Министерство образования и науки Российской Федерации

1. Санкт-Петербургский Политехнический Университет Петра Великого
2. —
3. Институт кибербезопасности и защиты информации

**КУРСОВАЯ РАБОТА**

**Оптимизация Игры «Жизнь»**

по дисциплине «Языки программирования»

Выполнили Дударь А.Я.

студенты гр. 4831001/10002 Карамышев Е. С.

Преподаватель Малышев Е. В.

«\_\_\_» \_\_\_\_\_\_\_\_\_\_ 2023 г.

Санкт-Петербург

2023

**Содержание**

[Введение 3](#_Toc124880279)

[Глава 1 4](#_Toc124880280)

[1.1. Профилирование и анализ исходной программы 4](#_Toc124880281)

[1.2. Возможные варианты оптимизации 6](#_Toc124880282)

[Глава 2 8](#_Toc124880283)

[2.1. Реализация алгоритмической оптимизации 8](#_Toc124880284)

[2.2. Машинно-независимая оптимизация 12](#_Toc124880285)

[2.3. Машинно-зависимая оптимизация 15](#_Toc124880286)

[2.4. Оценка выполненных оптимизаций 18](#_Toc124880287)

[Заключение 19](#_Toc124880288)

[Приложение А. Полный код оптимизированной программы 20](#_Toc124880289)

# **Введение**

**Цель работы:** оптимизация курсовой работы по разработке Игры «Жизнь», написанной на 1-м курсе

**Задачи:**

1. Изучить основы профилирования программ с помощью встроенных механизмов Microsoft Visual Studio и зафиксировать показатели исходной программы
2. Оценить алгоритмическую сложность исходной программы и рассмотреть возможные модификации алгоритма
3. Выполнить алгоритмическую, машинно-независимые и машинно-зависимые оптимизации
4. Выполнить профилирование после произведённых оптимизаций и проанализировать эффективность внесённых изменений

В Главе 1 данной работы разобраны результаты профилирования исходной программы и рассмотрены возможные варианты оптимизаций. Глава 2 посвящена внедрению выбранных вариантов оптимизаций и анализу производительности новой программы.

# **Глава 1**

* 1. **Профилирование и анализ исходной программы**

В Visual Studio 2022 есть несколько вариантов выполнения профилирования программы. Первым является инструментирование, которое необходимо для точного определения числа и времени вызовов. Данный режим позволяет выбирать временной промежуток анализа и имеет набор различных представлений результатов такие как «Сводка», «Функции», «Дерево вызовов» и др. Будем использовать этот метод для оценки алгоритмической оптимизации, а также для сравнения исходной и финальной версий. Вторым способом является просмотр использования ЦП, GPU и памяти. Он также позволяет оценить время выполнения той или иной программы или функции, однако дополнительно показывает время, затраченное на выполнение команд или блоков команд в самом коде. Поэтому его будем применять для профилирования алгоритмической, машинно-независимой и машинно-зависимой оптимизаций.

Так как программа может работать в трёх режимах: автоматическая игра (случайное расположение изначального положения), пользовательская игра (пользователь мышкой задаёт изначально положение), примеры (заданные заранее устойчивые фигуры), необходимо провести профилирование каждого по отдельности. Для первого и третьего режимов будет выбираться промежуток в первые 10 секунд работы. По второму режиму профилирование проводиться не будет, так как достаточно сложно создавать одни и те же изначальные расстановки.

Результаты профилирование первого режима работы представлены на рис. 1 и рис. 2.

Изображение выглядит как стол

Автоматически созданное описание

Рисунок

Изображение выглядит как стол

Автоматически созданное описание

Рисунок

Результаты профилирование третьего режима работы представлены на рис. 1 и рис. 2.

Изображение выглядит как стол

Автоматически созданное описание

Рисунок

Изображение выглядит как стол

Автоматически созданное описание

Рисунок

Из полученных данных можем сделать вывод о том, что больше всего времени ЦП уходить на функцию Game\_Rule, которая отвечает за выполнение правил игры для каждой клетки, а также функций Auto\_Draw и User\_Draw, которые отвечают за прорисовку поля и вызов функции Game\_Rule.

Алгоритмическая сложность исходной реализации функций Auto\_Draw и User\_Draw O(n2). Поэтому было принято решение оптимизировать данные функции.

* 1. **Возможные варианты оптимизации**

Алгоритмическую оптимизацию будем проводить для первых двух режимов игры.

Следует отказаться от нескольких вещей:

1. Хранение клеток в двумерном массиве в пользу одномерного, чтобы клетки хранились в памяти последовательно и непрерывно
2. Использование вспомогательного двумерного массива field2 и функции смены массивов field1 и field2.
3. Использование функции Game\_Rule

Следует реализовать:

1. Настроить функции, которые обрабатывают двумерные массивы field1 и field2, на обработку одномерного массива А
2. Изменить в функциях Auto\_Draw и User\_Draw расчёт следующего поколения, путём создания временного одномерного массива, в который будут заноситься, только «перспективные» клетки, то есть те, которые могут быть живыми в следующем положении. При этом массив А будет заполняться нулями с помощью функции memset, а живые клетки будут заноситься из временного массива.

В качестве машинно-независимых оптимизаций планируется изменить функцию Game\_Rule, произвести различные оптимизации циклов рисовки и инициализации игры. Машинно-зависимые оптимизации, включая ассемблерные вставки, будут выбраны в ходе более детального просмотра кода.

# **Глава 2**

* 1. **Реализация алгоритмической оптимизации**

Для реализации нового алгоритма изменим функцию инициализации новой игры Auto\_New\_Game. На рис. 5 и рис. 6 представлена старая и новая реализация данной функции. Аналогичные преобразования проведены с функцией User\_New\_Game.

Изображение выглядит как текст

Автоматически созданное описание

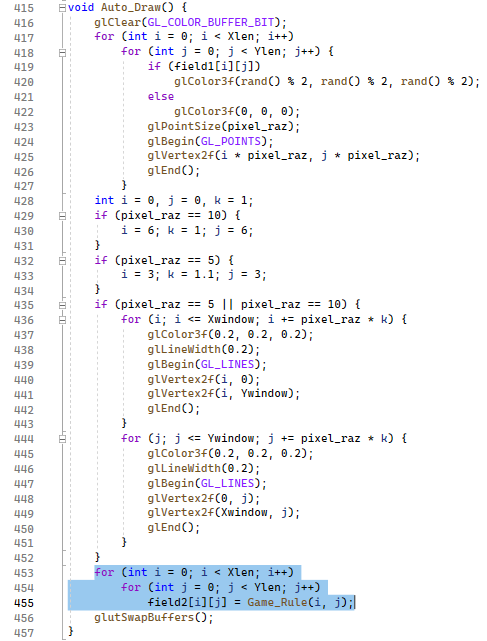
Рисунок

Изображение выглядит как текст

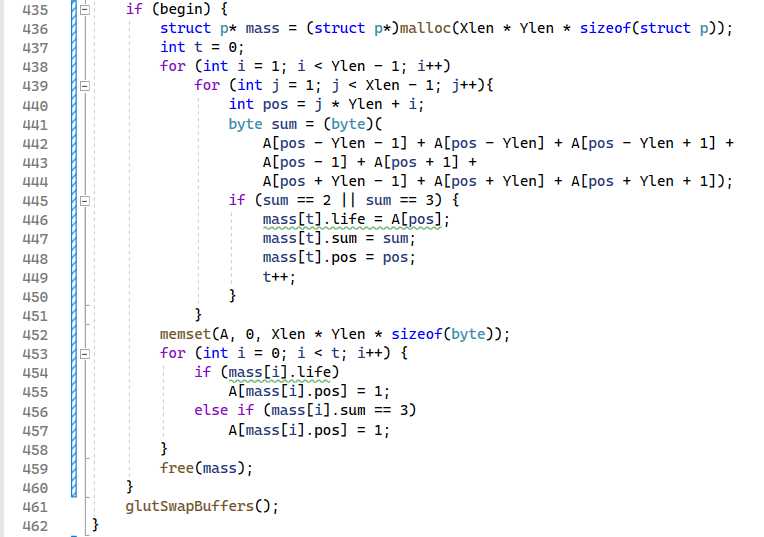
Автоматически созданное описание

Рисунок

Так как функции Auto\_Draw и User\_Draw выполняют одни и те же действия и имеют большой общий участок кода заменим их на функцию Draw\_B, в которой применим ранее описанные оптимизации алгоритма подсчёта следующего поколения. На рис. 7 представлена исходная реализация функции Auto\_Draw, выделенный код которой, был заменён на фрагмент из рис. 7.



