МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное автономное образовательное учреждение высшего образования
«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

ДОПУСТИТЬ К ЗАЩИТЕ

Заведующий кафедрой № 14

зав.каф., к.т.н., доцент	13.06.2024	В.Л. Оленев
должность, уч. степень, звание	подпись, дата	инициалы, фамилия

БАКАЛАВРСКАЯ РАБОТА

на тему Игровое приложение «Судоку»

выполнена Чумиловым Ильей Андреевичем
фамилия, имя, отчество студента в творительном падеже

по направлению подготовки 09.03.01 (код) Информатика и вычислительная техника (наименование направления)

(наименование направления)

направленности 01 (код) Автоматизированные системы (наименование направленности)

обработки информации и управления
наименование направленности

Студент группы № 1044 — 10.06.24 (подпись, дата) — И.А.Чумилов (инициалы, фамилия)

Руководитель

канд. техн. наук	10.06.24	А.В. Шахомиров
должность, уч. степень, звание	подпись, дата	инициалы, фамилия

Санкт-Петербург 2024

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное автономное образовательное учреждение высшего образования
«САНКТ–ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

УТВЕРЖДАЮ

Заведующий кафедрой № 14

зав.каф., к.т.н., доцент	27.03.2024	В. Л. Оленев
должность, уч. степень, звание	подпись, дата	инициалы, фамилия

ЗАДАНИЕ НА ВЫПОЛНЕНИЕ БАКАЛАВРСКОЙ РАБОТЫ

студенту группы 1044 (номер) Чумилову Илье Андреевичу (фамилия, имя, отчество)

на тему Игровое приложение «Судоку»

утвержденную приказом ГУАП от 27.03.2024 № 11-413/24

Цель работы: Разработка игрового приложения «Сукдоку» и на его основе анализ работы алгоритмов генерации игрового поля

Задачи, подлежащие решению: 1) изучить правила игры «Судоку», изучить, как должны располагаться числа на игровом поле, и какого оно может быть размера;
2) выбрать методы и реализовать алгоритмы заполнения игрового поля;
3) разработать игровое приложение и на его основе провести анализ алгоритмов генерации игрового поля, скорости работы и затрат памяти.

Содержание работы (основные разделы): 1) Выбор алгоритмов.
2) Реализация алгоритмов и игрового приложения со всем функционалом.
3) Анализ скорости работы и затрат памяти алгоритмов

Срок сдачи работы « 10 » июня 2024

Руководитель

доцент, к.т.н.	27.03.24	А. В. Шахомиров
должность, уч. степень, звание	подпись, дата	инициалы, фамилия

Задание принял(а) к исполнению

студент группы № 1044	27.03.24	И.А.Чумилов
	подпись, дата	инициалы, фамилия

РЕФЕРАТ

Выпускная квалификационная работа содержит 92 стр., 37 рис., 20 листингов, 6 табл., 14 источников, 6 прил.

Ключевые слова: алгоритм, генерация, анализ, функция, Windows окно, приложение, программное обеспечение (далее по тексту ПО).

Актуальность темы работы заключается в практической демонстрации важности выбора какого-либо алгоритма или метода при проектировании и разработке ПО. Этот выбор существенно влияет на работу всего продукта, в частности на скорость выполнения, надежность системы и масштабируемость.

Цель работы – разработка игрового приложения «Судоку», на основе которого будет проведен анализ эффективности различных алгоритмов, применяемых для заполнения игрового поля.

В процессе работы использовалась IDE Microsoft Visual Studio Community версии 2022, листинг программы оформлен в соответствии с ГОСТ 7.32-2017.

Полученные результаты и их новизна: разработанное приложение может быть использовано при изучении алгоритмов в роли наглядного примера. Данные, полученные в процессе анализа, покажут, как различные алгоритмы влияют на скорость работы и затраты памяти ПО.

Области применения: разработка и оптимизация ПО, образование и обучение программированию.

В будущем планируется более глубокое сравнение имеющихся алгоритмов, их улучшение, добавление других, а также применение подобного подхода анализа в других областях программирования.

СОДЕРЖАНИЕ

РЕФЕРАТ 3
ВВЕДЕНИЕ 6
1 История появления игры «Судоку» 7
2 Алгоритмы, использующиеся для генерации поля 9
2.1 Алгоритм Backtracking 9
2.2 Алгоритм Dancing links 11
2.3 Генетический алгоритм 14
3 Язык C++. Используемые библиотеки и заголовочные файлы C++ 18
4 Реализация алгоритмов генерации игрового поля 20
4.1 Реализация алгоритма Back tracking 20
4.2 Реализация алгоритма Dancing links 23
4.3 Реализация Генетического алгоритма 27
5 Реализация важных для работы программы алгоритмов 33
5.1 Алгоритм создания нулевых ячеек 33
5.2 Главная функция создания окон 34
5.3 Алгоритм сохранения настроек игры 36
5.4 Алгоритм проверки игрового поля 39
6 Демонстрация работы приложения 41
7 Анализ алгоритмов генерации игрового поля 47
ЗАКЛЮЧЕНИЕ 59
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 60
ПРИЛОЖЕНИЕ А. Листинг файла functionsetpole.cpp 62
ПРИЛОЖЕНИЕ Б. Листинг файла dancingLinks.cpp 65
ПРИЛОЖЕНИЕ В. Листинг файла solveWithGeneticAlgorithm.cpp 69
ПРИЛОЖЕНИЕ Г. Листинг файла functiondeletecells.cpp 75

ПРИЛОЖЕНИЕ Д. Листинг файла Diplom.cpp 76
ПРИЛОЖЕНИЕ Е. Листинг файла WindowsProject2.cpp 77

ВВЕДЕНИЕ

В современном мире самые разнообразные технологии окружают всех нас. Мы пользуемся ими в повседневной жизни и на работе, учебе. Каждое утро мы включаем свет у себя дома, ставим чайник, пользуемся телефоном, включаем телевизор. Однако мало кто задумывается о том, как все это работает и как это было создано. Задачей программистов является создание ПО для этих устройств. Каждая программа работает за счет разработанных алгоритмов. От их качества зависит скорость работы программы, передача разных данных, безотказность всей системы.

В процессе обучения будущих программистов часто говориться о важности правильного написания алгоритмов. Часто приведенные примеры не показывают в должной степени, на сколько алгоритмы влияют на работу программы. В небольших проектах разница между ними незначительная и незаметна. Из-за этого при проектировании крупного ПО могут возникнуть разные проблемы в области оптимизации и масштабируемости.

В ходе данной выпускной квалификационной работы проектируется игровое приложение «Судоку». Данная программа позволит провести анализ эффективности алгоритмов, которые генерируют игровое поле, и покажет, что под конкретные задачи требуется внимательно их выбирать. Для этого:

изучаются правила игры «Судоку», конкретно как должны располагаться числа на игровом поле и каким оно может быть по размерам;

выбираются и реализовываются уже имеющиеся алгоритмы и методы по заполнению игрового поля;

разрабатывается приложение и проводится анализ на его основе.

