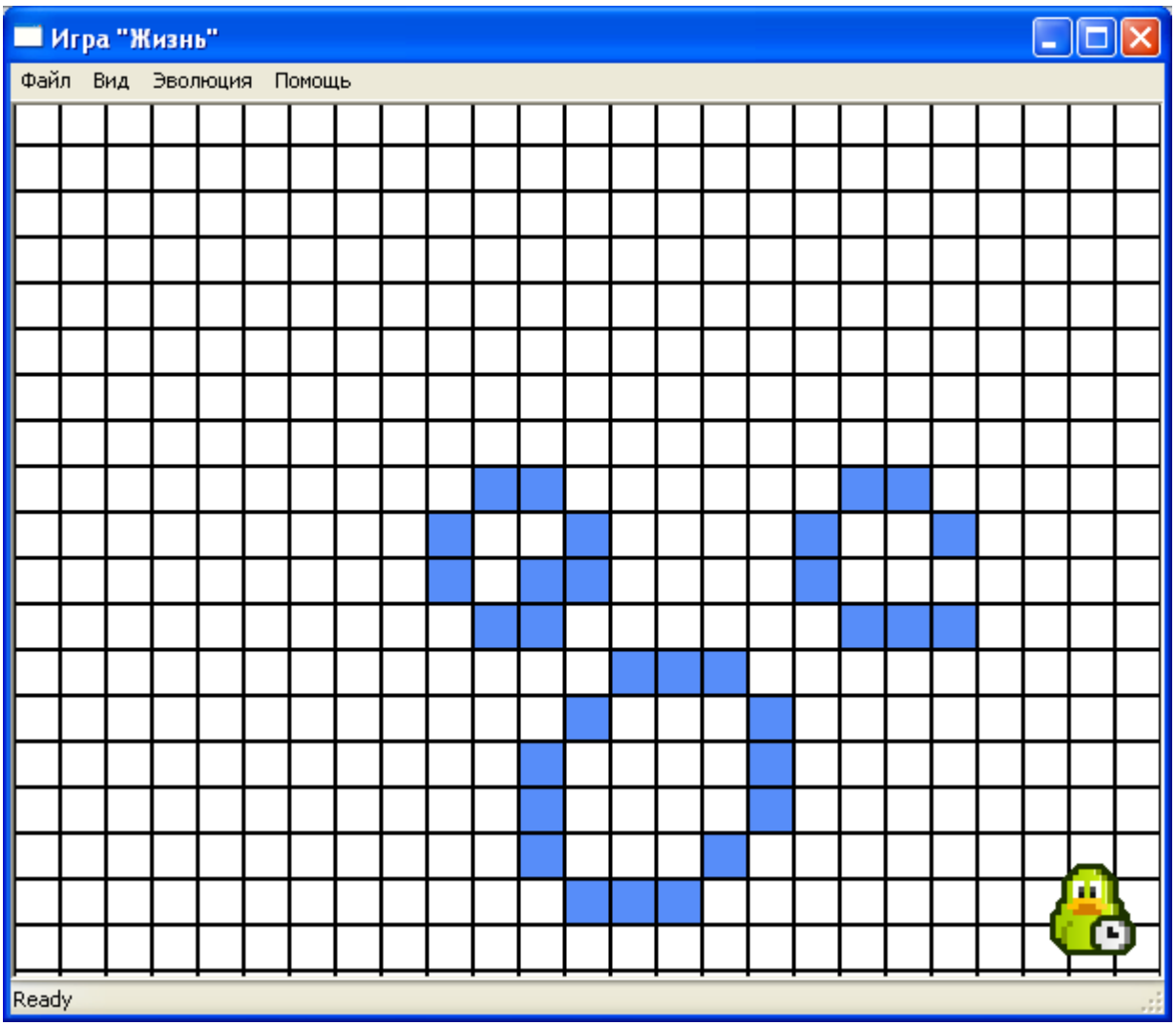

Рисунок 16 – Пример смены поколений (4-е поколение)

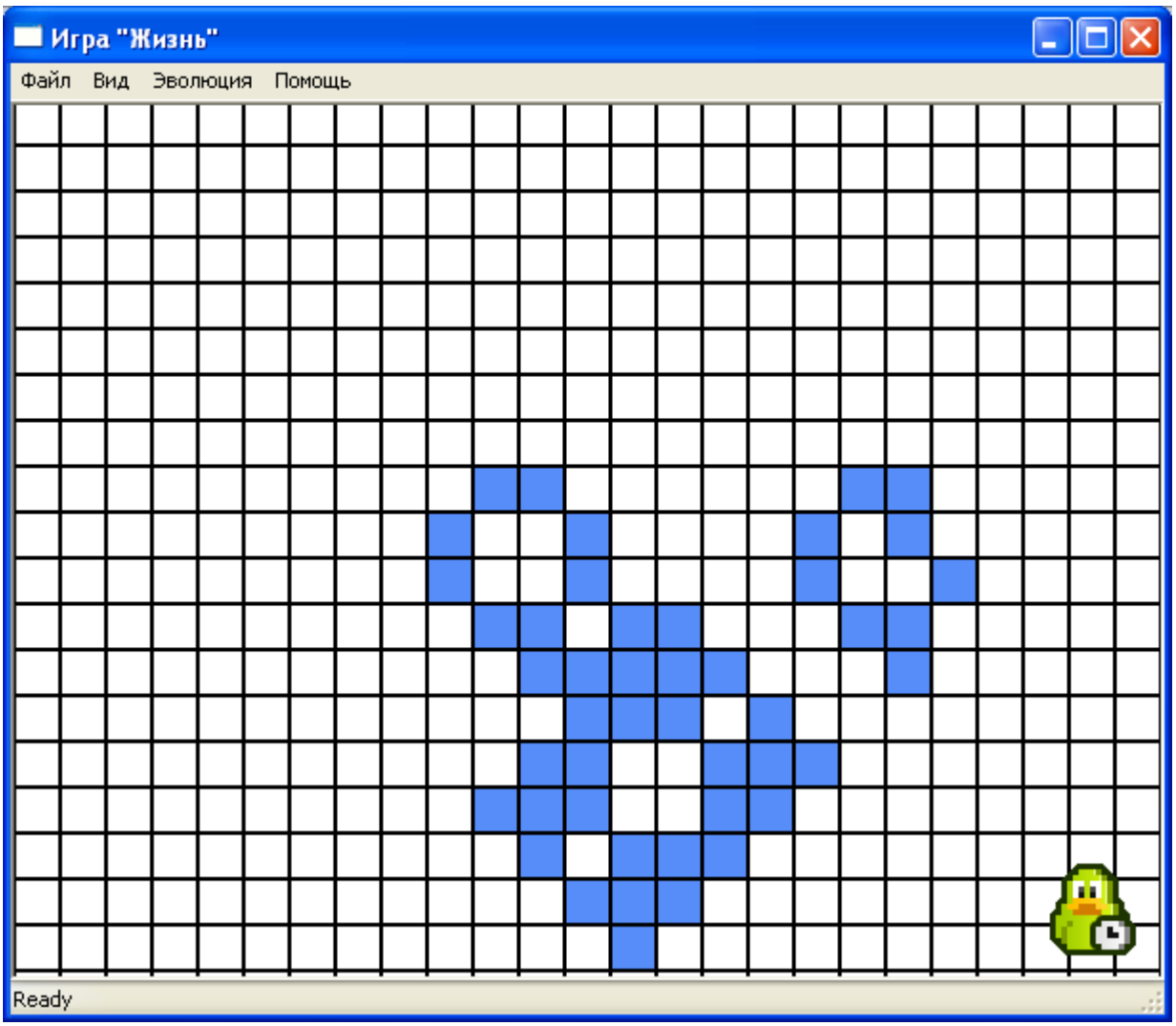

Рисунок 17 – Пример смены поколений (5-е поколение)

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.