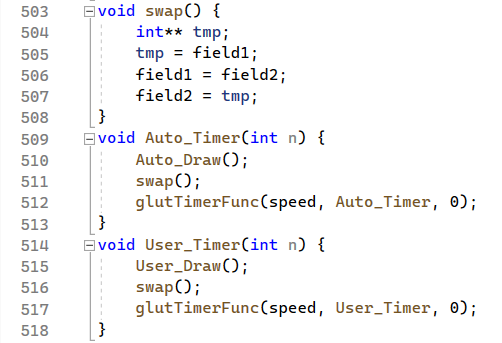
Рисунок



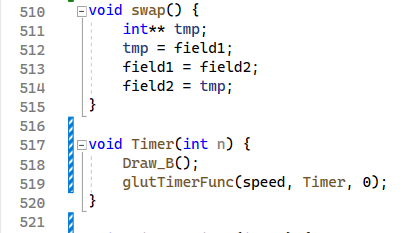
Рисунок

Новый алгоритм работает по следующему принципу. Создаётся массив структур, состоящей из полей byte life (значение 0 или 1), char sum (количество соседей), int pos (позиция в массиве А). В цикле считаем количество соседей для каждой клетки и, если это значение является «перспективным», то есть может повлечь за собой продолжение или рождение жизни, то заносим данные этой клетки в массив mass. Далее с помощью memset обнуляется массив А и заполняем в массиве А только живые клетки, взятые из массива mass.

Теперь в функциях таймера Auto\_Timer и User\_Timer нужно убрать вызов функции swap, и вместо двух функций получим функцию Timer. Функцию swap необходимо оставить в коде, так как с её помощью работает режим «Примеры». Исходная и модифицированная реализации данного фрагмента приведены на рис. 9 и рис. 10 соответственно.



Рисунок



Рисунок

Рассмотрим результаты профилирования на рис. 11 и рис. 12. Из результатов инструментирования можем сделать вывод, что среднее время, затраченное на выполнение эквивалентных функций Auto\_Draw и Draw\_B и их потомков уменьшилось на 15,76 мс, что быстрее примерно в 1,5 раза. Функция инициализации теперь работает лучше на 15%.

Исходя из данных об использовании процессора видим, что модуль game\_of\_life, который включает в себя все модули (функции) программы, стал работать на 133 мс быстрее, то есть произошло ускорение примерно в 1,92 раза.

Изображение выглядит как стол

Автоматически созданное описание

Рисунок

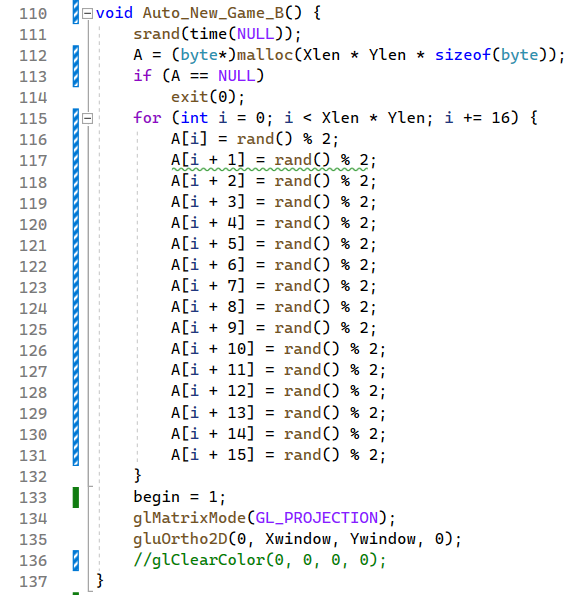
Изображение выглядит как стол

Автоматически созданное описание

Рисунок

* 1. **Машинно-независимая оптимизация**

Основные машинно-независимые оптимизации проводились для режима работы программы с примерами, так как в нём запускаются устойчивые фигуры, которые изменяются во времени одинаково, однако дополнительно была выполнена развёрстка цикла и удаление лишнего кода в функции Auto\_New\_Game\_B (рис. 13).



Рисунок

Больше всего подверглась оптимизации функция Figure\_Draw, которая пришла на смену функции Auto\_Draw для рисования третьего режима работы. Во-первых, произошёл вынос команд, отвечающих за настройку размера точек, цвета и ширины линий, которые ранее выполнялся на всех итерациях цикла, а теперь только один раз до цикла (рис. 14 и рис. 15). Этот вид оптимизаций был выполнен в функциях Figure\_Draw и Draw\_B.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок

Изображение выглядит как текст

Автоматически созданное описание

Рисунок

Далее была произведена оптимизация вычисления следующего поколения для третьего режима работы путём изменения функции Game\_Rule и её вызова в Figure\_Draw. Теперь цикл проходит по массиву, не затрагивая крайние элементы массива, поэтому в функции Game\_Rule стало возможным отказаться от проверки этих граничных условий, тем самым уменьшили количества итераций цикла и выполнили оптимизацию переходов путём отказа от них. Была выполнена развёрстка цикла с вызовом функции Game\_Rule для получения линейного участка кода. В функции Game\_Rule отказались от цикла для подсчёта соседей клетки, а также оптимизировали переходы: теперь наиболее вероятное условие стоит выше, также была попытка отказаться от переходов при возврате функции, однако дизассемблированный код показал, что этого не произошло. На рис. 16 и рис. 17 представлены изначальная и оптимизированная функции Game\_Rule, а на рис. 18 приведена развёрстка цикла в функции Figure\_Draw.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок

Изображение выглядит как текст

Автоматически созданное описание

Рисунок

Изображение выглядит как текст

Автоматически созданное описание

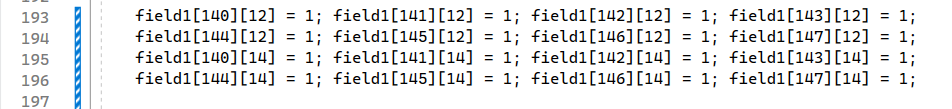
Рисунок

Также в функции Figure\_New\_Game, в которой происходит заполнение массива в начале игры возможно сделать отказ от цикла или произвести его развёрстку (рис. 19 и рис. 20).

Изображение выглядит как текст, оранжевый

Автоматически созданное описание

Рисунок



Рисунок

Так как основные машинно-независимые оптимизации были проведены именно для третьего режима игры и при этом алгоритмическая оптимизация его не затронула, появляется возможность её точной оценки. Результаты профилирования методом инструментирования приведены на рис. 21, данные по использованию ЦП показаны на рис. 22.

Сравнивая их с изначальными данными (рис. 3 и рис. 4) можем сравнить время выполнения изначальной и оптимизированный реализации функции Game\_Rule, а также функций Auto\_Draw и Figure\_Draw, которые выполняют одну и ту же функцию. В старой реализации среднее затраченное время на выполнение модуля game\_of\_life и его потомков составляло 277 мс, в модифицированной ­– 157 мс. Функция Auto\_Draw ранее выполнялась 48,02 мс, теперь 34,6 мс, а собственное время выполнения функции Game\_Rule уменьшилось с 177 мс до 73 мс.

Изображение выглядит как стол

Автоматически созданное описание

Рисунок

Изображение выглядит как стол

Автоматически созданное описание

Рисунок

* 1. **Машинно-зависимая оптимизация**

В качестве машинно-зависимой оптимизации в нашей реализации представляется возможным сделать ассемблерные вставки, обнулять переменные, учитывая разрядность, а также применить арифметику указателей.

Замена обнуления переменной была заменена на более быструю операцию – побитовый сдвиг (рис. 23)

Text, letter

Description automatically generated

Рисунок

Замена индексация же была смещена адресацией с помощью указателей (рис. 24)

Text

Description automatically generated

Рисунок

Также преимущества адресации с помощью указателей были применены в функции Menu, Choice и Button\_Color. Приведён пример на рис. 25**.** Массивы user\_set\_yell и user\_set\_green отвечают за окраску кнопок в жёлтый и зелёный цвета, создаются указатели на данные массивы, которые используются для отказа от индексации.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок

Кроме того, было реализовано несколько ассемблерных ставок. Вставка, приведённая на рис. 26, осуществляет вычитание из переменных x и y числовое значение 250 в функциях Choice и Menu.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок

Ассемблерная вставка, представленная на рис. 27, используется в функции Draw\_B для выбора начальной координаты рисования сетки и заменяет фрагмент кода, приведённый на рис. 28.

Изображение выглядит как текст

Автоматически созданное описание

Рисунок

Изображение выглядит как текст

Автоматически созданное описание

Рисунок

* 1. **Оценка выполненных оптимизаций**

Проведём суммарный анализ выполненных оптимизаций и их результатов.

Алгоритмическая оптимизация не позволила уменьшить сложность алгоритма, однако с помощью профилирования было установлено, что модифицированная программа стала работать практически в два раза быстрее.

В качестве машинно-независимой оптимизации были выполнены развёрстка и отказ от циклов, вынос инвариантного кода за пределы цикла, удаление ненужного кода, произведена оптимизация переходов. На основе профилирования можно сделать вывод, что функция Game\_Rule стала работать примерно в 2,42 раза быстрее, функция Figure\_Draw – в 1,3 раза быстрее, а вся программа в целом – в 1,76 раза быстрее.

Были выполнены некоторые машинно-зависимые оптимизации, а именно ассемблерные вставки и использование адресной арифметики вместо индексирования. Профилирование для данного вида оптимизации не смогло дать объективных показаний для оценки её эффективности.

# **Заключение**

Начальным этапом выполнения данной курсовой работы был анализ кода исходной программы и её алгоритмической сложности, а также рассмотрение возможных вариантов оптимизации. Опытным путём были получены базовые знания в области профилирования программ, на основе которых производилась оценка эффективности оптимизаций. В представленном порядке реализованы алгоритмическая, машинно-независимая и машинно-зависимая оптимизации, что позволило увеличить скорость работы программы примерно в 2 раза.