1 История появления игры «Судоку»

Игра «Судоку» – это популярная японская головоломка с числами. Однако придумали ее не там. Сама игра произошла от другой игры «Латинский квадрат» (фр. «Carré latin»). Ее придумал знаменитый швейцарский физик и математик Леонард Эйлер в XVIII веке. Она была измененной версией игры, популярной в 700 году в Китае. Там игра называлась «Магический квадрат». Это был квадрат 3 на 3 клетки. Нужно было разместить цифры от 1 до 9 так, чтобы их сумма в столбце, строке и по диагонали были равны. Леонард Эйлер внес следующие изменения в эту головоломку: он заменил цифры на букву и увеличил размеры поля в 3 раза. Также заменил условия расстановки чисел: они не должны пересекаться ни по вертикали, ни по горизонтали. Этот квадрат назвали «греко-латинский квадрат» [3][7][9].

Aα	Bδ	Cβ	Dε	Eγ
Bβ	Cε	Dγ	Eα	Aδ
Cγ	Dα	Eδ	Aβ	Bε
Dδ	Eβ	Aε	Bγ	Cα
Eε	Aγ	Bα	Cδ	Dβ

Рисунок 1 – Греко-латинский квадрат

На основе игры Эйлера в 1970-х года в Северной Америке стали придумывать свои числовые головоломки. В 1979 году создали головоломку в том виде, в котором она сейчас есть. Ее создателем стал американский архитектор Говард Гарнс. Он опубликовал ее в журнале «Dell Puzzle Magazine». В то время ее названием было «Number Place». Далее эту головоломку начали опубликовать на своих страницах в японском журнале «Nikoli» в 1984-1985 годах. Там она называлась на японском "Sūji wa dokushin

ni kagiru", что переводится как "числа ограничены одним местонахождением". Кайи Маки сократил название игры до «Sudoku», расшифровывается как «Su» – число, «doku» – единое. Игра сразу стала популярна среди японцев, так как японцы любят различные головоломки, которые затрачивают много времени. В Японии многие люди вынуждены тратить много времени на поездки на поезде или автобусе. Данные игры позволяют скоротать время в дороге[8].

Таким образом, данная головоломка стала популярной в Японии и стала печататься во многих газетах. В марте 1997, отдыхая в Токио, в книжном магазине ее обнаружил судья из Новой Зеландии Уэйн Гулд. Игра ему так понравилась, что он потратил шесть лет на разработку компьютерной программы, которая могла бы создавать головоломки Судоку. Его называют человеком, который «вернул» эту игру в западный мир. В дальнейшем игра обрела огромный успех. Она начала появляться в газетах по всему миру и захватывать сердца людей. В 2006 году был проведен первый чемпионат мира по игре Судоку в Италии.

2 Алгоритмы, использующиеся для генерации поля

Для создания алгоритмов, которые генерируют игровые поля, требуется разобраться в правилах игры.

Классическим игровым полем является квадрат размером 9 на 9 клеток, который разделен на квадраты поменьше со сторонами в 3 клетки. Таким образом, числа от 1 до 9 размещаются в 81 клетке. В начале игры часть клеток заполнена числами, которые называют «подсказками». Игрок должен заполнить пустые клетки так, чтобы число не повторялось в каждой строке, столбце и малом квадрате 3 на 3 клетки. Количество изначальных цифр задает сложность игры. 17 чисел являются минимальным допустимым количеством подсказок, необходимым для однозначного решения судоку [7].

Размеры поля могут быть различными, но корень этих чисел должен быть целым, чтобы можно было задать квадраты поменьше. Например: если размеры поля 16 на 16 клеток, то малый квадрат будет со стороной в 4 клетки.

На основе этих правил будем проектировать будущие алгоритмы для игры.

2.1 Алгоритм Backtracking

Алгоритм Backtracking – это рекурсивный итерационный метод, благодаря которому можно решить большое количество различных вычислительных задач комбинаторной оптимизации. Он использует перебор всех возможных значений при заданных ограничениях. В случае, если не находится нужное значение, алгоритм возвращается в предыдущее состояние и продолжает перебор следующего варианта. Данный термин первым ввел американский математик Деррик Генри Лемер в 1950 году [13][14].

Для начала работы алгоритма требуется инициализация начальных значений. Это связано с тем, что, если не будет найдено решение, алгоритм должен вернуться в предыдущее состояние, чтобы попробовать другой вариант. Также алгоритм добавляет новые элементы к уже существующему частичному решению. Поэтому требуется некое начальное состояние [10].

После инициализации начальных значений, алгоритм начинает свою работу. Он начинает искать подходящее значение или состояние для дальнейшего решения задачи. На следующем этапе алгоритм проверяет, соответствует ли данный элемент ограничениям или критериям задачи. В случае успешного нахождения, алгоритм переходит в следующее состояние и начинает свою работу по нахождению походящего значения. Это продолжается до тех пор, пока не будет решена задача. Если в одной из итераций не будет найдено решение, алгоритм вернется к предыдущему состоянию, в котором он будет искать новое подходящее под условие значение. Это будет продолжаться до того момента, пока не будут перебраны все возможные комбинации или найдено конечное решение. Таким образом, количество итераций может быть очень большим и для перебора потребуется большое количество времени. Максимальное количество итераций для данного алгоритма можно оценить как экспоненциальное значение входных данных.

$$k = N! \tag{1}$$

Где k – максимальное значение итераций, N – количество входных значений.

Так как разрабатывается игра «Судоку», приведем пример конкретно для нее. Классическое игровое поле имеет размеры 9 на 9 клеток. Как описывалось выше, при данных размерах мы в клетку можем поместить значение от 1 до 9. Таким образом, при работе алгоритма он будет помещать в каждую клетку сначала число 1. Если оно не повторяется в столбцах, строках и малом квадрате 3 на 3, то алгоритм перейдет к следующей пустой клетке и начнется новая итерация алгоритма. В случае отсутствия подходящего числа для этой клетке, алгоритм вернется к предыдущей и запустит новую итерацию. Алгоритм закончит свою работу, когда найдется конечное решение либо будут

перебраны все возможные комбинации. В данном случае максимальное количество итераций по формуле (1) будет равняться 362880, при N=9.

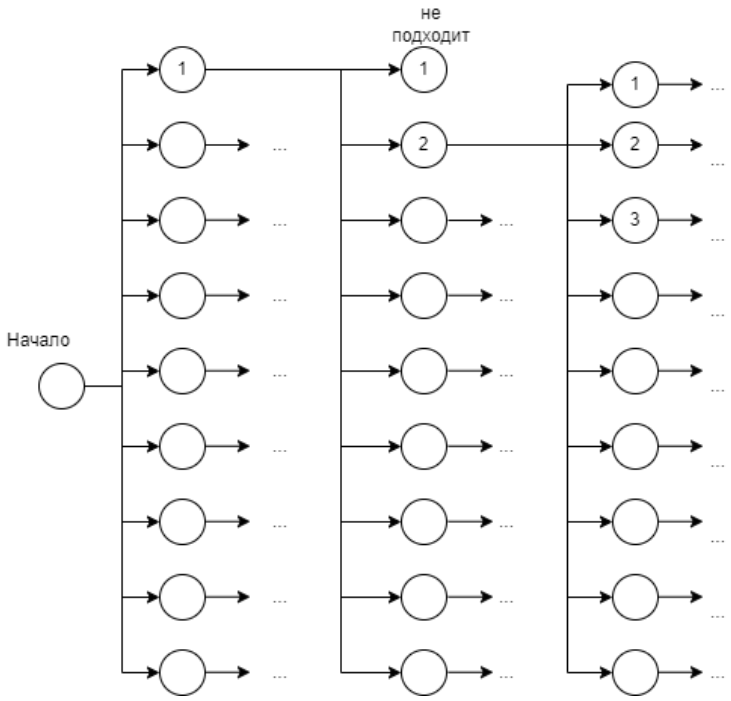

Рисунок 2 – Пример итераций алгоритма Backtracking

Расчет временной сложности позволит нам понять, как размер входных данных влияет на время выполнения алгоритма. Каждая итерация алгоритма представляет собой один шаг в процессе решения задачи. Это означает, что временная сложность напрямую зависит от количества итераций и вычисляется также по формуле (1). При N = 9, временная сложность будет также равна 362880.