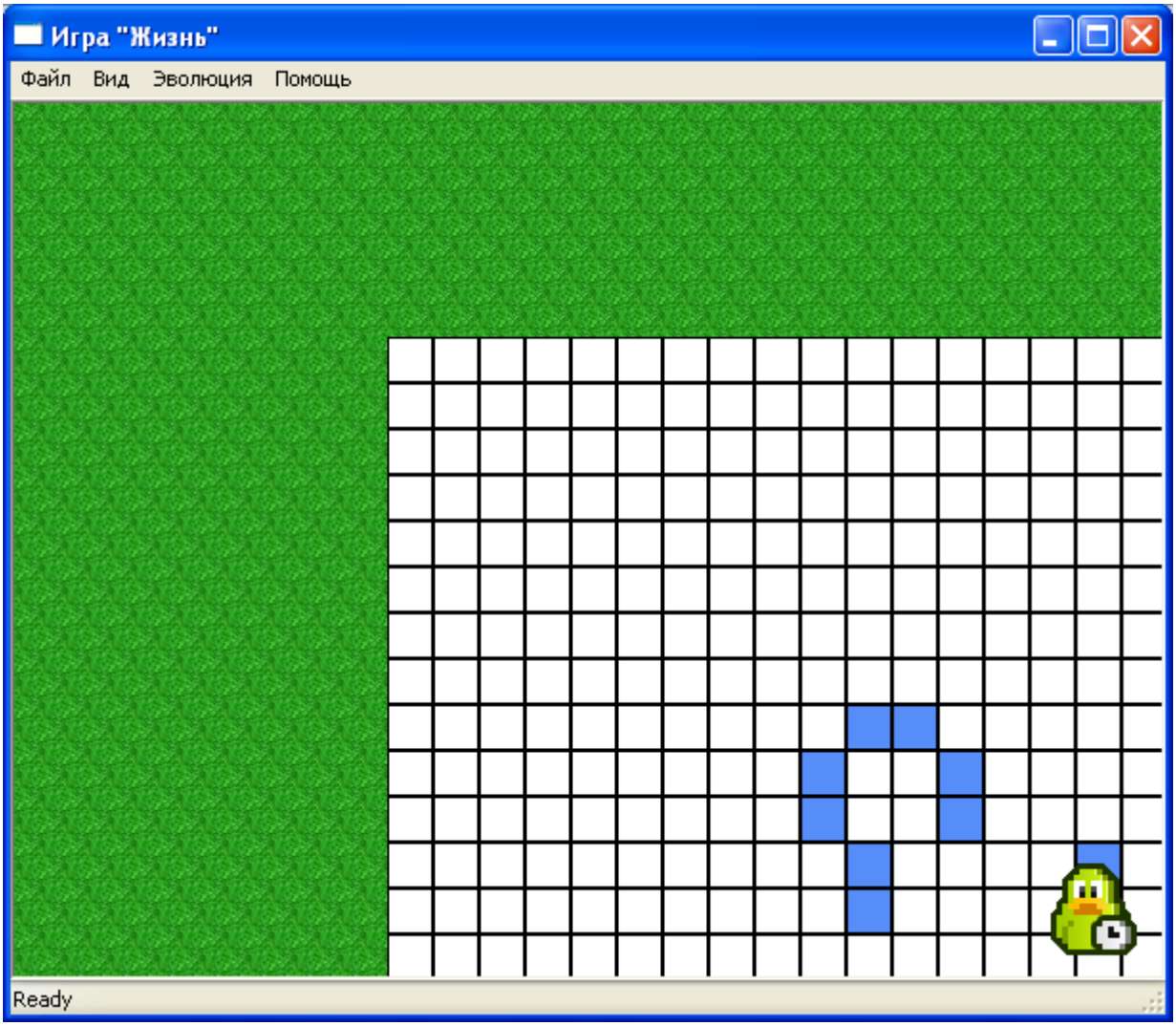

Рисунок 18 – Область вокруг поля с клетками заполняется тайтлами

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.

Приложение

Game.h

```
#pragma once

#ifndef __AFXWIN_H__
	#error "include 'stdafx.h' before including this file for PCH"
#endif

#include "stdafx.h"
#include "resource.h"
#include "Protoblast.h"
#include "Clock.h"
#include <vector>

/* Класс игровой логики, наследованный от класса приложения виндовса */
class CGameApp : public CWinApp
{
public:
	CGameApp();		// Конструктор
	~CGameApp();  // Деструктор

	BOOL InitInstance(); // Метод инициализации окна
	int ExitInstance();	// Метод выхода

	CSize GetFieldSize();	// Метод получения размера игрового поля
	void LeftButtonDoubleClick(UINT, CPoint);	// Метод от двойного клика левой
кнопки мыши
	void LeftButtonDown(UINT, CPoint);	// Метод от нажатия левой кнопки мыши
	void RightButtonUp(UINT, CPoint); // Метод от отпускания правой кнопки мыши
	void CreateNewGame();				// Метод создания новой игры
	void RenderField(CMemoryDC &, CRect);	// Метод рисования игрового поля
	void NcMouseMove();					// Метод от курсора, вышеднего за пределы
игрового поля
	void MouseMove(UINT, CPoint);	// Метод от курсора, бегающего по игровому полю
	void KeyDown(UINT, UINT, UINT);	// Метод от нажатия клавиши
	void Tick();						// Метод тика
	int GetProtoblastCount();		// Метод получения количества живых клеток
	int GetFPS();						// Метод получения частоты кадров в
секунду

	afx_msg void OnEvolve(UINT);	// Метод нажатия на один из пунктов установки
скорости эволюции, где-то в меню сверху
	afx_msg void OnAppAbout();		// Метод нажатия на пункт "О программе" в меню
сверху
	afx_msg void OnOpenGame();		// Метод нажатия на пункт "Открыть игру" в меню
сверху
	afx_msg void OnSaveGame();		// Метод нажатия на пункт "Сохранить игру как" в
меню сверху
	afx_msg void OnNewGame();		// Метод нажатия на пункт "Новая игра" в меню
сверху

	DECLARE_MESSAGE_MAP() // Макрос конца класса для объявления перехватываемых
сообщений

private:
	static const int m_fieldHeight = 69;	// Высота игрового поля
	static const int m_fieldWidth = 56;		// Ширина игрового поля
	static const int m_fps = 25;			// Частота кадров в секунду
```

```
	std::vector < std::vector < CProtoblast * > > m_field;// Двумерный массив клеток
на поле
	CPoint m_oldMousePosition; // Старая позиция указателя мыши
	CPoint m_hoverCell;        // Выделенная ячейка
	CPoint m_offset;           // Сдвиг поля для навигации
	CClock m_clock;                  // Индикатор режима эволюций
	int m_evolveTime;          // Время интервала между эволюциями
	int m_tickCount;           // Время с момента последней эволюции
	bool m_pause;              // А не на паузе ли игра?

	bool MousePositionToXY(CPoint, CPoint &);      // Метод реобразования координат
курсора в координаты клетки на поле
	void FreeField();          // Метод даления игрового поля
	void Evolve();                   // Метод эволюции-революции
};

extern CGameApp theApp;    // Объявляем объект класса везде!
```