# **Приложение А. Полный код оптимизированной программы**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include<stdio.h>

#include<windows.h>

#include<glut.h>

#include<stb\_easy\_font.h>

#include<malloc.h>

struct p {

int life;

int sum;

int pos;

};

int Xwindow = 1280, Ywindow = 720, speed = 30, pixel\_raz = 5, pause = 3, Xlen = 256, Ylen = 144, flag = 0, begin = 0;

byte\*\* field1, \*\* field2;

byte\* A, \* B;

byte figure = 0;

char buffer[10000];

int user = 0, num\_quads, flagok = 0, green\_auto = 0, green\_user = 0, greenST\_auto = 0, greenST\_user = 0, greenEx = 0, green\_exit = 0;

int user\_set\_green[12], user\_set\_yell[12], yeah, prevent = 0;

void firstone(int x, int y) { //ружьё

int i = x / pixel\_raz, j = y / pixel\_raz;

if ((i - 17) > 0 && (i + 19) < Xlen && (j - 5) > 0 && (j + 5) < Ylen) {

field1[i - 7][j] = 1; field1[i - 6][j] = 1; field1[i - 8][j - 1] = 1; field1[i - 7][j - 1] = 1;

field1[i - 8][j - 2] = 1; field1[i - 7][j + 1] = 1; field1[i - 6][j + 1] = 1; field1[i - 5][j + 1] = 1;

field1[i - 7][j + 2] = 1; field1[i - 6][j + 2] = 1;

field1[i - 8][j + 3] = 1; field1[i - 7][j + 3] = 1;

field1[i - 8][j + 4] = 1;

field1[i - 13][j] = 1; field1[i - 12][j] = 1;

field1[i - 13][j + 1] = 1; field1[i - 12][j + 1] = 1;

field1[i - 13][j + 2] = 1; field1[i - 12][j + 2] = 1;

field1[i - 17][j + 1] = 1; field1[i - 16][j + 1] = 1;

field1[i - 17][j + 2] = 1; field1[i - 16][j + 2] = 1;

field1[i + 10][j] = 1; field1[i + 10][j - 1] = 1; field1[i + 10][j - 2] = 1;

field1[i + 9][j + 1] = 1; field1[i + 9][j + 2] = 1;

field1[i + 9][j - 4] = 1; field1[i + 9][j - 3] = 1;

field1[i + 11][j + 1] = 1; field1[i + 11][j - 3] = 1;

field1[i + 12][j] = 1; field1[i + 12][j - 2] = 1;

field1[i + 13][j - 1] = 1;

field1[i + 17][j] = 1; field1[i + 17][j - 1] = 1;

field1[i + 18][j] = 1; field1[i + 18][j - 1] = 1;

}

}

void secondone(int x, int y) { //реле

int i = x / pixel\_raz, j = y / pixel\_raz;

if ((i - 23) > 0 && (i + 16) < Xlen && (j - 6) > 0 && (j + 6) < Ylen) {

field1[i - 4][j] = 1; field1[i - 5][j + 1] = 1; field1[i - 5][j + 2] = 1; field1[i - 4][j + 2] = 1; field1[i - 3][j + 2] = 1;

field1[i - 20][j + 1] = 1; field1[i - 19][j + 1] = 1; field1[i - 18][j + 1] = 1; field1[i - 17][j + 1] = 1;

field1[i - 16][j + 1] = 1; field1[i - 15][j + 1] = 1; field1[i - 14][j + 1] = 1; field1[i - 13][j + 1] = 1;

field1[i - 20][j + 2] = 1; field1[i - 18][j + 2] = 1; field1[i - 17][j + 2] = 1; field1[i - 16][j + 2] = 1; field1[i - 15][j + 2] = 1;

field1[i - 13][j + 2] = 1;

field1[i - 20][j + 3] = 1; field1[i - 19][j + 3] = 1; field1[i - 18][j + 3] = 1; field1[i - 17][j + 3] = 1;

field1[i - 16][j + 3] = 1; field1[i - 15][j + 3] = 1; field1[i - 14][j + 3] = 1; field1[i - 13][j + 3] = 1;

field1[i + 5][j - 1] = 1; field1[i + 6][j - 1] = 1; field1[i + 7][j - 1] = 1; field1[i + 8][j - 1] = 1;

field1[i + 9][j - 1] = 1; field1[i + 10][j - 1] = 1; field1[i + 11][j - 1] = 1; field1[i + 12][j - 1] = 1;

field1[i + 5][j - 2] = 1; field1[i + 7][j - 2] = 1; field1[i + 8][j - 2] = 1; field1[i + 9][j - 2] = 1;

field1[i + 10][j - 2] = 1; field1[i + 12][j - 2] = 1;

field1[i + 5][j - 3] = 1; field1[i + 6][j - 3] = 1; field1[i + 7][j - 3] = 1; field1[i + 8][j - 3] = 1;

field1[i + 9][j - 3] = 1; field1[i + 10][j - 3] = 1; field1[i + 11][j - 3] = 1; field1[i + 12][j - 3] = 1;

}

}

void thirdone(int x, int y) { //галактика кока (сюрикен)

int i = x / pixel\_raz, j = y / pixel\_raz;

if ((i - 8) > 0 && (i + 8) < Xlen && (j - 8) > 0 && (j + 8) < Ylen) {

for (int hey = -1; hey <= 4; hey++) { field1[i - 4][j + hey] = 1; field1[i - 3][j + hey] = 1; }

for (int hey = -4; hey <= 1; hey++) { field1[i + hey][j - 4] = 1; field1[i + hey][j - 3] = 1; }

for (int hey = -4; hey <= 1; hey++) { field1[i + 4][j + hey] = 1; field1[i + 3][j + hey] = 1; }

for (int hey = -1; hey <= 4; hey++) { field1[i + hey][j + 4] = 1; field1[i + hey][j + 3] = 1; }

}

prevent = 1;

}

void fourthone(int x, int y) { //Achim's p144

int i = x / pixel\_raz, j = y / pixel\_raz;

if ((i - 15) > 0 && (i + 15) < Xlen && (j - 11) > 0 && (j + 11) < Ylen) {

field1[i + 13][j - 10] = 1; field1[i + 12][j - 10] = 1; field1[i + 12][j - 9] = 1; field1[i + 13][j - 9] = 1;

field1[i + 13][j + 8] = 1; field1[i + 13][j + 7] = 1; field1[i + 12][j + 8] = 1; field1[i + 12][j + 7] = 1;

field1[i - 14][j - 10] = 1; field1[i - 14][j - 9] = 1; field1[i - 13][j - 10] = 1; field1[i - 13][j - 9] = 1;

field1[i - 14][j + 8] = 1; field1[i - 14][j + 7] = 1; field1[i - 13][j + 8] = 1; field1[i - 13][j + 7] = 1;

field1[i + 3][j - 7] = 1; field1[i + 4][j - 8] = 1; field1[i + 4][j - 6] = 1; field1[i + 5][j - 8] = 1; field1[i + 5][j - 6] = 1;

field1[i + 6][j - 7] = 1;

field1[i - 4][j + 5] = 1; field1[i - 5][j + 4] = 1; field1[i - 5][j + 6] = 1; field1[i - 6][j + 4] = 1; field1[i - 6][j + 6] = 1;

field1[i - 7][j + 5] = 1;

field1[i][j] = 1; field1[i - 1][j] = 1; field1[i - 2][j] = 1; field1[i - 2][j + 1] = 1; field1[i - 2][j + 2] = 1; field1[i - 1][j + 2] = 1; field1[i][j + 2] = 1;

field1[i][j + 1] = 1; field1[i][j - 1] = 1; field1[i - 1][j - 1] = 1; field1[i - 1][j - 2] = 1; field1[i - 1][j - 3] = 1; field1[i - 1][j - 4] = 1;

field1[i][j - 4] = 1; field1[i + 1][j - 4] = 1; field1[i + 1][j - 3] = 1; field1[i + 1][j - 2] = 1; field1[i][j - 2] = 1;

}

prevent = 1;

}

void WaitWat(int n, int x, int y) {

switch (n) {

case 2:

firstone(x, y); //k or K

break;

case 3:

secondone(x, y); //l or L

break;

case 4:

thirdone(x, y); //j or J

break;

case 5:

fourthone(x, y); //h or H

break;

}

}

void Auto\_New\_Game\_B() {

srand(time(NULL));

A = (byte\*)malloc(Xlen \* Ylen \* sizeof(byte));

if (A == NULL)

exit(0);

for (int i = 0; i < Xlen \* Ylen; i += 16) {

A[i] = rand() % 2;

A[i + 1] = rand() % 2;

A[i + 2] = rand() % 2;

A[i + 3] = rand() % 2;

A[i + 4] = rand() % 2;

A[i + 5] = rand() % 2;

A[i + 6] = rand() % 2;

A[i + 7] = rand() % 2;

A[i + 8] = rand() % 2;

A[i + 9] = rand() % 2;

A[i + 10] = rand() % 2;

A[i + 11] = rand() % 2;

A[i + 12] = rand() % 2;

A[i + 13] = rand() % 2;

A[i + 14] = rand() % 2;

A[i + 15] = rand() % 2;

