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. Карамышев Е. С.

студенты гр. 4831001/10002

Преподаватель Малышев Е. В.

«\_\_\_» \_\_\_\_\_\_\_\_\_\_ 2023 г.

Санкт-Петербург

2023

**Содержание**

[**Введение** 3](#_Toc124711990)

[**Глава 1** 4](#_Toc124711991)

[**1.1.** **Профилирование и анализ исходной программы** 4](#_Toc124711992)

[**1.2.** **Возможные варианты оптимизации** 6](#_Toc124711993)

[**Глава 2** 8](#_Toc124711994)

[**2.1.** **Реализация алгоритмической оптимизации** 8](#_Toc124711995)

[**2.2.** **Машинно-независимая оптимизация** 10](#_Toc124711996)

[**2.3.** **Машинно-зависимая оптимизация** 10](#_Toc124711997)

[**2.4.** **Оценка выполненных оптимизаций** 10](#_Toc124711998)

[**Заключение** 11](#_Toc124711999)

# **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 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

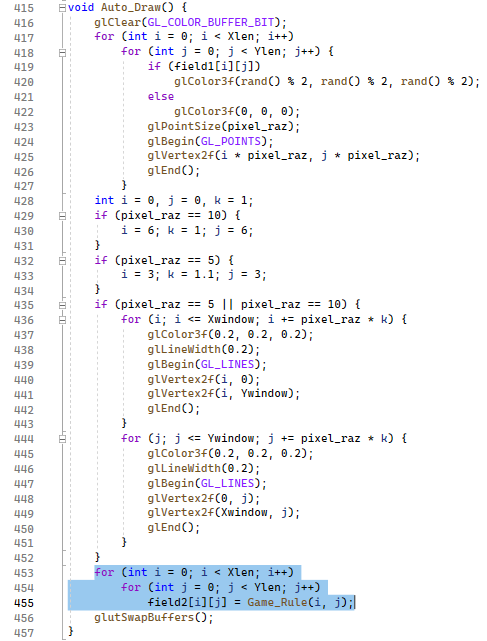
Рисунок

Изображение выглядит как текст

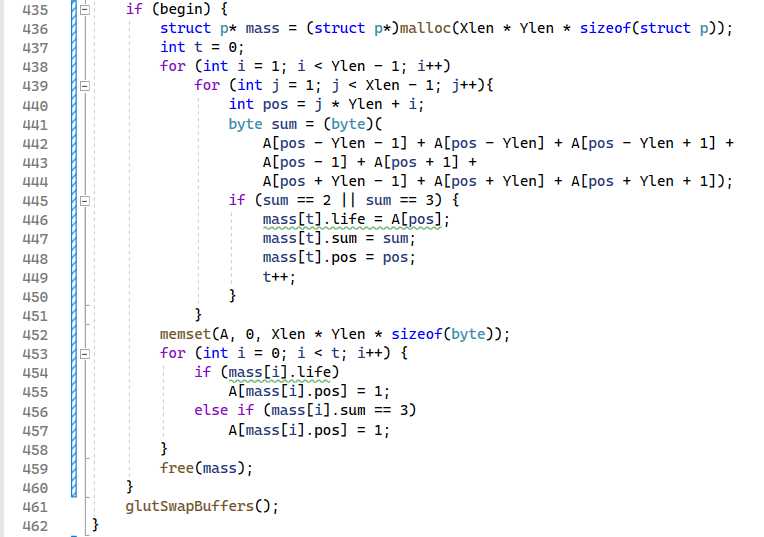
Автоматически созданное описание

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.



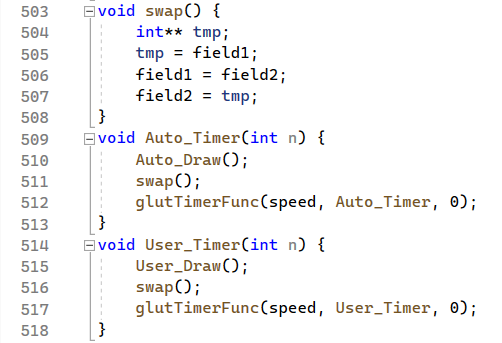
Рисунок



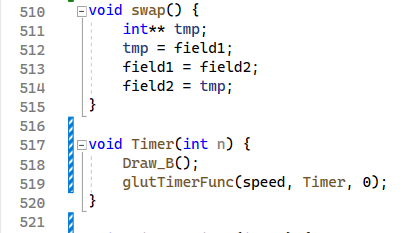
Рисунок

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 1,9 2 раза.

Изображение выглядит как стол

Автоматически созданное описание

Рисунок

Изображение выглядит как стол

Автоматически созданное описание

Рисунок

* 1. **Машинно-независимая оптимизация**

**…**

* 1. **Машинно-зависимая оптимизация**

**…**

* 1. **Оценка выполненных оптимизаций**

**…**

# **Заключение**

…