Game.cpp

```
#include "stdafx.h"
#include "afxwinappex.h"
#include "afxdialogex.h"
#include "Game.h"
#include "MainFrm.h"
#include "Protoblast.h"
#include "AboutDialog.h"
#include <fstream>

BEGIN_MESSAGE_MAP(CGameApp, CWinApp)     // Макрос начала объявления перехватываемых
сообщений
	ON_COMMAND_RANGE(ID_EVOLVE_1_SECOND, ID_EVOLVE_HAND, &CGameApp::OnEvolve)  //
Сообщения выбора скорости эволюции
	ON_COMMAND(ID_APP_ABOUT, &CGameApp::OnAppAbout)		// Сообщение о
заинтересованности в просмотре пункта "О программе"
	ON_COMMAND(ID_NEW_GAME, &CGameApp::OnNewGame)		// Кто-то хочет начать новую
игру - вот и сообщение
	ON_COMMAND(ID_OPEN_GAME, &CGameApp::OnOpenGame)		// Сообщение пункта "Открыть
игру"
	ON_COMMAND(ID_SAVE_GAME, &CGameApp::OnSaveGame)		// Сообщение пункта
"Сохранить игру как"
END_MESSAGE_MAP()   // Макрос конца объявления перехватываемых сообщений

CGameApp::CGameApp()// Конструктор
{
	CreateNewGame();     // Создаём новую игру

	m_pause = false;     // Снимаем игру с паузы
	m_hoverCell = CPoint(-1, -1);     // Никакая ячейка не выделена
	m_offset = CPoint(0, 0);   // Поле никуда не сдвинуто
}

CGameApp::~CGameApp()      // Деструктор
{
	FreeField();  // Удаляем игровое поле из памяти
}

CGameApp theApp;     // Объект игрового класса

void CGameApp::OnEvolve(UINT id)  // Метод нажатия на один из пунктов установки скорости
эволюции, где-то в меню сверху
{
	CheckMenuRadioItem(GetMenu(*m_pMainWnd), ID_EVOLVE_1_SECOND, ID_EVOLVE_HAND, id,
MF_BYCOMMAND);      // Задаем выделенный пункт в диапазоне пункт в меню сверху

	switch (id)   // В зависимости от желаемого
	{
	case ID_EVOLVE_1_SECOND:   // Раз в секунду
		m_evolveTime = 1;    // Пока секунда
		break;

	case ID_EVOLVE_3_SECONDS:  // Раз в три секунды
		m_evolveTime = 3;    // Три секунды
		break;

	case ID_EVOLVE_10_SECONDS: // Раз в десять секунд
		m_evolveTime = 10;   // Десять секунд
		break;

	case ID_EVOLVE_42_MINUTES: // Раз в сорок две минуты
		m_evolveTime = 42 * 60;     // Сорок два умножить на шестьдесят секунд
```

```
			break;

		default:
			m_evolveTime = 0;	// Ручной режим эволюции
		}

		m_evolveTime *= m_fps;		// Умножаем на количество кадров в секунду
		m_tickCount = 0;		// С момента последней эволюции еще ни одного момента не
прошло
		m_clock.SetTime(m_evolveTime);	// Задаём индикатору как часто ему крутиться
}

BOOL CGameApp::InitInstance()		// Метод инициализации окна
{
		INITCOMMONCONTROLSEX initCtrls;	// Настройка контролов

		initCtrls.dwSize = sizeof(initCtrls);	// Размер
		initCtrls.dwICC = ICC_WIN95_CLASSES;		//		Стиль
		InitCommonControlsEx(&initCtrls); // Задаём
		CWinApp::InitInstance();	// Родителю даём возможность тоже проинициализироваться
		EnableTaskbarInteraction(FALSE);  // Настройки поведения в панели задач
//		SetRegistryKey(_T("Mihail Snetkov"));
//		AfxInitRichEdit2();
		CMainFrame *pFrame = new CMainFrame;		// Создаём окно

		if (!pFrame)  // Если не создали окно
			return FALSE; // То играть не во что

		m_pMainWnd = pFrame; // Иначе это у нас главное окно
		pFrame->LoadFrame(IDR_MAINFRAME); // Загружаем верхнее меню
		pFrame->ShowWindow(SW_SHOW);		// Режим отображения
		pFrame->UpdateWindow();		// Обновляем окно

		OnEvolve(ID_EVOLVE_3_SECONDS);		// Задаём частоту эволюций

		return TRUE;  // Все отлично, можно играть!
}

void CGameApp::CreateNewGame()		// Метод создания новой игры
{
		FreeField();  // Вдруг до этого уже играли и есть что удалить

		for (size_t i = 0; i < m_fieldHeight; ++i)		// Пробегая по высоте поля
		{
			m_field.push_back(std::vector < CProtoblast * >());	// Создаем массив
строк поля

			for (size_t j = 0; j < m_fieldWidth; j++)		// И пробегая по ширине поля
				m_field[i].push_back(new CProtoblast()); // Создаём клетку
		}
}

void CGameApp::FreeField() // Метод удаления поля
{
		for (size_t i = 0; i < m_field.size(); ++i)		// Пробегая по количеству строк в
поле
		{
			for (size_t j = 0; j < m_field[i].size(); ++j)  // Промегаем по всей строке
				delete m_field[i][j];		// И удаляем каждую клетку

			m_field[i].clear();  // Удаляем строку поля
		}

		m_field.clear();		// Удаляем массив строк
}
```

```
void CGameApp::LeftButtonDoubleClick(UINT nFlags, CPoint point)     // Метод от двойного
клика левой кнопки мыши
{
	CPoint cell;  // Представим, что в эту координату на поле кликнули

	if (MousePositionToXY(point, cell))      // Если попали кликом по полю
		m_field[cell.x][cell.y]->Click(); // То оживляем клетку
}

void CGameApp::LeftButtonDown(UINT nFlags, CPoint point)     // Метод от нажимания левой
кнопки мыши
{
	m_oldMousePosition = CPoint(-1, -1);     // Сбрасываем последнее положение кнопки
мыши
}

void CGameApp::RightButtonUp(UINT nFlags, CPoint point)      // Метод от отпускания
правой кнопки мыши
{
	CPoint cell;  // Будто сюда кликнули

	if (MousePositionToXY(point, cell))      // Если попали в поле
		m_field[cell.x][cell.y]->Click(false);   // Убиваем клетку

	MouseMove(0, point); // Скорее всего убитую клетку надо выделить
}

void CGameApp::NcMouseMove()      // Метод от курсора, вышеднего за пределы игрового поля
{
	if (m_hoverCell == CPoint(-1, -1))       // Если никакая клетка не была выделена
		return;       // То, собственно, делать нечего

	m_field[m_hoverCell.x][m_hoverCell.y]->Unhover();      // Иначе унвыделяем клетку
	m_hoverCell = CPoint(-1, -1);     // И теперь точно выделять нечего
}

void CGameApp::MouseMove(UINT nFlags, CPoint point)   // Метод от передвижения курсора по
игровому полю
{
	if (m_pause)  // Если игра на паузе
		return;       // То делать нечего

	if (nFlags == MK_LBUTTON)  // Если двигаем с зажатой левой кнопкой мыши
	{
		if (m_oldMousePosition != CPoint(-1, -1))      // Если мышь чуть-чуть
передвинулась, потянув за собой поле
			m_offset += point - m_oldMousePosition;  // То тянем поле за мышкой

		m_oldMousePosition = point;// А теперь это актуальное положение мыши
	}

	CPoint cell;  // Вдруг задели клетку на поле

	if (MousePositionToXY(point, cell))      // Если попали по клетке, пока водили
курсором
	{
		if (m_hoverCell != cell && m_hoverCell != CPoint(-1, -1))    // Если клетка
не выделена, а есть другая выделенная клетка
			m_field[m_hoverCell.x][m_hoverCell.y]->Unhover();     // То другую
веделенную клетку унвыделим

		m_hoverCell = cell;  // А это выделим
		m_field[cell.x][cell.y]->Hover(); // Выделим, я сказал. :)
	}
```

```
	else	// Иначе если по клетке не попали
		if (m_hoverCell != CPoint(-1, -1))	// Если есть выделенные клетки
		{
			m_field[m_hoverCell.x][m_hoverCell.y]->Unhover();	// Унвыделим
выделенную
			m_hoverCell = CPoint(-1, -1);	// И пометим, что ничего не
выделено
		}
}

void CGameApp::Evolve()	// Метод эволюции-революции
{
	int movX[8] = {-1, -1, -1, 0, 1, 1, 1, 0};	// Соседи по оси X
	int movY[8] = {-1, 0, 1, 1, 1, 0, -1, -1};	// Соседи по оси Y

	std::vector < std::vector < bool > > tempField(m_fieldHeight, std::vector < bool
>(m_fieldWidth, false));	// Создаём пустое поле

	for (int i = 0; i < m_fieldHeight; ++i)	// Пробегаем по всем строкам поля
		for (int j = 0; j < m_fieldWidth; ++j)	// Ну и по каждой строке в
частности
		{
			tempField[i][j] = m_field[i][j]->IsLive();	// Помечаем, если
клетка живая

			int count = 0;	// Количество соседей

			for (int k = 0; k < 8; ++k)// Пробегаем по соседям
			{
				int x = i + movX[k];// Координаты соседа по оси X
				int y = j + movY[k];// Координаты соседа по оси Y

				if (x < 0 || y < 0 || x >= m_fieldHeight || y >= m_fieldWidth)
	// Если это сосед из параллельного мира
					continue;	// То с ним лучше не связываться

				if (m_field[x][y]->IsLive())	// Если сосед живой
					++count;	// То он добавляется в копилку живых
соседей
			}

			if (m_field[i][j]->IsLive())	// Если текущая клетка живая
			{
				if (count < 2 || count > 3)// И количество соседей не в норме
					tempField[i][j] = false;	// То пора умереть
			}
			else	// Иначе текущая клетка не живая
			{
				if (count == 3)	// Но количество соседей позволяет
					tempField[i][j] = true;	// Клетке ожить
			}
		}


	for (int i = 0; i < m_fieldHeight; ++i)	// Пробегаем по всем строкам
		for (int j = 0; j < m_fieldWidth; ++j)	// В каждой строке
			if (tempField[i][j] ^ m_field[i][j]->IsLive())	// Если у нас есть
разница в статусе