}

begin = 1;

glMatrixMode(GL\_PROJECTION);

gluOrtho2D(0, Xwindow, Ywindow, 0);

//glClearColor(0, 0, 0, 0);

}

void Figure\_New\_Game() {

figure = 1;

//srand(time(NULL));

int i, j;

field1 = (int\*\*)malloc(Xlen \* sizeof(int));

field2 = (int\*\*)calloc(Xlen, sizeof(int));

if (field1 == NULL || field2 == NULL)

exit(0);

int y = 0;

for (i = 0; i < Xlen; i++)

{

field1[i] = (int\*)calloc(Ylen, sizeof(int));

field2[i] = (int\*)calloc(Ylen, sizeof(int));

if (field1[i] == NULL || field2[i] == NULL)

exit(0);

}

field1[1][11 + 5] = 1; field1[2][11 + 5] = 1; field1[2][11 + 6] = 1; field1[1][11 + 6] = 1;

field1[12][11 + 5] = 1; field1[12][11 + 6] = 1; field1[12][11 + 7] = 1; field1[13][11 + 8] = 1;

field1[14][11 + 9] = 1; field1[15][11 + 8] = 1; field1[16][11 + 7] = 1; field1[17][11 + 7] = 1;

field1[16][11 + 6] = 1; field1[17][11 + 6] = 1; field1[16][11 + 5] = 1; field1[17][11 + 5] = 1;

field1[15][11 + 4] = 1; field1[14][11 + 3] = 1; field1[13][11 + 4] = 1; field1[26][11 + 1] = 1;

field1[23][11 + 2] = 1; field1[24][11 + 2] = 1; field1[25][11 + 2] = 1; field1[26][11 + 2] = 1;

field1[22][11 + 3] = 1; field1[23][11 + 3] = 1; field1[24][11 + 3] = 1; field1[25][11 + 3] = 1;

field1[22][11 + 4] = 1; field1[25][11 + 4] = 1; field1[22][11 + 5] = 1; field1[23][11 + 5] = 1;

field1[24][11 + 5] = 1; field1[25][11 + 5] = 1; field1[23][11 + 6] = 1; field1[24][11 + 6] = 1;

field1[25][11 + 6] = 1; field1[26][11 + 6] = 1; field1[26][11 + 7] = 1; field1[31][11 + 5] = 1;

field1[31][11 + 6] = 1; field1[35][11 + 3] = 1; field1[36][11 + 3] = 1; field1[35][11 + 4] = 1;

field1[36][11 + 4] = 1; field1[57][11 + 38] = 1; field1[62][11 + 38] = 1; field1[55][11 + 39] = 1;

field1[56][11 + 39] = 1; field1[58][11 + 39] = 1; field1[59][11 + 39] = 1; field1[60][11 + 39] = 1;

field1[61][11 + 39] = 1; field1[63][11 + 39] = 1; field1[64][11 + 39] = 1; field1[57][11 + 40] = 1; field1[62][11 + 40] = 1;// ружьё

field1[67][12] = 1; field1[68][12] = 1; field1[69][12] = 1; field1[67][13] = 1; field1[68][13] = 1;

field1[69][13] = 1; field1[67][14] = 1; field1[68][14] = 1; field1[69][14] = 1; field1[70][15] = 1;

field1[71][15] = 1; field1[72][15] = 1; field1[70][16] = 1; field1[71][16] = 1; field1[72][16] = 1;

field1[70][17] = 1; field1[71][17] = 1; field1[72][17] = 1;//восьмёрка

field1[87][17] = 1; field1[88][17] = 1; field1[89][17] = 1; field1[87][16] = 1; field1[88][16] = 1;

field1[89][16] = 1; field1[87][15] = 1; field1[88][15] = 1; field1[89][15] = 1; field1[90][14] = 1;

field1[91][14] = 1; field1[92][14] = 1; field1[90][13] = 1; field1[91][13] = 1; field1[92][13] = 1;

field1[90][12] = 1; field1[91][12] = 1; field1[92][12] = 1;//обратная восьмёрка

field1[110][12] = 1; field1[111][12] = 1; field1[112][12] = 1; field1[116][12] = 1; field1[117][12] = 1;

field1[118][12] = 1; field1[108][14] = 1; field1[108][15] = 1; field1[108][16] = 1; field1[113][14] = 1;

field1[113][15] = 1; field1[113][16] = 1; field1[110][17] = 1; field1[111][17] = 1; field1[112][17] = 1;

field1[115][14] = 1; field1[115][15] = 1; field1[115][16] = 1; field1[120][14] = 1; field1[120][15] = 1;

field1[120][16] = 1; field1[116][17] = 1; field1[117][17] = 1; field1[118][17] = 1; field1[110][19] = 1;

field1[111][19] = 1; field1[112][19] = 1; field1[116][19] = 1; field1[117][19] = 1; field1[118][19] = 1;

field1[108][20] = 1; field1[108][21] = 1; field1[108][22] = 1; field1[113][20] = 1; field1[113][21] = 1;

field1[113][22] = 1; field1[115][20] = 1; field1[115][21] = 1; field1[115][22] = 1; field1[110][24] = 1;

field1[111][24] = 1; field1[112][24] = 1; field1[116][24] = 1; field1[117][24] = 1; field1[118][24] = 1;

field1[120][20] = 1; field1[120][21] = 1; field1[120][22] = 1;//пульсар

field1[140][12] = 1; field1[141][12] = 1; field1[142][12] = 1; field1[143][12] = 1;

field1[144][12] = 1; field1[145][12] = 1; field1[146][12] = 1; field1[147][12] = 1;

field1[140][14] = 1; field1[141][14] = 1; field1[142][14] = 1; field1[143][14] = 1;

field1[144][14] = 1; field1[145][14] = 1; field1[146][14] = 1; field1[147][14] = 1;

field1[140][13] = 1; field1[147][13] = 1;

for (int i = 142; i <= 145; i++)

field1[i][13] = 1;

field1[155][12] = 1; field1[156][11] = 1; field1[155][13] = 1; field1[156][13] = 1; field1[157][13] = 1;

for (int i = 165; i <= 172; i++) {

field1[i][8] = 1;

field1[i][10] = 1;

}

field1[165][9] = 1; field1[172][9] = 1;

for (int i = 167; i <= 170; i++)

field1[i][9] = 1;//реле

field1[188][12] = 1; field1[189][12] = 1; field1[190][12] = 1;

for (int i = 183; i <= 195; i++)

if (i != 185 && i != 193)

field1[i][13] = 1;

field1[183][14] = 1; field1[187][14] = 1; field1[189][14] = 1; field1[191][14] = 1; field1[195][14] = 1;

for (int i = 184; i <= 194; i++)

if (i != 188 && i != 189 && i != 190)

field1[i][15] = 1;

field1[186][16] = 1; field1[192][16] = 1; field1[186][17] = 1; field1[192][17] = 1;

field1[180][18] = 1; field1[181][18] = 1; field1[185][18] = 1; field1[186][18] = 1; field1[192][18] = 1;

field1[193][18] = 1; field1[197][18] = 1; field1[198][18] = 1; field1[180][19] = 1; field1[198][19] = 1;

for (int i = 183; i <= 195; i++)

if (i != 188 && i != 190)

field1[i][19] = 1;

field1[181][20] = 1; field1[182][20] = 1; field1[183][20] = 1;

field1[188][20] = 1; field1[189][20] = 1; field1[190][20] = 1;

field1[195][20] = 1; field1[196][20] = 1; field1[197][20] = 1;

field1[183][22] = 1; field1[184][22] = 1; field1[183][23] = 1;

field1[184][25] = 1; field1[185][25] = 1; field1[185][24] = 1;

field1[194][22] = 1; field1[195][22] = 1; field1[195][23] = 1;

field1[193][24] = 1; field1[193][25] = 1; field1[194][25] = 1;// фонтан

for (int i = 212; i <= 215; i++) {

field1[i][12] = 1; field1[i][19] = 1;

}

for (int i = 210; i <= 217; i++) {

if (i != 213 && i != 214) {

field1[i][14] = 1; field1[i][17] = 1;

}

}

field1[212][13] = 1; field1[215][13] = 1; field1[210][15] = 1; field1[217][15] = 1;

field1[210][16] = 1; field1[217][16] = 1; field1[212][18] = 1; field1[215][18] = 1; //кросс

field1[234][12] = 1; field1[235][12] = 1; field1[236][12] = 1;

field1[232][14] = 1; field1[234][14] = 1; field1[236][14] = 1; field1[238][14] = 1;

field1[230][16] = 1; field1[232][16] = 1; field1[238][16] = 1; field1[240][16] = 1;

field1[230][17] = 1; field1[240][17] = 1;

field1[230][18] = 1; field1[232][18] = 1; field1[238][18] = 1; field1[240][18] = 1;

field1[232][20] = 1; field1[234][20] = 1; field1[236][20] = 1; field1[238][20] = 1;

field1[234][22] = 1; field1[235][22] = 1; field1[236][22] = 1;//звезда

for (int i = 10; i <= 15; i++) {

field1[i][160 - 60] = 1; field1[i][160 - 61] = 1;

}

for (int i = 13; i <= 18; i++) {

field1[i][160 - 67] = 1; field1[i][160 - 68] = 1;