2.2 Алгоритм Dancing links

Следующим алгоритмом, который мы будем анализировать в приложении, будет Dancing links. Это метод, который основывается на работе двусвязного списка, в котором происходит добавление и удаление узлов. Данный алгоритм является рекурсивным, он находит всевозможные решения задач точного покрытия, чем является игра Судоку.

Изобретение алгоритма приписывают неким Хироши Хитоцумацу и Кохэю Ношите в 1979. Однако этот алгоритм стал популярен после появление статьи «Dancing Links», которую опубликовал американский математик Дональд Кнут. Он же придумал название «танцевальные ссылки» (Dancing links). Данное название появилось, потому что в процессе работы алгоритма итерации заставляют связи между элементами перемещаться в разных направлениях. Создателю напомнило это танец [1][10].

Дональд Кнут имел свой алгоритм X для решения задач точного покрытия, однако работа его метода требовала большого количества времени. Для решения данной проблемы Кнут реализовал разреженную матрицу, в которой находятся только единицы. Каждый узел матрицы ссылается на соседние узлы, которые находятся слева, справа, сверху и снизу. Также на заголовок своего столбца. Таким образом появляется двусвязный список, состоящий из узлов, для каждой строки и столбца. Заголовком называют узел, который находится в специальной строке. В нее входят все столбцы матрицы. С помощью заголовков начинаются и заканчиваются итерации, в нем хранится информация о количестве элементов, что способствует выбору походящего столбца для алгоритма. Также в заголовке есть указатели, которые позволяют перемещаться можно столбцами матрицы. Заголовок помогает восстанавливать матрицу после выхода из рекурсии [13].

В своей статье Кнут подробно рассказывает про работу данного алгоритма. В нем происходит постоянное удаление и восстановление строк и столбцов в матрице. Выбирая столбцы и строки, мы их исключаем и решаем задачу дальше. Неразрешимой матрица становится тогда, когда в выбранном столбце нет строк. После этого матрицу нужно восстановить. При выполнении удаления происходит комплексное удаление элементов из матрицы. Во-первых, удаляются все столбцы, в которых выбранная строка содержит 1. Во-вторых, вместе с этими столбцами удаляются также все строки (включая саму выбранную строку), которые имеют 1 в любом из этих удаляемых столбцов. Когда столбцы заполнены, происходит его удаление. Строки удаляются, когда

они конфликтуют с выбранной строкой. При удалении столбца, удаляется сначала его заголовок. После чего удаляется каждая строка, где выбранный столбец содержит цифру 1. Строка удаляется также из других столбцов. Данные строки становятся недоступными. Происходит аналогичное удаление других столбцов. Это гарантирует, что удаленный узел будет удален ровно один раз и в предсказуемом порядке, чтобы его можно было соответствующим образом отследить. Если в результирующей матрице нет столбцов, значит, все они заполнены и выбранные строки образуют решение [10].

Для игры «Судоку» алгоритм будет работать следующим образом. Для примера будет использовать классическое поле 9 на 9 клеток и классические правила. Для начала нужно представить поле и ограничения в виде матрицы покрытия. В ней строками будут являться значения, которые могут быть в ячейках игрового поля. Это числа от 1 до 9. Столбцами матрицы будут ограничения: 9 для каждой строки, 9 для столбцов и 9 для квадратов 3 на 3 клетки. В тех строках, столбцах и квадратах, в которых можно разместить выбранное число, будет стоять единица. Так для всех чисел. После создания матрицы покрытия, алгоритм выбирает столбец, в котором присутствует меньше все единиц. Он удаляется, а строки, в которых содержатся единицы связываются. Это нужно для того, чтобы одинаковые цифры не размещались в одной строке, столбце или квадрате. Далее алгоритм для каждой связанной строки помещает значение в ячейку игрового поля. Удаляются столбцы, содержащие единицу в этой строке. Указатели передаются на соседние ячейки для работы структуры двусвязного списка. Далее алгоритм рекурсивно вызывает себя и работает с оставшейся матрицей покрытия. В случае, если решение не найдено, алгоритм восстанавливает удаленный узел, то есть столбцы и строки, и пробудет другое значение. Работа алгоритма заканчивается в тот момент, когда будут заполнено все поле, либо когда алгоритм зайдет в тупик и убедится, что нет решения.

Для расчета максимального количества итераций требуется знать размеры поля и количество клеток. Размеры классического поля 9 на 9 клеток.

Их общее количество 81. Если в начале работы алгоритма часть клеток заполнена, то он будет заполнять пустые клетки значениями от 1 до 9. Например, алгоритму нужно будет заполнить 30 пустых клеток. Значит максимальное количество итераций будет вычисляться следующим образом:

$$k = N^{(S)} \tag{2}$$

Где k – максимальное количество итераций, N – количество цифр, S – количество пустых клеток. Таким образом максимальное количество итераций примерно будет 2.05 * 10^17.

При расчете временной сложности алгоритма, следует учесть выполнение следующих операций: выбор подходящего столбца, восстановление узла в случае неудачи и рекурсивное решение подзадачи. Возьмем вышеуказанный пример. Формула будет следующей:

$$T(n) = O(n * n * k) \tag{3}$$

Где T(n) – временная сложность, k – максимальное количество итераций, n – количество операций, O(n) – верхняя асимптотическая оценка временной сложности.

Итого временная сложность будет примерно равняться 1.6 * 10^19.

2.3 Генетический алгоритм

Генетический алгоритм – эвристический алгоритм поиска, который использует методы аналогичные естественному отбору в природе. Эвристический – это значит, что алгоритм не гарантирует нахождение верного результата. Однако в большинстве случаев результаты удовлетворяют. Обычно такие алгоритмы используют, когда точное решение не может быть найдено или при большом количестве данных. В последнем случае алгоритм будет находить решение намного быстрее, чем алгоритмы, использующие

перебор. Генетический алгоритм использует методы естественной эволюции, такие как наследование, мутации, отбор и скрещивание [14].