			{
				if (tempField[i][j])// То если клетке суждено ожить
					m_field[i][j]->Click();	// Она оживёт
				else	// Иначе
					m_field[i][j]->Click(false);	// Она умрём
			}
}
```

```
void CGameApp::Tick()      // Метод тика
{
	if (m_pause)  // Если пауза
		return;       // То ничего делать не надо

	++m_tickCount;       // Увеличиваем счетчик с последней эволюции

	for (int i = 0; i < m_fieldHeight; ++i)  // Пробегаем по всем строкам
		for (int j = 0; j < m_fieldWidth; ++j)   // И по каждой строке в частности
			m_field[i][j]->Tick();     // Передаём тик в клетки

	m_clock.Tick();      // Часы тоже тикают

	if (m_evolveTime && m_tickCount % m_evolveTime == 0)  // Если пришло время для
эволюции
		Evolve();     // То эволюции быть
}

void CGameApp::RenderField(CMemoryDC &dc, CRect screen)      // Метод отрисовки игрового
поля
{
	CSize protoblastSize(m_field[0][0]->GetSize()); // Получаем размеры клетки

	for (int i = 0; i < m_fieldHeight; ++i)  // Пробегаем по всем строкам
		for (int j = 0; j < m_fieldWidth; ++j)   // И в каждой строке
			m_field[i][j]->Render(dc, CPoint(j * protoblastSize.cx, i *
protoblastSize.cy) + m_offset);   // Отрисовываем клетки


	int deltaLeft = m_offset.x;// Свободное пространство слева
	int deltaRight = screen.Width() - m_offset.x - protoblastSize.cx;   // Справа
	int deltaTop = m_offset.y; // Сверху
	int deltaBottom = screen.Height() - m_offset.y - protoblastSize.cy; // И снизу
	CBitmap bmp;  // Рисунок травы (тайтл)
	CDC dcMemory; // Контекст для рисования в памяти

	bmp.LoadBitmap(IDB_GRASS); // Загружаем травку
	dcMemory.CreateCompatibleDC(dc);  // Создаем совместимый контекста
	dcMemory.SelectObject(&bmp);      // Выбираем рисунок

	if (deltaTop > 0)    // Если сверху не хватает зелени, то заполняем
		for (int i = 1; i < deltaTop / protoblastSize.cy + 2; ++i)
			for (int j = -1; j < screen.Width() / protoblastSize.cx + 2; ++j)
			{
				int x = (m_offset.x % protoblastSize.cx) + protoblastSize.cx *
j;     // Координаты травы по оси X
				int y = m_offset.y - protoblastSize.cy * i;     // Координаты
травы по оси Y

				dc->BitBlt(x, y, protoblastSize.cx, protoblastSize.cy,
&dcMemory, 0, 0, SRCCOPY); // Рисуем траву
			}

	if (deltaBottom > 0)// Если снизу не хватает зелени, то опять её дорисовываем
		for (int i = 0; i < deltaBottom / protoblastSize.cy; ++i)
			for (int j = -1; j < screen.Width() / protoblastSize.cx + 2; ++j)
			{
				int x = (m_offset.x % protoblastSize.cx) + protoblastSize.cx *
j;
				int y = m_offset.y + protoblastSize.cy * (m_fieldHeight + i);

				dc->BitBlt(x, y, protoblastSize.cx, protoblastSize.cy,
&dcMemory, 0, 0, SRCCOPY);
			}
```

```
	if (deltaLeft > 0)  // Если слева пусто, то тоже траву рисуем
		for (int i = 0; i < m_fieldHeight; ++i)
			for (int j = 1; j < deltaLeft / protoblastSize.cx + 2; ++j)
			{
				int x = m_offset.x - protoblastSize.cx * j;
				int y = m_offset.y + protoblastSize.cy * i;

				dc->BitBlt(x, y, protoblastSize.cx, protoblastSize.cy,
&dcMemory, 0, 0, SRCCOPY);
			}

	if (deltaRight > 0) // Справа пусто? Нарисуем траву!
		for (int i = 0; i < m_fieldHeight; ++i)
			for (int j = 0; j < deltaRight / protoblastSize.cx; ++j)
			{
				int x = m_offset.x + (m_fieldWidth + j) * protoblastSize.cx;
				int y = m_offset.y + protoblastSize.cy * i;

				dc->BitBlt(x, y, protoblastSize.cx, protoblastSize.cy,
&dcMemory, 0, 0, SRCCOPY);
			}

	if (screen.Height() > 60 && screen.Width() > 60 && !m_pause) // Если часы-
индикатор влезают в окно
		m_clock.Render(dc, CPoint(screen.Width(), screen.Height()) - CPoint(10, 10)
- CPoint(m_clock.GetWidth(), m_clock.GetHeight()));   // То и их рисуем
}

int CGameApp::GetProtoblastCount()	// Метод получения количества живых клеток
{
	int count = 0;	// Только экстрасенс на текущий момент сколько живых клеток на
поле

	for (int i = 0; i < m_fieldHeight; ++i)  // А мы пробегаем по всем строкам
		for (int j = 0; j < m_fieldWidth; ++j)   // В каждой строке по всем клеткам
			if (m_field[i][j]->IsLive())	// И если клетка жива
				++count;	// То увеличиваем счётчик

	return count; // Возвращаем результат
}

CSize CGameApp::GetFieldSize()    // Метод получения размеров поля
{
	return CSize(m_fieldWidth, m_fieldHeight);	// Формируем размер поля
}

int CGameApp::ExitInstance()	// Метод выхода
{
	return CWinApp::ExitInstance();   // Родительский класс тоже хочет выйти
}

int CGameApp::GetFPS()	// Метод получения количества кадров в секунду
{
	return m_fps; // Количество кадров в секунду
}

bool CGameApp::MousePositionToXY(CPoint position, CPoint &cell)	// Метод получения
координат клетки по координатам курсора мыши
{
	position -= m_offset;	// Убираем сдвиг

	if (position.x < 0 || position.y < 0)    // Если мы не в поле
		return false; // То сразу нет
```

```
	int y = position.x / m_field[0][0]->GetWidth(); // Иначе координаты по оси Y
	int x = position.y / m_field[0][0]->GetHeight();	// Потом по оси X

	if (x >= m_fieldHeight || y >= m_fieldWidth)	// Если поле кончилось
		return false; // То опять нет

	cell = CPoint(x, y); // Иначе искомые координаты клетки поля под курсором

	return true;  // Все хорошо, данные годные
}

void CGameApp::OnAppAbout()// Метод вызова меню "О программе"
{
	m_pause = true;	// Ставим на паузу

	CAboutDlg aboutDlg;  // Диалог "О программе"

	aboutDlg.DoModal();  // Открываем диалог, наслаждаемся
	m_pause = false;	// Закрыли диалог, сняли с паузы игру
}

void CGameApp::OnOpenGame()// Метод вызова меню "Открыть игру"
{
	m_pause = true;	// Ставим на паузу

	CFileDialog dialog(TRUE, _T("*.life"), _T(""), 0, _T("Файлы игры
(*.life)|*.life|")); // Создаём диалог открытия файла

	if (dialog.DoModal() == IDOK)	// Если нажали OK
	{
		CreateNewGame();	// То создаем новую игру

		std::ifstream fin(dialog.GetPathName()); // Открываем файл с сохраненной
игрой
		int count;	// Количество живых клеток в файле

		fin >> count; // Получаем из файла количество клеток

		for (int i = 0; i < count; ++i)	// И теперь
		{
			int x, y;	// Координаты клетки на поле

			fin >> x >> y;	// Считываем координаты

			if (x > 0 && y > 0 && x < m_fieldHeight && y < m_fieldWidth) // Если
клетка влезаем на поле
				m_field[x][y]->Click();	// Она оживаем
		}

		MessageBox(NULL, _T("Игра загружена успешно!"), _T(""),
MB_ICONINFORMATION); // Радостное известие
		OnEvolve(ID_EVOLVE_HAND);  // Режим эволюции - ручками
	}

	m_pause = false;	// Снимаем с паузы
}

void CGameApp::OnSaveGame()// Метод вызова меню "Сохранить игру как"
{
	if (GetProtoblastCount() == 0)	// Если живых клеток нет
	{
		MessageBox(NULL, _T("Нечего сохранять!"), _T(""), MB_ICONERROR);	// То
плохая новость

		return;	// И можно ничего не сохранять
```

```
	}

	m_pause = true;	// Ставим на паузу

	CFileDialog dialog(FALSE, _T("*.life"), _T(""), OFN_HIDEREADONLY |
OFN_OVERWRITEPROMPT, _T("Файлы игры (*.life)|*.life|"));	// Создаем диалог сохранения

	if (dialog.DoModal() == IDOK)	// Если пользователь выбрал куда сохранить
	{
		std::ofstream fout(dialog.GetPathName());	// Открываем этот файл

		fout << GetProtoblastCount() << std::endl;	// Выводим количество живых
клеток
		fout << std::endl;

		for (int i = 0; i < m_fieldHeight; ++i)  // А потом пробегаем по всем
строкам
			for (int j = 0; j < m_fieldWidth; ++j)	// В каждой строке по всем
клеткам
				if (m_field[i][j]->IsLive())	// И если клетка живая
					fout << i << " " << j << std::endl;	// То ей
суждено быть записанной в файл

		MessageBox(NULL, _T("Игра сохранена успешно!"), _T(""),
MB_ICONINFORMATION); // Радостное сообщение
	}

	m_pause = false;	// Пауза закончилась
}

void CGameApp::KeyDown(UINT nChar, UINT nRepCnt, UINT nFlags)	// Метод от нажатия
клавишы
{
	if (!m_pause && nChar == 32 && m_evolveTime == 0)	// Если нажали пробел и игра
не на паузе и эволюция ручная
		Evolve();	// То время для эволюции
}

void CGameApp::OnNewGame() // Метод вызова меню "Новая игра"
{
	if (GetProtoblastCount() == 0)	// Если и так чисто
		return;	// То делать нечего

	m_pause = true;	// Иначе пауза

	if (MessageBox(NULL, _T("Начать новую игру?"), _T("Новая игра"), MB_YESNO |
MB_ICONQUESTION) == IDYES) // Интересный вопрос
		CreateNewGame();	// Если согласны на новую игру, то делаем её

	m_pause = false;	// Пауза закончилась
}
```