}

for (int j = 160 - 63; j >= 160 - 68; j--) {

field1[10][j] = 1; field1[11][j] = 1;

}

for (int j = 160 - 60; j >= 160 - 65; j--) {

field1[17][j] = 1; field1[18][j] = 1;

}//галактика Кока

for (int i = 41; i <= 48; i++) {

field1[i][160 - 50] = 1; field1[i][160 - 52] = 1;

}

for (int i = 41; i <= 48; i++)

if (i != 42 && i != 47)

field1[i][160 - 51] = 1;

for (int i = 40; i <= 47; i++) {

field1[i][160 - 65] = 1; field1[i][160 - 67] = 1;

}

for (int i = 40; i <= 47; i++)

if (i != 41 && i != 46)

field1[i][160 - 66] = 1;

field1[45][160 - 57] = 1; field1[43][160 - 58] = 1; field1[46][160 - 58] = 1;

field1[43][160 - 59] = 1; field1[44][160 - 60] = 1; field1[46][160 - 59] = 1;//мухомор

field1[71][160 - 57] = 1; field1[71][160 - 59] = 1; field1[70][160 - 58] = 1; field1[72][160 - 58] = 1;

field1[71][160 - 65] = 1; field1[71][160 - 67] = 1; field1[70][160 - 66] = 1; field1[72][160 - 66] = 1;

field1[86][160 - 57] = 1; field1[86][160 - 59] = 1; field1[85][160 - 58] = 1; field1[87][160 - 58] = 1;

field1[86][160 - 65] = 1; field1[86][160 - 67] = 1; field1[85][160 - 66] = 1; field1[87][160 - 66] = 1;

field1[79][160 - 58] = 1; field1[80][160 - 58] = 1; field1[81][160 - 58] = 1; field1[79][160 - 59] = 1;

field1[79][160 - 60] = 1; field1[80][160 - 60] = 1; field1[81][160 - 60] = 1; field1[81][160 - 59] = 1;

field1[79][160 - 64] = 1; field1[80][160 - 64] = 1; field1[81][160 - 64] = 1; field1[79][160 - 65] = 1;

field1[79][160 - 66] = 1; field1[80][160 - 66] = 1; field1[81][160 - 66] = 1; field1[81][160 - 65] = 1;//эврика

field1[111][160 - 45] = 1; field1[112][160 - 45] = 1;

field1[106][160 - 46] = 1; field1[107][160 - 46] = 1; field1[109][160 - 46] = 1; field1[114][160 - 46] = 1; field1[116][160 - 46] = 1; field1[117][160 - 46] = 1;

field1[106][160 - 47] = 1; field1[117][160 - 47] = 1;

field1[107][160 - 48] = 1; field1[108][160 - 48] = 1; field1[115][160 - 48] = 1; field1[116][160 - 48] = 1;

for (int i = 104; i <= 119; i++)

if (i != 107 && i != 108 && i != 115 && i != 116)

field1[i][160 - 49] = 1;

field1[104][160 - 50] = 1; field1[107][160 - 50] = 1; field1[116][160 - 50] = 1; field1[119][160 - 50] = 1;

field1[101][160 - 51] = 1; field1[102][160 - 51] = 1; field1[104][160 - 51] = 1; field1[106][160 - 51] = 1;

field1[117][160 - 51] = 1; field1[119][160 - 51] = 1; field1[121][160 - 51] = 1; field1[122][160 - 51] = 1;

field1[101][160 - 52] = 1; field1[103][160 - 52] = 1; field1[105][160 - 52] = 1; field1[118][160 - 52] = 1; field1[120][160 - 52] = 1; field1[122][160 - 52] = 1;

field1[103][160 - 53] = 1; field1[120][160 - 53] = 1;

field1[101][160 - 54] = 1; field1[104][160 - 54] = 1; field1[119][160 - 54] = 1; field1[122][160 - 54] = 1;

field1[104][160 - 55] = 1; field1[112][160 - 55] = 1; field1[113][160 - 55] = 1; field1[114][160 - 55] = 1; field1[119][160 - 55] = 1;

field1[100][160 - 56] = 1; field1[104][160 - 56] = 1; field1[112][160 - 56] = 1; field1[114][160 - 56] = 1; field1[119][160 - 56] = 1; field1[123][160 - 56] = 1;

field1[100][160 - 57] = 1; field1[104][160 - 57] = 1; field1[112][160 - 57] = 1; field1[114][160 - 57] = 1; field1[119][160 - 57] = 1; field1[123][160 - 57] = 1;

field1[104][160 - 58] = 1; field1[119][160 - 58] = 1;

field1[101][160 - 59] = 1; field1[104][160 - 59] = 1; field1[119][160 - 59] = 1; field1[122][160 - 59] = 1;

field1[103][160 - 60] = 1; field1[120][160 - 60] = 1;

field1[101][160 - 61] = 1; field1[103][160 - 61] = 1; field1[105][160 - 61] = 1; field1[118][160 - 61] = 1; field1[120][160 - 61] = 1; field1[122][160 - 61] = 1;

field1[101][160 - 62] = 1; field1[102][160 - 62] = 1; field1[104][160 - 62] = 1; field1[106][160 - 62] = 1;

field1[117][160 - 62] = 1; field1[119][160 - 62] = 1; field1[121][160 - 62] = 1; field1[122][160 - 62] = 1;

field1[104][160 - 63] = 1; field1[107][160 - 63] = 1; field1[116][160 - 63] = 1; field1[119][160 - 63] = 1;

for (int i = 104; i <= 119; i++)

if (i != 107 && i != 108 && i != 115 && i != 116)

field1[i][160 - 64] = 1;

field1[107][160 - 65] = 1; field1[108][160 - 65] = 1; field1[115][160 - 65] = 1; field1[116][160 - 65] = 1;

field1[106][160 - 66] = 1; field1[117][160 - 66] = 1;

field1[106][160 - 67] = 1; field1[107][160 - 67] = 1; field1[109][160 - 67] = 1; field1[114][160 - 67] = 1; field1[116][160 - 67] = 1; field1[117][160 - 67] = 1;

field1[111][160 - 68] = 1; field1[112][160 - 68] = 1;//pi portraitor

field1[142][160 - 45] = 1; field1[147][160 - 45] = 1; field1[142][160 - 47] = 1; field1[147][160 - 47] = 1;

for (int i = 140; i <= 149; i++)

if (i != 142 && i != 147)

field1[i][160 - 46] = 1;

field1[142][160 - 55] = 1; field1[147][160 - 55] = 1; field1[142][160 - 57] = 1; field1[147][160 - 57] = 1;

for (int i = 140; i <= 149; i++)

if (i != 142 && i != 147)

field1[i][160 - 56] = 1;

field1[142][160 - 65] = 1; field1[147][160 - 65] = 1; field1[142][160 - 67] = 1; field1[147][160 - 67] = 1;

for (int i = 140; i <= 149; i++)

if (i != 142 && i != 147)

field1[i][160 - 66] = 1;//пентадекатлоны

field1[177][160 - 62] = 1; field1[182][160 - 62] = 1; field1[177][160 - 64] = 1; field1[182][160 - 64] = 1;

for (int i = 175; i <= 184; i++)

if (i != 177 && i != 182)

field1[i][160 - 63] = 1;

field1[170][160 - 58] = 1; field1[171][160 - 58] = 1; field1[188][160 - 58] = 1; field1[189][160 - 58] = 1;

field1[171][160 - 59] = 1; field1[188][160 - 59] = 1;

field1[171][160 - 60] = 1; field1[173][160 - 60] = 1; field1[186][160 - 60] = 1; field1[188][160 - 60] = 1;

field1[172][160 - 61] = 1; field1[173][160 - 61] = 1; field1[186][160 - 61] = 1; field1[187][160 - 61] = 1;

field1[172][160 - 65] = 1; field1[173][160 - 65] = 1; field1[186][160 - 65] = 1; field1[187][160 - 65] = 1;

field1[171][160 - 66] = 1; field1[173][160 - 66] = 1; field1[186][160 - 66] = 1; field1[188][160 - 66] = 1;

field1[171][160 - 67] = 1; field1[188][160 - 67] = 1;

field1[170][160 - 68] = 1; field1[171][160 - 68] = 1; field1[188][160 - 68] = 1; field1[189][160 - 68] = 1;//шейкер

field1[204][160 - 58] = 1; field1[213][160 - 58] = 1;

field1[202][160 - 59] = 1; field1[203][160 - 59] = 1; field1[204][160 - 59] = 1; field1[213][160 - 59] = 1; field1[214][160 - 59] = 1; field1[215][160 - 59] = 1;

field1[201][160 - 60] = 1; field1[216][160 - 60] = 1;

field1[201][160 - 61] = 1; field1[204][160 - 61] = 1; field1[205][160 - 61] = 1; field1[206][160 - 61] = 1;

field1[211][160 - 61] = 1; field1[212][160 - 61] = 1; field1[213][160 - 61] = 1; field1[216][160 - 61] = 1;

field1[200][160 - 62] = 1; field1[201][160 - 62] = 1; field1[203][160 - 62] = 1; field1[207][160 - 62] = 1;