Появление такого алгоритма началось в 1954 году, когда орвежско-итальянский математик Нильс Баричелли, работавший в Принстонском университете, провел первые компьютерные опыты по моделированию эволюции. Его работа произвела настоящий фурор в научном мире. В 1957 году, австралийский генетик Алекс Фразер начал публиковать свои исследования по искусственному отбору. Вместе с ним и другими учеными, такими как немецко-американский математик и биофизик Ганс-Иоахим Бремерманн, они постепенно создавали все более сложные имитационные модели эволюционных процессов. В этих моделях присутствовали ключевые элементы современных генетических алгоритмов - популяция решений, подвергаемая рекомбинации, мутации, отбору и скрещиванию. Поначалу все эти модели использовались лишь для простеньких игр. Но в 1960-70-е годы ситуация изменилась, когда немецкие ученые Инго Рехенберг и Ханс-Пауль Швефель доказали, что искусственная эволюция может быть мощным инструментом решения сложных инженерных задач. Параллельно развивался и другой подход - эволюционное программирование для создания искусственного интеллекта, предложенное ученым Лоуренсом Дж. Фогелем. Настоящий прорыв произошел благодаря трудам американского ученого Джона Холланда в начале 1970-х. Именно он разработал теоретические основы и формальные методы, ставшие фундаментом современных генетических алгоритмов. До середины 1980-х все это оставалось уделом ученых-теоретиков, но потом началось практическое применение - появились первые коммерческие продукты, использующие генетические алгоритмы [11].

Сам алгоритм работает следующим образом. Все начинается с начальной популяции решений. Каждое решение – это копия «генотипа». В случае игры Судоку – это начальное поле. Далее популяция оценивается «функцией приспособленности». Она вычисляет «пригодность» решения, то есть считает количество конфликтов. Для игры Судоку конфликтом считается

повторение одного и того же числа в строке, столбце или в квадрате. Далее алгоритм выбирает подходящее решение и применяет «генетические операторы». Ими являются «скрещивание» и «мутация». «Скрещивание» – это процесс, при котором новое решение наследует неизмененные части своих родителей. То есть копирует подходящие под условия ячейки. «Мутацией» называют процесс, при котором меняются «гены» – значения ячеек под подходящие условия. Для новых решений также рассчитывается значение приспособленности. Далее производится отбор лучших решений в следующее поколение и процесс начинает повторятся заново. Он будет повторяться, пока не закончится указанное количество поколений, либо не найдено решение, либо закончится время на эволюцию.

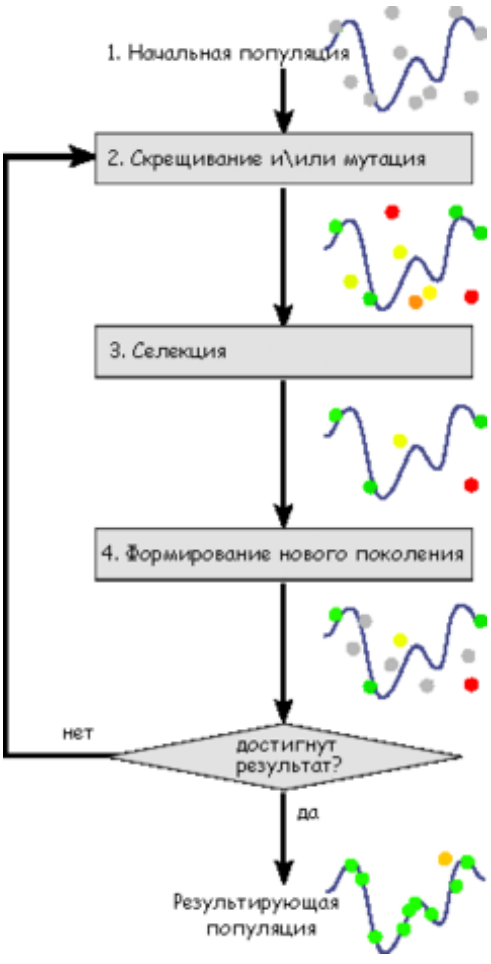

Рисунок 3 – Схема работы генетического алгоритма

Расчет максимального количества итераций не имеет смысла, так как он задается заранее. Временная сложность также зависит от количества итераций. Но так как там есть другие параметры, то можем ее рассчитать по следующей формуле:

$$T(n) = O(G * (size * h^3 + size * log(size) + 2(h^2))) \quad (4)$$

Где T(n) – временная сложность, G – количество итераций, h –размер игрового поля, size – размер популяции, O(n) – верхняя асимптотическая оценка временной сложности.

O (size * h^3) – временная сложность оценки пригодности.

O (size * log(size)) – временная сложность отбора.

O (2(h^2)) – временная сложность скрещивания и мутации.

При 1 итерации, размере поля 9 на 9 и размере популяции 100 получаем значение 81622.

3 Язык C++. Используемые библиотеки и заголовочные файлы C++

Для написания игрового приложения будем использовать язык программирования C++. Это мощный, сложный, но в то же время очень гибкий и универсальный язык программирования, который был создан в 1983 году датским программистом Бьярном Страуструпом. Этот язык является расширением и улучшенной версией языка C, к которому были добавлены объектно-ориентированные возможности и множество других функций. Он может использоваться для разработки самых разных программ - от низкоуровневого системного программного обеспечения и драйверов до высокопроизводительных игр и приложений с красивым графическим интерфейсом. Ключевое преимущество C++ – это возможность для разработчиков получить полный контроль над памятью и аппаратными деталями компьютера. Этот язык включает в себя концепции классов, наследования, полиморфизма и инкапсуляции, которые позволяют создавать более структурированные, масштабируемые и сложные программы [3][5][6].

Данный язык имеет стандартные библиотеки – набор готовых классов, функций и модулей, которые помогут в создании игрового приложения и анализа алгоритмов.

Vector – это библиотека позволяет создавать динамические массивы и вектора. В программе будет возможность выбора размеров игрового поля, поэтому это лучшее решение для данной задачи. Это библиотека также позволяет управлять динамическим массивом с помощью функций push_back(), pop_back() и других [12].

Windows.h – это заголовочный файл, позволяющая работать с операционной системой Windows. Он предоставляет доступ к Windows API, с помощью которого можно создавать окна и управлять ими с помощью макросов и констант [12][4].

Tchar.h – это заголовочный файл, который предоставляет набор макросов и разных типов данных для работы с Unicode-совместимыми

строками для Windows. Нам он требуется для созданий различных окон и полей [12].

Psapi.h - это заголовочный файл, предоставляющий функции для получения информации о процессах, которые запущены в системе. С помощью него мы будем отслеживать использование памяти алгоритмами [12].

Fstream – это библиотека, позволяющая работать с файлами и потоками ввода – вывода в них. В программе будет таймер, который считает время, за которое пользователь решает головоломку. Если пользователь решает быстрее лучшего результата, его результат записывается в файл и будет выводится как лучшее время [12].

Sstream – это библиотека, позволяющая работать с потоками ввода – вывода в памяти, позволяя читать и записывать данные в строки [12].

Random – это библиотека с помощью которой можно генерировать случайные числа при помощи различных алгоритмов, например, mt19937 (Mersenne Twister) [12].

Algorithm – это библиотека, предоставляющая большое количество полезных алгоритмов для работы с массивами и векторами. Эти алгоритмы помогают в поиске, сортировке, перестановке и во многих других случаях [25].

Iostream – это библиотека, позволяющая управлять потоками ввода – вывода в консоль. Также эта библиотека позволяет генерировать случайные числа [12].

4 Реализация алгоритмов генерации игрового поля

Для начала разберем реализацию главных алгоритмов приложения.

4.1 Реализация алгоритма Back tracking

Для реализации данного алгоритма воспользуемся описанием его работы из пункта 2.1.