MainFrm.h

```
#pragma once

#include "ChildView.h"

/* Класс основого фрейма, наследованный от класса основного окна */
class CMainFrame : public CFrameWnd
{
public:
	CMainFrame(); // Конструктор по-умолчанию
	~CMainFrame();	// Деструктор

	BOOL PreCreateWindow(CREATESTRUCT &);	// Метод подготовки к созданию окна
	BOOL OnCmdMsg(UINT, int, void *, AFX_CMDHANDLERINFO *);	// Метод получения
сообщений

private:
	CStatusBar m_wndStatusBar; // Статус-бар
	CChildView m_wndView;		// Вид

	afx_msg void OnTimer(UINT);			// Метод перехвата тика таймера
	afx_msg int OnCreate(LPCREATESTRUCT);	// Метод перехвата сообщения создания окна
	afx_msg void OnSetFocus(CWnd *);	// Метод перехвата получения фокуса окна

	DECLARE_MESSAGE_MAP()			// Макрос конца класса для объявления
перехватываемых сообщений
	DECLARE_DYNAMIC(CMainFrame)	// Макрос возможности получения информации о
классе во время выполнения программы
};
```

MainFrm.cpp

```
#include "stdafx.h"
#include "Game.h"
#include "MainFrm.h"

IMPLEMENT_DYNAMIC(CMainFrame, CFrameWnd) // Макрос возможности получения информации о
классе во время выполнения программы

BEGIN_MESSAGE_MAP(CMainFrame, CFrameWnd) // Макрос начала объявления перехватываемых
сообщения
	ON_WM_CREATE()			// Перехват сообщения создания окна
	ON_WM_SETFOCUS()	// Перехват сообщения получения фокуса окна
	ON_WM_TIMER()		// Перехват сообщения тика таймера
END_MESSAGE_MAP()		// Макрос конца объявления перехвата сообщений

static UINT indicators[] ={ID_SEPARATOR};	// Массив элементов статус-бара

CMainFrame::CMainFrame()	// Конструктор по-умолчанию
{
}

CMainFrame::~CMainFrame()	// Деструктор
{
}

int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)	// Метод перехвата сообщения
создания окна
{
	if (CFrameWnd::OnCreate(lpCreateStruct) == -1)	// Если не удалось создать окно
		return -1;	// Всё плохо

	if (!m_wndView.Create(NULL, NULL, AFX_WS_DEFAULT_VIEW, CRect(0, 0, 0, 0), this,
AFX_IDW_PANE_FIRST, NULL)) // Если не получилось создать вид
		return -1;	// Всё плохо

	if (!m_wndStatusBar.Create(this)) // Если не удалось создать статус-бар
		return -1;	// Всё плохо

	m_wndStatusBar.SetIndicators(indicators, sizeof(indicators) / sizeof(UINT));
	// Установка индикаторов статус-бара
	SetTimer(1, 1000 / theApp.GetFPS(), NULL);	// Установка таймера

	return 0;	// Всё хорошо
}

void CMainFrame::OnTimer(UINT nIDEvent)	// Метод перехвата сообщения тика таймера
{
	theApp.Tick();	// Передаём управление объекту игровой логики
	m_wndView.RedrawWindow();	// Запрос на перерисовку вида

	CString text; // Текст в статус-бар

	text.Format(_T("Количество живых клеток на поле: %d"),
theApp.GetProtoblastCount());	// Форматируем информацию
	m_wndStatusBar.SetPaneText(0, text);	// Добавляем в статус-бар
}

BOOL CMainFrame::PreCreateWindow(CREATESTRUCT &cs)	// Метод подготовки к созданию окна
{
	if(!CFrameWnd::PreCreateWindow(cs))	// Если родительскому классу подготовиться
к созданию окна
		return FALSE; // Выходим
```

```
	cs.dwExStyle &= ~WS_EX_CLIENTEDGE;	// Впуклая рамка окна
	cs.lpszClass = AfxRegisterWndClass(0);	// Класс

	return TRUE;  // Всё хорошо, можно создавать окно
}

void CMainFrame::OnSetFocus(CWnd *pOldWnd)	// Метод перехвата получения фокуса окна
{
	m_wndView.SetFocus();	// Передаём управление виду
}

BOOL CMainFrame::OnCmdMsg(UINT nID, int nCode, void *pExtra, AFX_CMDHANDLERINFO
*pHandlerInfo)	// Метод получения сообщений
{
	if (m_wndView.OnCmdMsg(nID, nCode, pExtra, pHandlerInfo))	// Если вид забрал
сообщение
		return TRUE;  // То хватит их пересылать

	return CFrameWnd::OnCmdMsg(nID, nCode, pExtra, pHandlerInfo);	// Иначе пусть
родительский класс с сообщениями разбирается
}
```

ChildView.h

```
#pragma once

#include "Protoblast.h"

/* Класс дочернего вида, наследуемый от класса окна */
class CChildView : public CWnd
{
public:
	CChildView(); // Конструктор по-умолчанию
	~CChildView();	// Деструктор

private:
	BOOL PreCreateWindow(CREATESTRUCT &);	// Метод подготовки к созданию окна

	afx_msg void OnLButtonDblClk(UINT, CPoint);	// Метод перехвата двойного клика
левой кнопки мыши
	afx_msg void OnNcMouseMove(UINT, CPoint);	// Метод перехвата передвижения
курсора вне области вида
	afx_msg void OnLButtonDown(UINT, CPoint);	// Метод перехвата нажатия левой
кнопки мыши
	afx_msg void OnRButtonUp(UINT, CPoint);		// Метод перехвата отпускания
правой кнопки мыши
	afx_msg void OnMouseMove(UINT, CPoint);		// Метод перехвата передвижения
курсора в области вида
	afx_msg void OnKeyDown(UINT, UINT, UINT);	// Метод нажатия на клавишу
	afx_msg void OnPaint();					// Метод рисования в
виде
	afx_msg BOOL OnEraseBkgnd(CDC *);		// Метод определения необходимости
очистки фона

	DECLARE_MESSAGE_MAP()	// Макрос конца класса для объявления перехватываемых
сообщений
};
```