field1[210][160 - 62] = 1; field1[214][160 - 62] = 1; field1[216][160 - 62] = 1; field1[217][160 - 62] = 1;

field1[203][160 - 63] = 1; field1[207][160 - 63] = 1; field1[208][160 - 63] = 1;

field1[209][160 - 63] = 1; field1[210][160 - 63] = 1; field1[214][160 - 63] = 1;

field1[203][160 - 64] = 1; field1[207][160 - 64] = 1; field1[210][160 - 64] = 1; field1[214][160 - 64] = 1;

field1[204][160 - 65] = 1; field1[205][160 - 65] = 1; field1[206][160 - 65] = 1;

field1[211][160 - 65] = 1; field1[212][160 - 65] = 1; field1[213][160 - 65] = 1;

field1[204][160 - 67] = 1; field1[205][160 - 67] = 1; field1[207][160 - 67] = 1;

field1[210][160 - 67] = 1; field1[212][160 - 67] = 1; field1[213][160 - 67] = 1;

field1[204][160 - 68] = 1; field1[206][160 - 68] = 1; field1[207][160 - 68] = 1;

field1[210][160 - 68] = 1; field1[211][160 - 68] = 1; field1[213][160 - 68] = 1;//очки

for (int i = 231; i <= 239; i++)

if (i != 235)

field1[i][160 - 63] = 1;

for (int j = 160 - 58; j >= 160 - 68; j--)

if (j != 160 - 63)

field1[235][j] = 1;

field1[233][160 - 59] = 1; field1[237][160 - 59] = 1;

field1[231][160 - 60] = 1; field1[233][160 - 60] = 1; field1[237][160 - 60] = 1; field1[239][160 - 60] = 1;

field1[232][160 - 61] = 1; field1[233][160 - 61] = 1; field1[237][160 - 61] = 1; field1[238][160 - 61] = 1;

field1[230][160 - 62] = 1; field1[240][160 - 62] = 1;

field1[230][160 - 64] = 1; field1[240][160 - 64] = 1;

field1[232][160 - 65] = 1; field1[233][160 - 65] = 1; field1[237][160 - 65] = 1; field1[238][160 - 65] = 1;

field1[231][160 - 66] = 1; field1[233][160 - 66] = 1; field1[237][160 - 66] = 1; field1[239][160 - 66] = 1;

field1[233][160 - 67] = 1; field1[237][160 - 67] = 1;//колесо

glMatrixMode(GL\_PROJECTION);

gluOrtho2D(-1, Xwindow, Ywindow, -1);

//glClearColor(0, 0, 0, 0);

}

void User\_New\_Game\_B() {

//srand(time(NULL));

A = (byte\*)calloc(Xlen \* Ylen, sizeof(byte));

if (A == NULL)

exit(0);

glMatrixMode(GL\_PROJECTION);

gluOrtho2D(0, Xwindow, Ywindow, 0);

//glClearColor(0, 0, 0, 0);

}

char Game\_Rule(int x, int y) {

short sum\_neighbors = field1[x - 1][y] + field1[x - 1][y - 1] + field1[x - 1][y + 1] + field1[x][y + 1]

+ field1[x][y - 1] + field1[x + 1][y] + field1[x + 1][y - 1] + field1[x + 1][y + 1];

if (!field1[x][y])

return !!(sum\_neighbors == 3);

else

return !!(sum\_neighbors == 3 || sum\_neighbors == 2);

}

void Draw\_B() {

//glClear(GL\_COLOR\_BUFFER\_BIT);

glPointSize(pixel\_raz);

byte\* p = A;

byte\* pM = A + Xlen \* Ylen;

for (int i = 0; p < pM; p++, i++) {

if (\*p)

glColor3f(rand() % 2, rand() % 2, rand() % 2);

else

glColor3f(0, 0, 0);

glBegin(GL\_POINTS);

glVertex2f(i / Ylen \* pixel\_raz, i % Ylen \* pixel\_raz);

glEnd();

}

int i, j;

\_asm {

cmp [pixel\_raz],10

jne Else

mov[i], 6

mov[j], 6

Else:

cmp[pixel\_raz], 5

jne End

mov[i], 3

mov[j], 3

End:

}

if (pixel\_raz == 5 || pixel\_raz == 10) {

glColor3f(0.2, 0.2, 0.2);

glLineWidth(0.2);

for (i; i <= Xwindow; i += pixel\_raz) {

glBegin(GL\_LINES);

glVertex2f(i, 0);

glVertex2f(i, Ywindow);

glEnd();

}

for (j; j <= Ywindow; j += pixel\_raz) {

glBegin(GL\_LINES);

glVertex2f(0, j);

glVertex2f(Xwindow, j);

glEnd();

}

}

if (begin) {

struct p\* mass = (struct p\*)malloc(Xlen \* Ylen \* sizeof(struct p));

int t = 0;

for (int i = 1; i < Ylen - 1; i++)

for (int j = 1; j < Xlen - 1; j++)

{

int pos = j \* Ylen + i;

byte sum = (byte)(

A[pos - Ylen - 1] + A[pos - Ylen] + A[pos - Ylen + 1] +

A[pos - 1] + A[pos + 1] +

A[pos + Ylen - 1] + A[pos + Ylen] + A[pos + Ylen + 1]);

if (sum == 2 || sum == 3) {

mass[t].life = A[pos];

mass[t].sum = sum;

mass[t].pos = pos;

t++;

}

}

memset(A, 0, Xlen \* Ylen \* sizeof(byte));

for (int i = 0; i < t; i++) {

if (mass[i].life)

A[mass[i].pos] = 1;

else if (mass[i].sum == 3)

A[mass[i].pos] = 1;

}

free(mass);

}

glutSwapBuffers();

}

void Figure\_Draw() {

glPointSize(pixel\_raz);

for (int i = 1; i < Xlen-1; i++)

for (int j = 1; j < Ylen-1; j++)

{

if (field1[i][j])

glColor3f(rand() % 2, rand() % 2, rand() % 2);

else

glColor3f(0, 0, 0);

glBegin(GL\_POINTS);

glVertex2f(i \* pixel\_raz, j \* pixel\_raz);

glEnd();

}

int i, j;

i = j = 3;

glColor3f(0.2, 0.2, 0.2);

glLineWidth(0.2);

for (i; i <= Xwindow; i += pixel\_raz) {

glBegin(GL\_LINES);

glVertex2f(i, 0);

glVertex2f(i, Ywindow);

glEnd();

}

for (j; j <= Ywindow; j += pixel\_raz) {

glBegin(GL\_LINES);

glVertex2f(0, j);

glVertex2f(Xwindow, j);

glEnd();

}

if (begin) {

for (int i = 1; i < Xlen - 1; i++)

for (int j = 1; j < Ylen - 1; j += 4) {

field2[i][j] = Game\_Rule(i, j);

field2[i][j + 1] = Game\_Rule(i, j + 1);

field2[i][j + 2] = Game\_Rule(i, j + 2);

field2[i][j + 3] = Game\_Rule(i, j + 3);

}

}

glutSwapBuffers();

}

void swap() {

int\*\* tmp;

tmp = field1;

field1 = field2;

field2 = tmp;

}

void Timer(int n) {

Draw\_B();

glutTimerFunc(speed, Timer, 0);

}

void Figure\_Timer(int n) {

Figure\_Draw();

swap();

glutTimerFunc(speed, Figure\_Timer, 0);

}

void StandartKeybord(unsigned char key, int x, int y) {

if (key == 32)

Sleep(pause \* 1000);

if (key == 13) {

if (!begin)

begin = 1;

else {

Sleep(3000);

free(field1);

free(field2);

free(A);

exit(0);

}

}

if (key == 75 || key == 107) yeah = 1;

else if (key == 76 || key == 108) yeah = 2;

else if (key == 74 || key == 106) yeah = 3;

else if (key == 72 || key == 104) yeah = 4;

}

void SpecialKeybord(int key, int x, int y) {

if (key == GLUT\_KEY\_UP && speed > 10)

speed -= 10;

if (key == GLUT\_KEY\_UP && speed < 10 && speed >= 2)

speed--;

if (key == GLUT\_KEY\_DOWN)

speed += 10;

}

void Mouse\_Pressed\_Move(int x, int y) {

if (x / pixel\_raz < 0 || y / pixel\_raz < 0 || x / pixel\_raz >= Xlen || y / pixel\_raz >= Ylen)

exit(0);

if (figure)

field1[x / pixel\_raz][y / pixel\_raz] = 1;

else

A[x / pixel\_raz \* Ylen + y / pixel\_raz] = 1;

}

void Mouse\_Pressed(int button, int state, int x, int y) {

if (x / pixel\_raz < 0 || y / pixel\_raz < 0 || x / pixel\_raz >= Xlen || y / pixel\_raz >= Ylen)

exit(0);

if (button == GLUT\_LEFT\_BUTTON && button == GLUT\_DOWN) {

if (figure)

field1[x / pixel\_raz][y / pixel\_raz] = 1;

else

A[x / pixel\_raz \* Ylen + y / pixel\_raz] = 1;

if (figure) {

yeah++;

WaitWat(yeah, x, y);

yeah >>=3;