Для данного алгоритма требуется начальная инициализация игрового поля для того, чтобы каждый раз оно было уникальным при каждом запуске программы. Значения поля будут храниться в двумерном векторе.

Данный вектор мы будем передавать в основную функцию алгоритма вместе с размерами поля h. В самой функции мы будем использовать указатель на этот вектор и через него манипулировать значениями ячеек.

Напишем функцию «baseInit» (листинг 1), которая будет заполнять примерно треть вектора случайными числами. Строки и столбцы также будут выбираться случайным образом. Чтобы соблюдались правила игры, обозначенные в пункте 2, требуется отдельная функция для проверки числа.

```
void baseInit(std::vector<std::vector<long int>>& arr, int h) {
    srand(time(NULL));
    int numCells = h * h / 3;
    for (int i = 0; i < numCells; i++) {
        int row = rand() % h;
        int col = rand() % h;
        if (arr[row][col] == 0) {
            int num = rand() % h + 1;
            if (isSafe(arr, row, col, num, h)) {
                arr[row][col] = num;
            }
        }
    }
}
```

Листинг 1 – Функция «baseInit»

Функция «isSafe» (листинг 2) отвечает за каждую проверку числа на правила игры. С помощью оператора «for» мы будем передвигаться по выбранной строке и столбце. Если появляется совпадение, то данное число

нельзя поставить в данную ячейку и функция вернет «false». Иначе дальше алгоритм высчитывает малый квадрат, и функция проверяет, если это же число в нем. В случае соблюдения всех правил, алгоритм возвращает в функцию, которая его вызвала, значение «true». В случае функции «baseInit» число помещается в вектор и алгоритм продолжает заполнять матрицу дальше.

```
bool isSafe(std::vector<std::vector< long int>>& arr, int row,
int col, int num, int h) {
    // Проверяем строку
    for (int i = 0; i < h; i++) {
        if (arr[row][i] == num)
            return false;
    }

    // Проверяем столбец
    for (int i = 0; i < h; i++) {
        if (arr[i][col] == num)
            return false;
    }

    // Проверяем квадрат
    int sqrtH = sqrt(h);
    int boxRowStart = row - row % sqrtH;
    int boxColStart = col - col % sqrtH;

    for (int i = 0; i < sqrtH; i++) {
        for (int j = 0; j < sqrtH; j++) {
            if (arr[i + boxRowStart][j + boxColStart] == num &&
(i + boxRowStart != row || j + boxColStart != col))
                return false;
        }
    }

    return true;
}
```

Листинг 2 – Функция «isSafe»

После начальной инициализации поля, начинается основная работа алгоритма. Вызывается функция, которая заполняет оставшиеся пустые ячейки. Это будет функция «solveRemaining» (листинг 3). Она начинает свою работу с первой пустой ячейки. Далее в эту ячейку пытается разместить число от 1 до h, используя функцию проверки «isSafe». В случае нахождения

подходящего значения, алгоритм размещает его в матрице и рекурсивно вызывает функцию «solveRemaining». Работа функции будет прекращена в тот момент, когда больше не останется пустых ячеек либо когда невозможно будет разместить число.

```
bool solveRemaining(std::vector<std::vector<long int>>& arr, int
h) {
    int row = -1;
    int col = -1;
    bool isEmpty = true;

    // Находим следующую пустую ячейку
    for (int i = 0; i < h; i++) {
        for (int j = 0; j < h; j++) {
            if (arr[i][j] == 0) {
                row = i;
                col = j;
                isEmpty = false;
                break;
            }
        }
        if (!isEmpty)
            break;
    }

    // Если пустых ячеек нет, значит массив заполнен
    if (isEmpty)
        return true;

    // Пробуем разместить числа от 1 до h
    for (int num = 1; num <= h; num++) {
        if (isSafe(arr, row, col, num, h)) {
            arr[row][col] = num;
            if (solveRemaining(arr, h))
                return true;
            arr[row][col] = 0;
        }
    }

    return false;
}
```

Листинг 3 – Функция «solveRemaining»

Полный листинг этого алгоритма будет находиться в Приложении А.

4.2 Реализация алгоритма Dancing links

Для реализации данного алгоритма воспользуемся описанием его работы из пункта 2.2.

Так как данный алгоритм базируется на узлах и заголовках, то начнем с создания структур заголовков и узлов (листинг 4). В узле должна храниться информация о его строке, столбце, а также значение в нем. Узел связывается соседними узлами с помощью указателей. Также со своим заголовком. В нем хранится информация о его размере и указатель на узел.

```
struct Node {
    int row, col, value;
    Node* up, * down, * left, * right;
    Header* head;

    Node(int r, int c, int v) : row(r), col(c), value(v),
    up(nullptr), down(nullptr), left(nullptr), right(nullptr),
head(nullptr) {}
};

// Структура заголовка узла
struct Header {
    int size;
    Node* head;

    Header() : size(0), head(nullptr) {}
};
```

Листинг 4 – Структуры Node и Header

После инициализации заголовков, их нужно связать с их узлами. Для это создадим вспомогательную функцию «linkNode» (листинг 5). Каждый узел связывается с его заголовком. Первый узел связывается сам с собой, а каждый следующий связывается с последним добавленным через указатели left и right (листинг 6). Узлы связываются с заголовками строки, столбца и квадрата.

```
void linkNode(Header& header, Node* node) {
    node->head = &header;
    if (header.head == nullptr) {
        header.head = node->left = node->right = node;
    }
    else {
        Node* oldHead = header.head;
        node->left = oldHead;
        node->right = oldHead->right;
        oldHead->right->left = node;
        oldHead->right = node;
    }
    header.size++;
}
```

Листинг 5 – Функция «linkNode»

```
for (int row = 0; row < size; row++) {
        for (int col = 0; col < size; col++) {
            for (int value = 1; value <= size; value++) {
                nodes.push_back(new Node(row, col, value)); //
Создаем новый узел
                Node* node = nodes.back();

                // Связываем узел с заголовками строки, столбца
и квадрата
                linkNode(headers[row], node);
                linkNode(headers[size + col], node);
                linkNode(headers[2 * size + (row / h) * h + col
/ h], node);
            }
        }
    }
```

Листинг 6 – Создание и связывания узлов с заголовками

Далее происходит создание двумерного вектора, состоящий из 0, размерами size x size (листинг 8). С помощью генерации случайных чисел, заполняются ячейки игрового поля, если это значение еще не было использовано. Если узел конфликтует с данным значением, то он удаляется. В этом поможет еще одна вспомогательная функция «removeNode» (листинг 7). В ней удаляемый узел соединяет соседние узлы с помощью указателей. Затем уменьшается размер заголовка. Функция последовательно обрабатывает узлы, расположенные выше удаляемого узла в той же строке, столбце или квадрате. Для каждого такого узла она уменьшает размер соответствующего заголовка.

```
void removeNode(Node* node) {
    if (node == nullptr || node->left == nullptr || node->right
== nullptr || node->head == nullptr) {
        return; // Проверка на корректность инициализации узла и
его указателей
    }

    node->left->right = node->right;
    node->right->left = node->left;
    node->head->size--;

    for (Node* curr = node->up; curr != nullptr && curr != node;
curr = curr->up) {
        Node* temp = curr->left;
        while (temp != curr) {
            if (temp->head == nullptr) {
                // Проверка на корректность указателя head
                return;
            }
            temp->head->size--;
            temp = temp->left;
            if (temp == nullptr) {
                // Проверка на корректность указателя left
                return;
            }
        }
    }
}
```