ChildView.cpp

```
#include "stdafx.h"
#include "Game.h"
#include "ChildView.h"

CChildView::CChildView()   // Конструктор по-умолчанию
{
}

CChildView::~CChildView()  // Деструктор
{
}

BEGIN_MESSAGE_MAP(CChildView, CWnd)      // Начало объявления перехватываемых сообщений
	ON_WM_PAINT()               // Сообщение рисования вида
	ON_WM_ERASEBKGND()          // Сообщение необходимости очистки фона перед
рисованием
	ON_WM_LBUTTONDBLCLK()       // Сообщение двойного клика левой кнопки мыши
	ON_WM_MOUSEMOVE()           // Сообщение передвижения мыши в области вида
	ON_WM_LBUTTONDOWN()         // Сообщение нажатия левой кнопки мыши
	ON_WM_RBUTTONUP()           // Сообщение отпускания правой кнопки мыши
	ON_WM_NCMOUSEMOVE()         // Сообщения передвижения мыши вне области вида
	ON_WM_KEYDOWN()                   // Сообщение нажатия на клавишу
END_MESSAGE_MAP()                   // Конец объявления перехватываемых сообщений

void CChildView::OnNcMouseMove(UINT hitTest, CPoint point)   // Метод перехвата
передвижения мыши вне области вида
{
	theApp.NcMouseMove();       // Передача управление объекту игровой логики
}

BOOL CChildView::PreCreateWindow(CREATESTRUCT& cs)     // Метод подготовки к созданию окна
{
	if (!CWnd::PreCreateWindow(cs))   // Если у родительсткого класса не получилось
подготовить окно к созданию
		return FALSE; // То создавать нечего

	cs.dwExStyle |= WS_EX_CLIENTEDGE; // Окно имеет рамку с притопленными краями
	cs.style &= ~WS_BORDER;     // Окно имеет бордюр
	cs.lpszClass = AfxRegisterWndClass(CS_HREDRAW | CS_VREDRAW | CS_DBLCLKS,
::LoadCursor(NULL, IDC_ARROW), reinterpret_cast<HBRUSH>(COLOR_WINDOW + 1), NULL); //
Хотим обычное человеческое окно

	return TRUE;  // Всё удалось, можно создавать окно
}

BOOL CChildView::OnEraseBkgnd(CDC* pDC)  // Метод определения необходимости очистки фона
перед рисованием вида
{
	return FALSE; // Очищать фон не трубуется
}

void CChildView::OnKeyDown(UINT nChar, UINT nRepCnt, UINT nFlags)   // Метод нажатия на
клавишу
{
	theApp.KeyDown(nChar, nRepCnt, nFlags);  // Передаём бразды правления объекту
игровой логики
}

void CChildView::OnLButtonDblClk(UINT nFlags, CPoint point)  // Метод перехвата двойного
клика левой кнопки мыши
{
```

```
	theApp.LeftButtonDoubleClick(nFlags, point);	// Передаём управление объекту
игровой логики
}

void CChildView::OnMouseMove(UINT nFlags, CPoint point)		// Метод перехвата движения
мыши в области вида
{
	theApp.MouseMove(nFlags, point);  // Передаём управление объекту игровой логики
}

void CChildView::OnLButtonDown(UINT nFlags, CPoint point)	// Метод перехвата нажатия
левой кнопки мыши
{
	theApp.LeftButtonDown(nFlags, point);	// Передаём управление объекту игровой
логики
}

void CChildView::OnRButtonUp(UINT nFlags, CPoint point)		// Метод перехвата
отпускания правой кнопки мыши
{
	theApp.RightButtonUp(nFlags, point);	// Передаём управление объекту игровой
логики
}

void CChildView::OnPaint() // Метод рисования в виде
{
	CPaintDC dc(this);  // Получаем текущий контекст рисования
	CMemoryDC pDC(&dc); // Хотим двойную буфферизацию, чтобы не мерцало
	CRect rect;  // Прямоугольник

	GetClientRect(&rect);		// Получаем размеры вида
	theApp.RenderField(pDC, rect);	// Передаём управление объекту игровой логики
для отрисовки всего и вся
}
```

Resource.h

```
#define IDD_ABOUTBOX                    100
#define IDR_MAINFRAME                   128
#define IDR_GameTYPE                    130
#define IDB_GRASS                       313
#define IDB_CLOCK                       315
#define IDB_HAND                        316
#define ID_NEW_GAME                     32774
#define ID_OPEN_GAME                    32775
#define ID_SAVE_GAME                    32776
#define ID_EVOLVE_1_SECOND              32782
#define ID_EVOLVE_3_SECONDS             32783
#define ID_EVOLVE_10_SECONDS            32784
#define ID_EVOLVE_42_MINUTES            32785
#define ID_EVOLVE_HAND                  32786

#ifdef APSTUDIO_INVOKED
#ifndef APSTUDIO_READONLY_SYMBOLS
#define _APS_NEXT_RESOURCE_VALUE        317
#define _APS_NEXT_COMMAND_VALUE         32787
#define _APS_NEXT_CONTROL_VALUE         1000
#define _APS_NEXT_SYMED_VALUE           310
#endif
#endif
```

Clock.h

```
#pragma once

#define _USE_MATH_DEFINES

#include "stdafx.h"
#include "MemoryDC.h"

/* Класс индикации режима вызова эволюции клеток */
class CClock
{
public:
	CClock();	// Конструктор по-умолчанию
	~CClock();	// Деструктор

	void SetTime(int);		// Метод установки интервала между вызовами эволюции
клеток
	void Render(CMemoryDC &, CPoint); // Метод вывода на экран индикатора
	void Tick();			// Метод обновления состояния
	int GetHeight();		// Метод получения высоты изображения-индикатора
	int GetWidth();			// Метод получения ширины изображения-индикатора

protected:
	static const int m_height = 49;	// Высота изображения-индикатора
	static const int m_width = 49;	// Ширина изображения-индикатора

	int m_tick;		// Счётчик количества тиков до с момента последней эволюции
	int m_time;		// Интервал между вызовами эволюции клеток
};
```

Clock.cpp

```
#include "stdafx.h"
#include "Clock.h"
#include "MemoryDC.h"
#include "resource.h"
#include <cmath>

CClock::CClock()    // Конструктор по-умолчанию
{
	m_time = 0;   // Ручной режим вызова эволюции
}

CClock::~CClock()   // Деструктор
{
}

void CClock::Render(CMemoryDC &dc, CPoint position)   // Метод вывода на экран индикатора
{
	CBitmap bmp;  // Изображение индикатора
	CDC dcMemory; // Контекст рисования в памяти

	bmp.LoadBitmap(m_time ? IDB_CLOCK : IDB_HAND);  // В зависимости от режима вызова
эволюций загружается либо циферблат, либо пингвин
	dcMemory.CreateCompatibleDC(dc);  // Создание совместимого контекста рисования с
имеющимся
	dcMemory.SelectObject(&bmp);      // Установка изображения-индикатора
	dc->TransparentBlt(position.x, position.y, m_width, m_height, &dcMemory, 0, 0,
m_width, m_height, RGB(0, 0, 0)); // Вывод в общий контекст рисования изображения с
учётом прозрачного цвета

	if (m_time)   // Если режим вызова эволюций автоматический
	{
		CPen pen(PS_SOLID, 3, RGB(255, 255, 255));      // Карандаш для рисования
стрелки часов

		dc->SelectObject(&pen);    // Выбираем карандаш
		dc->MoveTo(position + CPoint(22, 24));   // Переходим в центр циферблата

		double x = (double)position.x + 22.0 + 6.0 * sin((double)m_tick * M_PI /
(double)m_time * 2.0);     // Получаем положение конца стрелки по оси абсцисс
		double y = (double)position.y + 24.0 - 6.0 * cos((double)m_tick * M_PI /
(double)m_time * 2.0);     // Получаем положение конца стрелки по оси ординат

		dc->LineTo((int)x, (int)y);// Рисуем стрелку из центра циферблата
	}
}

void CClock::Tick()  // Метод обновления состояния
{
	++m_tick;     // Увеличиваем счетчик количества тиков с последней эволюции
}

void CClock::SetTime(int time)    // Метод установки интервала между вызовами эволюции
{
	m_time = time;       // Запоминаем интервал
	m_tick = 0;          // Эволюции пока быть не должно
}

int CClock::GetHeight()    // Метод получения высоты изображения-индикатора
{
	return m_height;     // Возвращаем высоту
}
```

```
int CClock::GetWidth()      // Метод получения ширины изображения-индикатора
{
	return m_width;             // Возвращаем ширину
}
```

MemoryDC.h

```
#pragma once

/* Класс для реализация двойной буферизации, наследуется от класса от контекста рисования
*/
class CMemoryDC : public CDC
{
public:
	CMemoryDC(CDC *, const CRect *pRect = NULL);	// Конструктор
	~CMemoryDC();					// Конструктор по-умолчанию
	CMemoryDC* operator->();	// Оператор получение доступа к методам класса
	operator CMemoryDC*();			// Оператор разыменования

private:
	CBitmap m_bitmap;		// Текущий экран
	CBitmap *m_oldBitmap;	// Старый экран
	CDC *m_pDC;					// Контекст рисования первоначальный
	CRect m_rect;			// Область рисования
	BOOL m_bMemDC;				// Рисуем ли сейчас в памяти?
};
```