}

}

}

void Window\_Size(GLint w, GLint h) {

if (glutGet(GLUT\_WINDOW\_WIDTH) != Xwindow || glutGet(GLUT\_WINDOW\_HEIGHT) != Ywindow)

glutReshapeWindow(Xwindow, Ywindow);

}

void Quads(int x1, int y1, int x2, int y2) {

glBegin(GL\_QUADS);

glVertex3f(x1, y1, 0);

glVertex3f(x1, y2, 0);

glVertex3f(x2, y2, 0);

glVertex3f(x2, y1, 0);

glEnd();

}

void Menu() {

if (!flagok) {

glScalef(0.15, -0.15, 1);

flagok = 1;

}

glPushMatrix();

glColor3f(1, 1, 1);

glEnableClientState(GL\_VERTEX\_ARRAY);

glVertexPointer(2, GL\_FLOAT, 16, buffer);

glScalef(1, 1, 0);

glColor3f(1, 1, 0);

num\_quads = stb\_easy\_font\_print(-35, -70, "GAME OF LIFE", NULL, buffer, sizeof(buffer));

glDrawArrays(GL\_QUADS, 0, num\_quads \* 4);

glColor3f(1, 1, 0);

if (user)

glColor4f(0.0f, 1.0f, 1.0f, 1.0f);

if (!user)

glColor3f(1, 1, 0);

if (green\_auto == 1)

glColor3f(0, 1, 0);

Quads(-65, -50, -22, -30);

glColor3f(1, 0, 0);

glScalef(0.8, 0.8, 0.8);

num\_quads = stb\_easy\_font\_print(-75, -60, " Auto\nSettings", NULL, buffer, sizeof(buffer));

glDrawArrays(GL\_QUADS, 0, num\_quads \* 4);

glColor4f(0.0f, 1.0f, 1.0f, 1.0f);

if (user)

glColor3f(1, 1, 0);

if (green\_user == 1)

glColor3f(0, 1, 0);

Quads(28, -62, 82, -38);

glColor3f(1, 0, 0);

num\_quads = stb\_easy\_font\_print(35, -60, " User\nSettings", NULL, buffer, sizeof(buffer));

glDrawArrays(GL\_QUADS, 0, num\_quads \* 4);

num\_quads = stb\_easy\_font\_print(-75, -30, "Window Speed Size Pause", NULL, buffer, sizeof(buffer));

glDrawArrays(GL\_QUADS, 0, num\_quads \* 4);

glScalef(0.6, 0.6, 0.6);

glColor3f(0.5, 0.5, 0.5);

int\* P\_l = user\_set\_yell;

if (user) {

if (\*P\_l)

glColor3f(1, 1, 0);

else

glColor4f(0.0f, 1.0f, 1.0f, 1.0f);

}

int\* P\_g = user\_set\_green;

if (\*P\_g)

glColor3f(0, 1, 0);

Quads(-126, -32, -68, -21);

if (user) {

if (\*(P\_l+1))

glColor3f(1, 1, 0);

else

glColor4f(0.0f, 1.0f, 1.0f, 1.0f);

}

if (\*(P\_g +1))

glColor3f(0, 1, 0);

Quads(-126, -19, -68, -9);

if (user) {

if (\*(P\_l + 2))

glColor3f(1, 1, 0);

else

glColor4f(0.0f, 1.0f, 1.0f, 1.0f);

}

if (\*(P\_g + 2))

glColor3f(0, 1, 0);

Quads(-126, -7, -68, 2);

glColor3f(1, 0, 0);

num\_quads = stb\_easy\_font\_print(-125, -30, " 1530x800\n 1000x760\n 800x600 ", NULL, buffer, sizeof(buffer));

glDrawArrays(GL\_QUADS, 0, num\_quads \* 4);

glColor3f(0.5, 0.5, 0.5);

if (user) {

if (\*(P\_l + 3))

glColor3f(1, 1, 0);

else

glColor4f(0.0f, 1.0f, 1.0f, 1.0f);

}

if (\*(P\_g + 3))

glColor3f(0, 1, 0);

Quads(-50, -32, 0, -21);

if (user) {

if (\*(P\_l + 4))

glColor3f(1, 1, 0);

else

glColor4f(0.0f, 1.0f, 1.0f, 1.0f);

}

if (\*(P\_g + 4))

glColor3f(0, 1, 0);

Quads(-50, -19, 0, -9);

if (user) {

if (\*(P\_l + 5))

glColor3f(1, 1, 0);

else

glColor4f(0.0f, 1.0f, 1.0f, 1.0f);

}

if (\*(P\_g + 5))

glColor3f(0, 1, 0);

Quads(-50, -7, 0, 3);

glColor3f(1, 0, 0);

num\_quads = stb\_easy\_font\_print(-45, -30, " High\nAverage\n Low", NULL, buffer, sizeof(buffer));

glDrawArrays(GL\_QUADS, 0, num\_quads \* 4);

glColor3f(0.5, 0.5, 0.5);

if (user) {

if (\*(P\_l + 6))

glColor3f(1, 1, 0);

else

glColor4f(0.0f, 1.0f, 1.0f, 1.0f);

}

if (\*(P\_g + 6))

glColor3f(0, 1, 0);

Quads(15, -32, 65, -21);

if (user) {

if (\*(P\_l + 7))

glColor3f(1, 1, 0);

else

glColor4f(0.0f, 1.0f, 1.0f, 1.0f);

}

if (\*(P\_g + 7))

glColor3f(0, 1, 0);

Quads(15, -19, 65, -9);

if (user) {

if (\*(P\_l + 8))

glColor3f(1, 1, 0);

else

glColor4f(0.0f, 1.0f, 1.0f, 1.0f);

}

if (\*(P\_g + 8))

glColor3f(0, 1, 0);

Quads(15, -7, 65, 3);

glColor3f(1, 0, 0);

num\_quads = stb\_easy\_font\_print(20, -30, " Big\nAverage\n Small", NULL, buffer, sizeof(buffer));

glDrawArrays(GL\_QUADS, 0, num\_quads \* 4);

glColor3f(0.5, 0.5, 0.5);

if (user) {

if (\*(P\_l + 9))

glColor3f(1, 1, 0);

else

glColor4f(0.0f, 1.0f, 1.0f, 1.0f);

}

if (\*(P\_g + 9))

glColor3f(0, 1, 0);

Quads(78, -32, 125, -21);

if (user) {

if (\*(P\_l + 10))

glColor3f(1, 1, 0);

else

glColor4f(0.0f, 1.0f, 1.0f, 1.0f);

}

if (\*(P\_g + 10))

glColor3f(0, 1, 0);

Quads(78, -19, 125, -9);

if (user) {

if (\*(P\_l + 11))

glColor3f(1, 1, 0);

else

glColor4f(0.0f, 1.0f, 1.0f, 1.0f);

}

if (\*(P\_g + 11))

glColor3f(0, 1, 0);

Quads(78, -7, 125, 3);

glColor3f(1, 0, 0);

num\_quads = stb\_easy\_font\_print(90, -30, " 3 c\n 5 c\n10 c", NULL, buffer, sizeof(buffer));

glDrawArrays(GL\_QUADS, 0, num\_quads \* 4);

glScalef(2, 2, 2);

glColor4f(0.0f, 1.0f, 1.0f, 1.0f);

if (greenST\_auto == 1)

glColor3f(0, 1, 0);

Quads(-45, 17, 50, 29);

glColor4f(0.0f, 1.0f, 1.0f, 1.0f);

if (greenST\_user == 1)

glColor3f(0, 1, 0);

Quads(-45, 30, 50, 42);

if (!user) {

glColor4f(0.0f, 1.0f, 1.0f, 1.0f);

if (greenEx == 1)

glColor3f(0, 1, 0);

}

else

glColor3f(0.5, 0.5, 0.5);

Quads(-45, 43, 50, 53);

glColor4f(0.0f, 1.0f, 1.0f, 1.0f);

if (green\_exit == 1)

glColor3f(0, 1, 0);

Quads(-45, 54, 50, 65);

glColor3f(1, 0, 0);

num\_quads = stb\_easy\_font\_print(-40, 20, "Start Auto Game\nStart User Game\n Examples\n Exit", NULL, buffer, sizeof(buffer));

glDrawArrays(GL\_QUADS, 0, num\_quads \* 4);

glDisableClientState(GL\_VERTEX\_ARRAY);

glPopMatrix();

glutSwapBuffers();

glScalef(1, 1, 1);

}

void Botton\_color(int x, int y) {

\_asm {

mov edx, x

sub edx, 250

mov x, edx

mov edx, y

sub edx, 250

mov y, edx

}

if (x > -200 && x<-70 && y>-155 && y < -95)

green\_auto = 1;

else

green\_auto = 0;

if (x > 70 && x<205 && y>-155 && y < -95)

green\_user = 1;

else

green\_user = 0;

if (x > -135 && x < 150 && y > 50 && y < 87)

greenST\_auto = 1;

else

greenST\_auto = 0;

if (x > -135 && x < 150 && y < 125 && y>90)

greenST\_user = 1;

else

greenST\_user = 0;

if (x > -135 && x < 150 && y < 158 && y>130)

greenEx = 1;

else

greenEx = 0;

if (x > -135 && x < 150 && y < 194 && y>165)

green\_exit = 1;

else

green\_exit = 0;

glutPostRedisplay();

if (user) {

int\* P\_g = user\_set\_green;

if (x > -190 && x<-102 && y>-50 && y < -33)