Листинг 7 - Функция «removeNode»

После заполнения всего вектора числами, восстанавливаются узлы для возврата структуры в исходное состояние.

```
std::vector<std::vector<long int>> result(size, std::vector<long
int>(size, 0));
    std::uniform_int_distribution<long int> dist(1, size);
    for (int row = 0; row < size; row++) {
        for (int col = 0; col < size; col++) {
            std::vector<bool> used(size + 1, false); // Вектор
для отслеживания использованных чисел
            int value;
            do {
                value = dist(gen);
            } while (used[value]); // Продолжаем генерацию, пока
не найдем уникальное число
            result[row][col] = value;
            used[value] = true;
```

```
            // Удаляем узлы, конфликтующие с выбранным числом
            for (Node* node : nodes) {
                if ((node->row == row || node->col == col ||
                    (node->row / h) * h + node->col / h == (row
/ h) * h + col / h)
                    && node->value != value) {
                    removeNode(node);
                }
            }
        }
    }

    // Восстанавливаем удаленные узлы
    for (Node* node : nodes) {
        recoverNode(node);
    }

    return result;
```

Листинг 8 - Заполнение вектора числами

Для восстановления узла, функция «recoverNode» (листинг 9) последовательно обрабатывает узлы, расположенные выше восстанавливаемого узла в той же строке, столбце или квадрате. Для каждого такого узла она увеличивает размер соответствующего заголовка. В конце функция восстанавливает сам узел в двусвязный список, обновляя указатели left и right соседних узлов.

```
void recoverNode(Node* node) {
    if (node == nullptr || node->left == nullptr || node->right
== nullptr || node->head == nullptr) {
        return; // Проверка на корректность инициализации узла и
его указателей
    }

    if (node->up != nullptr) {
    for (Node* curr = node->up; curr != node; curr = curr->up) {
         if (curr != nullptr && curr->left != nullptr) {
for (Node* temp = curr->left; temp != curr; temp = temp->left) {
                if (temp != nullptr && temp->head != nullptr) {
                        temp->head->size++;
                    }
                }
            }
        }
    }
```

```
    if (node->head != nullptr) {
        // Проверка на корректность указателя head
        node->head->size++;
        node->left->right = node;
        node->right->left = node;
    }
}
```

Листинг 9 – Функция «recoverNode»

Полный листинг этого алгоритма будет находиться в Приложении Б.

4.3 Реализация Генетического алгоритма

Данный алгоритм последний, который будет анализироваться и сравниваться. Для его реализации воспользуемся описанием из пункта 2.3.

При вызове данного алгоритма в его параметрах передаются значения количества популяции, количества итераций и размеры поля. Его работа начинается с инициализации начальной популяции. Ею является поле игры. Для этого заполним часть поля цифрами. Воспользуемся функцией «baseInit» и «isSafe» из первого алгоритма (см. пункт 4.1(листинг 1 – 2)).

Далее воспользуемся функцией «initPopulation» (листинг 10) для задания начальной популяции. Здесь мы копируем начальное поле в трехмерный вектор.

```
vector<vector<vector<long int>>> initPopulation(int size, const
vector<vector<long int>>& initialBoard, int h) {
    vector<vector<vector<long int>>> population(size);
    for (auto& solution : population) {
        solution = initialBoard;
    }
    return population;
}
```

Листинг 10 – Функция «initPopulation»

Далее происходит оценка каждого решений на пригодность. Для этого воспользуемся функцией «fitness» (листинг 11). В ней происходит проверка решения на конфликты. То есть программа проверяет значения ячеек на

правила игры, начала на строки, затем на столбцы и в конце на квадраты. В каждой проверке с помощью вектора «seen», алгоритм отслеживает использованные числа. Если число повторилось, то к количеству конфликтов добавляется единица. Таким образом создается оценка данного решения на пригодность.

```
int fitness(const vector<vector<long int>>& solution, int h) {
    int conflicts = 0;

    // Проверка строк
    for (int row = 0; row < h; row++) {
        vector<bool> seen(h + 1, false);
        for (int col = 0; col < h; col++) {
            if (solution[row][col] != 0) {
                if (seen[solution[row][col]]) {
                    conflicts++;
                }
                else {
                    seen[solution[row][col]] = true;
                }
            }
        }
    }

    // Проверка столбцов
    for (int col = 0; col < h; col++) {
        vector<bool> seen(h + 1, false);
        for (int row = 0; row < h; row++) {
            if (solution[row][col] != 0) {
                if (seen[solution[row][col]]) {
                    conflicts++;
                }
                else {
                    seen[solution[row][col]] = true;
                }
            }
        }
    }

    // Проверка квадратов
    int sqrtH = static_cast<int>(sqrt(h));
    for (int sqRow = 0; sqRow < sqrtH; sqRow++) {
        for (int sqCol = 0; sqCol < sqrtH; sqCol++) {
            vector<bool> seen(h + 1, false);
for (int row = sqRow * sqrtH; row < (sqRow + 1) * sqrtH; row++)
{
for (int col = sqCol * sqrtH; col < (sqCol + 1) * sqrtH; col++)
{
```

```
                    if (solution[row][col] != 0) {
                        if (seen[solution[row][col]]) {
                            conflicts++;
                        }
                        else {
                            seen[solution[row][col]] = true;
                        }
                    }
                }
            }
        }

    return conflicts;
}
```

Листинг 11 – Функция «fitness»

После оценки всех решений алгоритм сортирует решения и далее выбирается пара «родителей»: первый родитель имеет наименьшую пригодность, а второй – наибольшую. Так как начальная инициализация поля происходит сразу по правилам игры, то для ускорения работы программы будет 2 «родителя», причем одинаковых.

Далее происходит процесс «скрещивания». Для данной пары вызывается функция «crossover» (листинг 12) и создается два потомка, которые наследуют значения полей своих родителей. Остальные пустые ячейки заполняются по правилам игры.

```
vector<vector<long int>> crossover(const vector<vector<long
int>>& parent1, const vector<vector<long int>>& parent2, int h)
{
    vector<vector<long int>> child(h, vector<long int>(h, 0));
    int sqrtH = static_cast<int>(sqrt(h));

    // Копируем неизменные части из родительских решений
    for (int row = 0; row < h; row++) {
        for (int col = 0; col < h; col++) {
            if (parent1[row][col] != 0) {
                child[row][col] = parent1[row][col];
            }
            else if (parent2[row][col] != 0) {
                child[row][col] = parent2[row][col];
            }
        }
```

```
    }

    // Заполняем пустые ячейки, избегая повторений
    for (int row = 0; row < h; row++) {
        for (int col = 0; col < h; col++) {
            if (child[row][col] == 0) {
                vector<bool> seen(h + 1, false);
                // Проверяем строку
                for (int i = 0; i < h; i++) {
                    if (child[row][i] != 0) {
                        seen[child[row][i]] = true;
                    }
                }
                // Проверяем столбец
                for (int i = 0; i < h; i++) {
                    if (child[i][col] != 0) {
                        seen[child[i][col]] = true;
                    }
                }
                // Проверяем квадрат
                int sqRow = row / sqrtH;
                int sqCol = col / sqrtH;
for (int i = sqRow * sqrtH; i < (sqRow + 1) * sqrtH; i++) {
for (int j = sqCol * sqrtH; j < (sqCol + 1) * sqrtH; j++) {
                        if (child[i][j] != 0) {
                            seen[child[i][j]] = true;
                        }
                    }
                }
                for (int num = 1; num <= h; num++) {
                    if (!seen[num]) {
                        child[row][col] = num;
                        break;
                    }
                }
            }
        }
    }

    return child;
}
```