MemoryDC.cpp

```
#include "stdafx.h"
#include "MemoryDC.h"

CMemoryDC::CMemoryDC(CDC *pDC, const CRect *pRect) :  // Конструктор
	CDC()  // Вызываем конструктор родительского класса
{
	m_pDC = pDC;  // Копируем указатель на контекст рисования
	m_oldBitmap = NULL;  // Старое изображение области рисования
	m_bMemDC = !pDC->IsPrinting();    // Был ли контекст использован для рисования

	if (pRect == NULL)  // Если область рисования не указана
		pDC->GetClipBox(&m_rect);  // То будем рисовать везде
	else
		m_rect = *pRect;    // Иначе только там, где надо

	if (m_bMemDC) // Если уже рисовали на экране
	{
		CreateCompatibleDC(pDC);   // Создаем совместимый контекст рисования
		pDC->LPtoDP(&m_rect);      // Получаем размеры
		m_bitmap.CreateCompatibleBitmap(pDC, m_rect.Width(), m_rect.Height());
	// Создаем копию экрана
		m_oldBitmap = SelectObject(&m_bitmap);   // Ставим старую копию и запоминаем
то, что было
		SetMapMode(pDC->GetMapMode());    // Настройки рисования переносим
		SetWindowExt(pDC->GetWindowExt());       // Переносим настройки рисования
		SetViewportExt(pDC->GetViewportExt());   // Настройки переносим рисования
		pDC->DPtoLP(&m_rect);      // Получаем размер
		SetWindowOrg(m_rect.left, m_rect.top);   // Сдвиг окна
	}
	else   // Если на экране не рисовали
	{
		m_bPrinting = pDC->m_bPrinting;   // Рисовали ли раньше?
		m_hDC = pDC->m_hDC; // Копируем контекст рисоавния
		m_hAttribDC = pDC->m_hAttribDC;   // Копируем аттрибуты рисования
	}

	FillSolidRect(m_rect, pDC->GetBkColor());       // Заливаем всё фоновым цветом
}

CMemoryDC::~CMemoryDC()    // Деструктор
{
	if (m_bMemDC) // Если рисовали до этого что-то
	{
		m_pDC->BitBlt(m_rect.left, m_rect.top, m_rect.Width(), m_rect.Height(),
this, m_rect.left, m_rect.top, SRCCOPY); // Выводим в контекст рисования то, что
нарисовали в памяти
		SelectObject(m_oldBitmap); // Отдаём обратно старое изображение
	}
	else   // Если не рисовали до этого что-нибудь
		m_hDC = m_hAttribDC = NULL;// Нечего выводить
}

CMemoryDC* CMemoryDC::operator->()       // Вызов методов
{
	return this;
}

CMemoryDC::operator CMemoryDC*()  // Оператор разыменование указателя
{
	return this;
}
```

Protoblast.h

```
#pragma once

#include "stdafx.h"
#include "ProtoblastEffect.h"

/* Класс клетки игрового поля */
class CProtoblast
{
public:
	CProtoblast();		// Конструктор по-умолчанию
	~CProtoblast();		// Деструктор

	CSize GetSize();	// Метод получения размера клетки
	void Render(CMemoryDC &, CPoint); // Метод отрисовки клетки
	void Tick();		// Метод обновления состояния
	void Click(bool isLeftButton = true);	// Метод клика по клетке кнопкой мыши
	void Hover();		// Метод выделения клетки
	void Unhover();			// Метод унвыделения клетки
	bool IsLive();			// Метод определения является ли клетка живой?
	int GetWidth();			// Метод получения ширины клетки
	int GetHeight();	// Метод получения высоты клетки


private:
	CProtoblastEffect *m_effect;	// Рисование клетки
	CSize m_frame;					// Размер клетки
	CSize m_area;				// Размер внутренней области

	void SetStatus(EStatus);	// Метод установки состояния клетки
};
```

Protoblast.cpp

```
#include "stdafx.h"
#include "Protoblast.h"
#include "ProtoblastEffect.h"

CProtoblast::CProtoblast() // Конструктор по-умолчанию
{
	m_frame = CSize(24, 24);	// Размеры клетки
	m_area = m_frame - CSize(2, 2);	// Размеры внутренней области клетки
	m_effect = NULL;	// Текущий вид

	SetStatus(STATUS_DISABLE); // Клетка неактивна
}

CProtoblast::~CProtoblast()// Деструктор
{
	if (m_effect) // Если клетку рисовали
		delete m_effect;	// То больше рисовать её не стоит
}

void CProtoblast::Render(CMemoryDC &dc, CPoint position)	// Метод отрисовки клетки
{
	CPen pen(PS_SOLID, 1, RGB(0, 0, 0));	// Карандаш рисования
	CPen *pLastPen = dc.SelectObject(&pen);  // Устанавливаем карандаш

	dc.Rectangle(position.x, position.y, position.x + m_frame.cx, position.y +
m_frame.cy);  // Рисуем прямоугольник
	dc.SelectObject(pLastPen); // Возвращаем прошлый караншад
	m_effect->Render(dc, position + CPoint(1, 1));  // Отрисовываем внутренню область
}

void CProtoblast::Tick()	// Метод обновления состояния
{
	m_effect->Tick();	// Передаём управление объекту рисования
}

void CProtoblast::Click(bool isLeftButton)	// Метод клика по клетке кнопкой мыши
{
	if (isLeftButton)	// Если это была левая кнопка мыши
		SetStatus(STATUS_ACTIVE);  // То клетка оживает
	else	// Иначе
		SetStatus(STATUS_DISABLE); // Умираем :(
}

void CProtoblast::Hover()  // Метод выедления клетки
{
	SetStatus(STATUS_HOVER);	// Выделяем клетку
}


void CProtoblast::Unhover()// Метод унвыделения клетки
{
	SetStatus(m_effect->GetLastStatus());	// Клетка принимает предпоследнее
состояние
}

void CProtoblast::SetStatus(EStatus status)	// Метод установки состояния клетки
{
	if (status == STATUS_NONE) // Если нечего устанавливать
		return;		// То ничего делать не надо

	EStatus currStatus = m_effect ? m_effect->GetStatus() : STATUS_NONE;	//
Получаем текущее состояние клетки
```

```
	if (status == currStatus)  // если текущее и новое состояния совпадают
		return;		// Больше ничего делать не надо

	if (status == STATUS_HOVER && currStatus != STATUS_DISABLE)  // Если хотим
выделить клетку и она не является неживой
		return;		// То так дело не пойдет

	CProtoblastEffect *temp = m_effect;	// Текущий эффект рисования

	switch (status)		// В зависимости от состояния
	{
	case STATUS_ACTIVE:  // Оживляем клетку
		m_effect = new CProtoblastEffectActive(m_area, temp);  // Эффект живой
клетки
		break;

	case STATUS_HOVER:   // Выделяем
		m_effect = new CProtoblastEffectHover(m_area, temp);   // Выделяем
		break;

	default:	// Иначе наверное она неживая
		m_effect = new CProtoblastEffectDisable(m_area, temp);// Мертвая клетка
	}

	delete temp;  // Удаляем старый эффект рисования
}

int CProtoblast::GetWidth()// Получаем ширину клетки
{
	return m_frame.cx;   // Ширина
}

int CProtoblast::GetHeight()	// Получаем высоту клетки
{
	return m_frame.cy;   // Высота
}

CSize CProtoblast::GetSize()	// Получаем размер клетки
{
	return m_frame;		// Размер!
}

bool CProtoblast::IsLive() // Является ли клетка живой?
{
	return (m_effect->GetStatus() == STATUS_ACTIVE);	// Если живая, то клетка
живая 146%
}
```