\*P\_g = 1;

else

\*P\_g = 0;

if (x > -190 && x<-102 && y>-28 && y < -15)

\*(P\_g+1) = 1;

else

\*(P\_g + 1) = 0;

if (x > -190 && x<-102 && y>-12 && y < 3)

\*(P\_g + 2) = 1;

else

\*(P\_g + 2) = 0;

if (x > -75 && x <0 && y>-50 && y < -33)

\*(P\_g + 3) = 1;

else

\*(P\_g + 3) = 0;

if (x > -75 && x <0 && y>-28 && y < -15)

\*(P\_g + 4) = 1;

else

\*(P\_g + 4) = 0;

if (x > -75 && x <0 && y>-12 && y < 3)

\*(P\_g + 5) = 1;

else

\*(P\_g + 5) = 0;

if (x > 22 && x <96 && y>-50 && y < -33)

\*(P\_g + 6) = 1;

else

\*(P\_g + 6) = 0;

if (x > 22 && x <96 && y>-28 && y < -15)

\*(P\_g + 7) = 1;

else

\*(P\_g + 7) = 0;

if (x > 22 && x <96 && y>-12 && y < 3)

\*(P\_g + 8) = 1;

else

\*(P\_g + 8) = 0;

if (x > 116 && x <186 && y>-50 && y < -33)

\*(P\_g + 9) = 1;

else

\*(P\_g + 9) = 0;

if (x > 116 && x <186 && y>-28 && y < -15)

\*(P\_g + 10) = 1;

else

\*(P\_g + 10) = 0;

if (x > 116 && x <186 && y>-12 && y < 3)

\*(P\_g + 11) = 1;

else

\*(P\_g + 11) = 0;

glutPostRedisplay();

}

}

void Reshape(GLint w, GLint h) {

if (glutGet(GLUT\_WINDOW\_WIDTH) != 500 || glutGet(GLUT\_WINDOW\_HEIGHT) != 500)

glutReshapeWindow(500, 500);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

gluOrtho2D(-12.0, 12.0, -12.0, 12.0);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

if (flagok)

glScalef(0.15, -0.15, 1);

}

void Сhoice(int btn, int state, int x, int y) {

\_asm {

mov edx, x

sub edx, 250

mov x, edx

mov edx, y

sub edx, 250

mov y, edx

}

if (state == GLUT\_DOWN && btn == GLUT\_LEFT\_BUTTON) {

if (x > -203 && x<-70 && y>-155 && y < -95) {

user = 0;

Xwindow = 1280;

Ywindow = 720;

speed = 20;

pixel\_raz = 5;

pause = 5;

Xlen = Xwindow / pixel\_raz;

Ylen = Ywindow / pixel\_raz;

}

if (x > 70 && x<205 && y>-155 && y < -95)

user = 1;

glutPostRedisplay();

if (x > -135 && x < 150 && y > 50 && y < 87) {

begin = 1;

glutDestroyWindow(1);

glutInitWindowSize(Xwindow, Ywindow);

glutInitWindowPosition(0, 5);

glutInitDisplayMode(GLUT\_DOUBLE);

glutCreateWindow("Game of Life");

Auto\_New\_Game\_B();

glutDisplayFunc(Draw\_B);

Timer(0);

glutMouseFunc(Mouse\_Pressed);

glutMotionFunc(Mouse\_Pressed\_Move);

glutKeyboardFunc(StandartKeybord);

glutSpecialFunc(SpecialKeybord);

glutReshapeFunc(Window\_Size);

glutMainLoop();

}

if (x > -135 && x < 150 && y < 125 && y>90) {

glutDestroyWindow(1);

glutInitWindowSize(Xwindow, Ywindow);

glutInitWindowPosition(0, 5);

glutInitDisplayMode(GLUT\_RGB | GLUT\_DOUBLE);

glutCreateWindow("Game of Life");

User\_New\_Game\_B();

glutMouseFunc(Mouse\_Pressed);

glutMotionFunc(Mouse\_Pressed\_Move);

glutDisplayFunc(Draw\_B);

Timer(0);

glutKeyboardFunc(StandartKeybord);

glutSpecialFunc(SpecialKeybord);

glutReshapeFunc(Window\_Size);

glutMainLoop();

}

if (x > -135 && x < 150 && y < 158 && y>130 && !user) {

begin = 1;

glutDestroyWindow(1);

glutInitWindowSize(Xwindow, Ywindow);

glutInitWindowPosition(0, 5);

glutInitDisplayMode(GLUT\_RGB | GLUT\_DOUBLE);

glutCreateWindow("Game of Life");

Figure\_New\_Game();

glutMouseFunc(Mouse\_Pressed);

glutMotionFunc(Mouse\_Pressed\_Move);

glutDisplayFunc(Figure\_Draw);

Figure\_Timer(0);

glutKeyboardFunc(StandartKeybord);

glutSpecialFunc(SpecialKeybord);

glutReshapeFunc(Window\_Size);

glutMainLoop();

}

if (x > -135 && x < 150 && y < 194 && y>165)

exit(0);

if (user) {

int\* P\_y = user\_set\_yell;

if (x > -190 && x<-102 && y>-50 && y < -33) {

Xwindow = 1530;

Ywindow = 800;

\*P\_y = 1;

}

else if ((!\*(P\_y + 1) || !\*(P\_y + 2)) && (x > -190 && x < -102))

\*P\_y = 0;

if (x > -190 && x<-102 && y>-28 && y < -15) {

Xwindow = 1000;

Ywindow = 760;

\*(P\_y+1) = 1;

}

else if ((!\*P\_y || !\*(P\_y + 2)) && (x > -190 && x < -102))

\*(P\_y + 1) = 0;

if (x > -190 && x<-102 && y>-12 && y < 3) {

Xwindow = 800;

Ywindow = 600;

\*(P\_y + 2) = 1;

}

else if ((!\*P\_y || !\*(P\_y + 1)) && (x > -190 && x < -102))

\*(P\_y + 2) = 0;

if (x > -75 && x <0 && y>-50 && y < -33) {

speed = 10;

\*(P\_y + 3) = 1;

}

else if ((!\*(P\_y + 4) || !\*(P\_y + 5)) && (x > -75 && x < 0))

\*(P\_y + 3) = 0;

if (x > -75 && x <0 && y>-28 && y < -15) {

speed = 50;

\*(P\_y + 4) = 1;

}

else if ((!\*(P\_y + 5) || !\*(P\_y + 6)) && (x > -75 && x < 0))

\*(P\_y + 4) = 0;

if (x > -75 && x <0 && y>-12 && y < 3) {

speed = 200;

\*(P\_y + 5) = 1;

}

else if ((!\*(P\_y + 3) || !\*(P\_y + 4)) && (x > -75 && x < 0))

\*(P\_y + 5) = 0;

if (x > 22 && x <96 && y>-50 && y < -33) {

pixel\_raz = 10;

\*(P\_y + 6) = 1;

}

else if ((!\*(P\_y + 7) || !\*(P\_y + 8)) && (x > 22 && x < 96))

\*(P\_y +6) = 0;

if (x > 22 && x <96 && y>-28 && y < -15) {

pixel\_raz = 5;

\*(P\_y +7) = 1;

}

else if ((!\*(P\_y + 6) || !\*(P\_y + 8)) && (x > 22 && x < 96))

\*(P\_y + 7) = 0;

if (x > 22 && x <96 && y>-12 && y < 3) {

pixel\_raz = 2;

\*(P\_y + 8) = 1;

}

else if ((!\*(P\_y + 6) || !\*(P\_y + 7)) && (x > 22 && x < 96))

\*(P\_y + 8) = 0;

if (x > 116 && x <186 && y>-50 && y < -33) {

pause = 3;

\*(P\_y + 9) = 1;

}

else if ((!\*(P\_y + 10) || !\*(P\_y + 11)) && (x > 116 && x < 186))

\*(P\_y + 9) = 0;

if (x > 116 && x <186 && y>-28 && y < -15) {

pause = 5;

\*(P\_y + 10) = 1;

}

else if ((!\*(P\_y + 9) || !\*(P\_y + 11)) && (x > 116 && x < 186))

\*(P\_y + 10) = 0;

if (x > 116 && x <186 && y>-12 && y < 3) {

pause = 10;

\*(P\_y + 11) = 1;

}

else if ((!\*(P\_y + 9) || !\*(P\_y + 10)) && (x > 116 && x < 186))

\*(P\_y + 11) = 0;

}

if (Xwindow && Ywindow && speed && pause && pixel\_raz) {

Xlen = Xwindow / pixel\_raz;

Ylen = Ywindow / pixel\_raz;

}

glutPostRedisplay();

}

}

int main(char argc, char\*\* argv) {

FreeConsole();

glutInit(&argc, argv);

glutInitWindowSize(500, 500);

glutInitWindowPosition(200, 200);

glutInitDisplayMode(GLUT\_DOUBLE);

glutCreateWindow("Game of Life");

glutDisplayFunc(Menu);

glutReshapeFunc(Reshape);

glutPassiveMotionFunc(Botton\_color);

glutMouseFunc(Сhoice);

glutMainLoop();

}