Листинг 12 – Функция «crossover»

После этого каждое решение подвергается «мутации» с помощью функции «mutate» (листинг 13). В ней алгоритм выбирает случайную ячейку и проверяет, возможно ли ее безопасно изменить. Если нельзя, то алгоритм «мутации» прекращается, иначе в ячейку помещается новое число.

```
void mutate(vector<vector<long int>>& solution, int h) {
    random_device rd;
    srand(time(NULL)); // Инициализируем генератор случайных
чисел
    mt19937 gen(rd());
    uniform_int_distribution<> rowDist(0, h - 1);
    uniform_int_distribution<> colDist(0, h - 1);
    uniform_int_distribution<long int> numDist(1, h);

    // Выбираем случайную ячейку для мутации
    int row = rowDist(gen);
    int col = colDist(gen);

    // Проверяем, можно ли безопасно изменить значение ячейки
    vector<bool> seen(h + 1, false);
    bool canMutate = true;

    // Проверяем строку
    for (int i = 0; i < h; i++) {
        if (solution[row][i] != 0) {
            seen[solution[row][i]] = true;
        }
    }
    // Проверяем столбец
    for (int i = 0; i < h; i++) {
        if (solution[i][col] != 0) {
            seen[solution[i][col]] = true;
        }
    }
    // Проверяем квадрат
    int sqrtH = static_cast<int>(sqrt(h));
    int sqRow = row / sqrtH;
    int sqCol = col / sqrtH;
    for (int i = sqRow * sqrtH; i < (sqRow + 1) * sqrtH; i++) {
   for (int j = sqCol * sqrtH; j < (sqCol + 1) * sqrtH; j++) {
            if (solution[i][j] != 0) {
                seen[solution[i][j]] = true;
            }
        }
    }

    // Если в строке, столбце или квадрате есть повторяющиеся
числа, отказываемся от мутации
    for (int num = 1; num <= h; num++) {
        if (seen[num]) {
            canMutate = false;
            break;
        }
    }

    // Выполняем мутацию, если это возможно
    if (canMutate) {
        long int newValue;
```

```
        do {
            newValue = rand() % h + 1;;
        } while (seen[newValue]);
        solution[row][col] = newValue;
    }
}
```

Листинг 13 – Функция «mutate»

Измененные решение помещаются в имеющуюся популяцию. Далее работа алгоритма повторяется с оценки пригодности решений популяции. Количество данных повторов задается заранее и не может быть меньше 1. Каждая итерация улучшает результат работы алгоритма и позволяет получать каждую итерацию более измененные поля. При заданных условия и в общем для игры Судоку хватит одной итерации, после которого из всех решений выбирается лучшее с наименьшим значением пригодности, то есть количества конфликтов.

5 Реализация важных для работы программы алгоритмов

В данном разделе будут упомянуты алгоритмы, которые важны в реализации данного приложения. Весь листинг программы будет находиться в Приложениях Г – Е.

5.1 Алгоритм создания нулевых ячеек

Алгоритмы, с помощью которых генерируется игровое поле, полностью заполняют двумерный вектор числами. Для игры нужны пустые или нулевые ячейки. Для этого используем функцию «deleteRandomCells» (листинг 14). Данная функция будет задавать сложность игры от 1 до 5. Функция обнуляет случайные ячейки. Для поля 4 на 4 количество нулевых ячеек всегда будет равно 10. Для остальных размеров будет вычитываться как умножение сложности на максимальное число. Функция будет возвращать готовый для игры вектор.

```
std::vector<std::vector<long int>>
deleteRandomCells(std::vector<std::vector<long int>>& arrF, int
h,int d) {
    srand(time(NULL)); // Инициализируем генератор случайных
чисел
    std::vector<std::vector<long int>> arr= arrF;
    std::random_device rd;
    std::mt19937 gen(rd());
    std::uniform_int_distribution<> dis(0, h * h - 1);
    int numCells = d * h ;
    if (h == 4) {
        numCells = 10;
    }
    for (int i = 0; i < numCells; i++) {
        int index = dis(gen);
        int row = index / h;
        int col = index % h;
        if (arr[row][col] != 0) {
            arr[row][col] = 0;
        }
        else {
            i--;
        }
    }
```

```
    return arr;
}
```

Листинг 14 – Функция «deleteRandomCells»