ProtoblastEffect.h

```
#pragma once

#include "stdafx.h"
#include "MemoryDC.h"

enum EStatus {STATUS_NONE, STATUS_DISABLE, STATUS_ACTIVE, STATUS_HOVER};   // Перечисление
состояний клетки

/* Абстрактный класс эффекта рисования клетки */
class CProtoblastEffect
{
public:
	CProtoblastEffect(CSize area);	// Конструктор
	virtual ~CProtoblastEffect();	// Виртуальный деструктор

	COLORREF GetCurrColor();	// Метод получения текущего цвета клетки

	virtual void Tick() = 0;	// Метод тика
	virtual void Render(CMemoryDC &, CPoint);	// Метод отрисовки
	virtual EStatus GetStatus() = 0;	// Метод получения статуса
	virtual EStatus GetLastStatus();	// Метод получения предыдущего статуса

protected:
	COLORREF m_currColor;	// Текущий цвет
	CSize m_area;		// Размер

	void ChangeColor(COLORREF, int, int, int);	// Метод изменения текущего цвета в
сторону заданного цвета пошагово для каждой компоненты
	byte MakeChangeColorStep (int, int, int);	// Метод изменения цветовой
компоненты до заданной с заданным шагом
};

/* Класс эффекта мертвой клетки, наследованный от класса эффекта рисования клетки */
class CProtoblastEffectDisable : public CProtoblastEffect
{
public:
	CProtoblastEffectDisable(CSize, CProtoblastEffect *); // Конструктор
	~CProtoblastEffectDisable();	// Деструктор

	void Tick();		// Метод тика
	EStatus GetStatus();// Метод получения статуса

protected:
	static const COLORREF m_endColor = RGB(255, 255, 255);// Конечный цвет (белый)
};

/* Класс эффекта живой клетки, наследованный от класса эффекта рисования клетки */
class CProtoblastEffectActive : public CProtoblastEffect
{
public:
	CProtoblastEffectActive(CSize, CProtoblastEffect *);  // Конструктор
	~CProtoblastEffectActive();// Деструктор

	void Tick();		// Метод тика
	EStatus GetStatus();// Метод получения статуса

protected:
	static const COLORREF m_endColor = RGB(87, 141, 249); // Конечный цвет
(голубовастый ближе к синему)
};

/* Класс эффекта выделенной клетки, наследованный от класса эффекта рисования клетки */
```

```
class CProtoblastEffectHover : public CProtoblastEffect
{
public:
	CProtoblastEffectHover(CSize, CProtoblastEffect *last);	// Конструктор
	~CProtoblastEffectHover(); // Деструктор

	void Tick();  // Метод тика
	void Render(CMemoryDC &, CPoint); // Метод рисования
	EStatus GetStatus();	// Метод получения статус
	EStatus GetLastStatus();	// Метод получения последнего статуса

protected:
	COLORREF m_colors[4];	// Промежуточные цвета
	int m_color;		// Индекс текущего выбранного промежуточного цвета
	int m_tickCount;	// Счетчик с начала анимации
	EStatus m_lastStatus;	// Предпоследний статус
};
```

Protoblast.cpp

```
#include "stdafx.h"
#include "ProtoblastEffect.h"
#include <algorithm>

CProtoblastEffect::CProtoblastEffect(CSize area) :    // Конструктор
	m_area(area)  // Вызываем родительский конструктор
{
}

CProtoblastEffect::~CProtoblastEffect()  // Деструктор
{
}

COLORREF CProtoblastEffect::GetCurrColor()     // Метод получения текущего цвета
{
	return m_currColor;  // Текущий цвет
}

void CProtoblastEffect::ChangeColor(COLORREF endColor, int stepR, int stepG, int stepB)
	// Метод изменения текущего цвета в сторону заданного цвета пошагово для каждой
компоненты
{
	if (m_currColor == endColor)      // Если достигли цели
		return;       // То делать нечего

	int r = MakeChangeColorStep(GetRValue(m_currColor), GetRValue(endColor), stepR);
	// Иначе для каждой компоненты приближаем цель
	int g = MakeChangeColorStep(GetGValue(m_currColor), GetGValue(endColor), stepG);
	// Приближаем цель для каждой компоненты
	int b = MakeChangeColorStep(GetBValue(m_currColor), GetBValue(endColor), stepB);
	// Приближаем...

	m_currColor = RGB(r, g, b);// Получаем текущий цвет
}

byte CProtoblastEffect::MakeChangeColorStep(int chCurr, int chEnd, int step)     // Метод
изменения цветовой компоненты до заданной с заданным шагом
{
	if (step < 0) // Если шаг отрицательный
		step *= -1;   // То делаем его положительным

	if (chCurr == chEnd) // Если компоненты похожи
		return chCurr;       // То эффект достигнут

	if (chCurr > chEnd)  // Если надо назад уменьшить компоненту
		chCurr = max(chCurr - step, chEnd);      // То так и делаем
	else
		chCurr = min(chCurr + step, chEnd);      // Иначе надо немного вперед

	return chCurr;       // Что получилось, то и возвращаем
}

void CProtoblastEffect::Render(CMemoryDC &dc, CPoint position)      // Метод отрисовки
{
	dc.FillSolidRect(position.x, position.y, m_area.cx, m_area.cy, m_currColor);
	// Заливаем внутреннюю область текущим цветом
}

EStatus CProtoblastEffect::GetLastStatus()     // Получаем последний статус
{
	return STATUS_NONE;  // Последнее здесь не нужно
}
```

```
CProtoblastEffectDisable::CProtoblastEffectDisable(CSize area, CProtoblastEffect *last) :
	// Конструктор
	CProtoblastEffect(area)	// Конструктор родительского класса
{
	m_currColor = last ? last->GetCurrColor() : m_endColor;	// Если у прошлого
эффекта есть текущий цвет, то он и для нас тоже текущий
}

CProtoblastEffectDisable::~CProtoblastEffectDisable() // Деструктор
{
}

void CProtoblastEffectDisable::Tick()	// Метод тика
{
	ChangeColor(m_endColor, 20, 20, 20);	// Приближаем текущий цвет к конечнему 20-
мильными шагами
}

EStatus CProtoblastEffectDisable::GetStatus()	// Метод получения статуса
{
	return STATUS_DISABLE;	// Статус мертвой клетки
}

CProtoblastEffectActive::CProtoblastEffectActive(CSize area, CProtoblastEffect *last) :
	// Конструктор
	CProtoblastEffect(area)	// Конструктор родительского класса
{
	m_currColor = last ? last->GetCurrColor() : m_endColor;	// Если была прошлая
клетка, то забираем её текущий цвет себе
}

CProtoblastEffectActive::~CProtoblastEffectActive()	// Деструктор
{
}

EStatus CProtoblastEffectActive::GetStatus()	// Метод полученрия статуса
{
	return STATUS_ACTIVE;	// Статус живой клетки
}

void CProtoblastEffectActive::Tick()	// Метод тика
{
	ChangeColor(m_endColor, 20, 20, 20);	// Приближаем текущий цвет к цели
}

CProtoblastEffectHover::CProtoblastEffectHover(CSize area, CProtoblastEffect *last) :
	// Конструктор
	CProtoblastEffect(area)	// Конструктор родительского класса
{
	m_lastStatus = last->GetStatus(); // Получаем статус прошлого эффекта
	m_color = 0;  // Индекс текущего цвета
	m_colors[0] = RGB(255, 255, 153); // Ярко-желтый
	m_colors[1] = RGB(255, 255, 102); // Желтый
	m_colors[2] = RGB(255, 255, 51);  // Не ярко-желтый
	m_colors[3] = m_colors[1]; // Желтый
	m_currColor = last->GetCurrColor();	// Получаем текущий цвет, чтобы потом его
вернуть следующему эффекту
}

CProtoblastEffectHover::~CProtoblastEffectHover()	// Деструктор
{
}

void CProtoblastEffectHover::Tick()	// Метод тика
```

```
{
	++m_tickCount;	// Тикаем

	if (m_tickCount % 10 == 0) // Если пора менять цвет
	{
		m_tickCount = 0;	// То тикаем заново
		m_color = (m_color + 1) % 4;	// Меняем цвет на следующий
	}
}

void CProtoblastEffectHover::Render(CMemoryDC &dc, CPoint position) // Метод рисования
{
	dc.FillSolidRect(position.x, position.y, m_area.cx, m_area.cy, m_colors[m_color]);
	// Рисуем внутреннюю область цвета, на который указывает индекс цвета
}

EStatus CProtoblastEffectHover::GetStatus()	// Метод получения статуса
{
	return STATUS_HOVER;// Статус выделенной клетки
}

EStatus CProtoblastEffectHover::GetLastStatus()// Метод получения статуса прошлого
эффекта
{
	return m_lastStatus;// Прошлый статус
}
```

AboutDialog.h

```
#pragma once

#include "stdafx.h"
#include "afxdialogex.h"
#include "Game.h"

/* Класс диалога "О программе", наследуемый от класса диалогов с поддеркой передачи
данных */

class CAboutDlg : public CDialogEx
{
public:
	enum {IDD = IDD_ABOUTBOX}; // Идентификатор диалога

	CAboutDlg();  // Конструктор по-умолчанию

private:
	DECLARE_MESSAGE_MAP() // Макрос конца класса для объявления перехватываемых
сообщений
};
```

AboutDialog.cpp

```
#include "stdafx.h"
#include "AboutDialog.h"

BEGIN_MESSAGE_MAP(CAboutDlg, CDialogEx)  // Начало объявления перехватываемых сообщений
END_MESSAGE_MAP()   // Конец объявления перехватываемых сообщений

CAboutDlg::CAboutDlg() :          // Конструктор по-умолчанию
       CDialogEx(CAboutDlg::IDD)  // Вызов конструктора для родительского класса
{
}
```