5.2 Главная функция создания окон

Функция «WinMain» (листинг 15) нужна для начала работы всей программы, а также для создания окон приложения. Первым делом происходит регистрация класса для главного окна, в котором описываются его размеры, стиль, указатель на оконную процедуру и тд. Данный процесс повторяется для каждого следующего окна. Также происходят проверки на создание окон. Для главного окна также описываются кнопки для дальнейшей работы. В конце описан основной цикл обработки сообщений Windows для приложения.

```
int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
LPSTR lpCmdLine, int nCmdShow) {
    WNDCLASSEX wc;
    HWND hwnd;
    MSG msg;
    AdjustWindowRect(&windowRect, WS_OVERLAPPEDWINDOW, FALSE);

    // Регистрация класса окна для главного окна
    wc.cbSize = sizeof(WNDCLASSEX);
    wc.style = CS_HREDRAW | CS_VREDRAW;
    wc.lpfnWndProc = MainWndProc;
    wc.cbClsExtra = 0;
    wc.cbWndExtra = 0;
    wc.hInstance = hInstance;
    wc.hIcon = LoadIcon(NULL, IDI_APPLICATION);
    wc.hCursor = LoadCursor(NULL, IDC_ARROW);
    wc.hbrBackground = (HBRUSH)(COLOR_WINDOW + 1);
    wc.lpszMenuName = NULL;
    wc.lpszClassName = _T("MainWindowClass");
    wc.hIconSm = LoadIcon(NULL, IDI_APPLICATION);

    if (!RegisterClassEx(&wc)) {
        MessageBox(NULL, _T("Не удалось зарегистрировать класс
главного окна!"), _T("Ошибка"), MB_OK | MB_ICONERROR);
        return 0;
    }

    // Регистрация класса окна для второго окна
    WNDCLASSEX wcChild;
```

```
    wcChild.cbSize = sizeof(WNDCLASSEX);
    wcChild.style = CS_HREDRAW | CS_VREDRAW;
    wcChild.lpfnWndProc = SecondWndProc;
    wcChild.cbClsExtra = 0;
    wcChild.cbWndExtra = 0;
    wcChild.hInstance = hInstance;
    wcChild.hIcon = LoadIcon(NULL, IDI_APPLICATION);
    wcChild.hCursor = LoadCursor(NULL, IDC_ARROW);
    wcChild.hbrBackground = (HBRUSH)(COLOR_WINDOW + 1);
    wcChild.lpszMenuName = NULL;
    wcChild.lpszClassName = _T("SecondWindowClass");
    wcChild.hIconSm = LoadIcon(NULL, IDI_APPLICATION);

    if (!RegisterClassEx(&wcChild)) {
        MessageBox(NULL, _T("Не удалось зарегистрировать класс
второго окна!"), _T("Ошибка"), MB_OK | MB_ICONERROR);
        return 0;
    }

    // Регистрация класса окна для третьего окна
    WNDCLASSEX wcChild2;
    wcChild2.cbSize = sizeof(WNDCLASSEX);
    wcChild2.style = CS_HREDRAW | CS_VREDRAW;
    wcChild2.lpfnWndProc = ThirdWndProc;
    wcChild2.cbClsExtra = 0;
    wcChild2.cbWndExtra = 0;
    wcChild2.hInstance = hInstance;
    wcChild2.hIcon = LoadIcon(NULL, IDI_APPLICATION);
    wcChild2.hCursor = LoadCursor(NULL, IDC_ARROW);
    wcChild2.hbrBackground = (HBRUSH)(COLOR_WINDOW + 1);
    wcChild2.lpszMenuName = NULL;
    wcChild2.lpszClassName = _T("ThirdWindowClass");
    wcChild2.hIconSm = LoadIcon(NULL, IDI_APPLICATION);

    if (!RegisterClassEx(&wcChild2)) {
        MessageBox(NULL, _T("Не удалось зарегистрировать класс
второго окна!"), _T("Ошибка"), MB_OK | MB_ICONERROR);
        return 0;
    }

    // Создание главного окна
    hwnd = CreateWindowEx(0,_T("MainWindowClass"),_T("Главное
окно"),
        WS_OVERLAPPEDWINDOW,windowX, windowY,windowWidth,
windowHeight,NULL, NULL,hInstance,NULL);
    if (!hwnd) {
        MessageBox(NULL, _T("Не удалось создать главное окно!"),
_T("Ошибка"), MB_OK | MB_ICONERROR);
        return 0;
    }

    HWND hChildWnd = NULL, hChildWnd2 = NULL;
```

```
    CreateWindow(_T("BUTTON"), _T("Играть"), WS_VISIBLE |
WS_CHILD, 100, 10, 250, 30, hwnd, (HMENU)ID_BTN_OPEN_WINDOW,
hInstance, NULL);
    CreateWindow(_T("BUTTON"), _T("Настройки игры"), WS_VISIBLE
| WS_CHILD, 100, 100, 250, 30, hwnd, (HMENU)ID_BTN_OPEN_WINDOW2,
hInstance, NULL);
    CreateWindow(_T("BUTTON"), _T("Выход"), WS_CHILD |
WS_VISIBLE | BS_PUSHBUTTON, 100, 200, 250, 30, hwnd,
(HMENU)ID_BTN_ACTION2, hInstance, NULL);

    ShowWindow(hwnd, nCmdShow);
    UpdateWindow(hwnd);

    while (GetMessage(&msg, NULL, 0, 0)) {
        if (!IsDialogMessage(hChildWnd, &msg)) {
            TranslateMessage(&msg);
            DispatchMessage(&msg);
        }
    }
    return (int)msg.wParam;
}
```

Листинг 15 – Функция «WinMain»

5.3 Алгоритм сохранения настроек игры

Данный алгоритм (листинг 16) сохраняет выбранные пользователем настройки игры после нажатия кнопки «Сохранить». Имеются глобальные переменный «Diff», «Size» и «numAlgor». В первую переменную сохраняется введенное значение из поля «Сложность игры». Аналогично для второй переменной сохраняется значение из поля «Размеры игры». Для третьей переменной сохраняется значение из выпадающего списка. После сохранения пользователю во всплывающем окне выводится информация о сохраненных значениях сложности и размеров поля игры.

```
switch (LOWORD(wParam)) {
        case ID_BTN_EXIT:
            SendMessage(hChildWnd2, WM_CLOSE, 0, 0);
            break;
        case ID_BTN_SAVE: {

            TCHAR diffStr[10], sizeStr[10];
            GetWindowText(hwndDiffEdit, diffStr, 10);
            GetWindowText(hwndSizeEdit, sizeStr, 10);
```

```
            Diff = _tstoi(diffStr);
            Size = _tstoi(sizeStr);
            numAlgor= SendMessage(hwndAlgorithmCombo,
CB_GETCURSEL, 0, 0);


            TCHAR message[100];
            if (Diff > 0 && Size > 0) {
                _stprintf_s(message, _T("Значения сохранены:
Сложность = %d, Размеры поля = %d"), Diff, Size);
            }
            else {
                _stprintf_s(message, _T("Некорректные значения.
Введите положительные числа."));
            }
            MessageBox(hChildWnd2, message, _T("Сохранение
настроек"), MB_OK | MB_ICONINFORMATION);
            break;
            }
        }
```

Листинг 16 – Алгоритм сохранения настроек игры

Алгоритм вычисления скорости работы алгоритмов генерации поля и использование ими памяти

Данный алгоритм (листинг 19) позволяет вычислить, за какое время выполнится функция «mains» (листинг 17), которая принимает в параметрах размеры игрового поля и номер используемого алгоритма. С помощью оператора «if» используется один из 3 алгоритмов генерации игрового поля.

```
vector<vector<long int>> mains(int size,int numAlgor) {
    int h = size;
    vector<vector<long int>> arrFull(h, vector<long int>(h, 0));
    if (numAlgor == 0) {
        backtracking(arrFull, h);
    }
    if (numAlgor == 1) {
        arrFull = generateDancingLinks(h);

    }
    if (numAlgor == 2) {
        arrFull = solveWithGeneticAlgorithm( 2, 1, h);
    }
    return arrFull;
}
```

Листинг 17 – Функция «mains»

Функция «QueryPerformanceCounter» получает текущее время высокоточного таймера системы и сохраняет его в переменной. Также эта функция получает время после работы функции «mains». Разница двух переменный и будет временем работы алгоритма.

Для вычисления количества используемой памяти используется функция «getTotalMemoryUsage» (листинг 18). С помощью функции «GetCurrentProcess» мы получаем информацию об использовании памяти данным процессом и возвращаем ненулевое значение.

```
long long getTotalMemoryUsage() {
    PROCESS_MEMORY_COUNTERS pmc;
    if (GetProcessMemoryInfo(GetCurrentProcess(), &pmc,
sizeof(pmc))) {
        return static_cast<long long>(pmc.WorkingSetSize);
    }
    else {
        return 0LL; // Если не удалось получить информацию,
возвращаем 0
    }
}
```

Листинг 18 – Функция «getTotalMemoryUsage»

Далее полученные значения выводятся в нужном формате в всплывающем окне.

```
        QueryPerformanceCounter(&startTime);
        MemoryUsageBefore= getTotalMemoryUsage();
        otherData = mains( Size,numAlgor);
        MemoryUsageAfter= getTotalMemoryUsage();
        MemoryUsageTotal = MemoryUsageAfter - MemoryUsageBefore;
        QueryPerformanceCounter(&endTime);
        QueryPerformanceFrequency(&freq);
        elapsedSeconds = static_cast<double>(endTime.QuadPart -
startTime.QuadPart) / freq.QuadPart;
        _stprintf_s(bufferTimeCreate, _T("Время выполнения: %.6f
секунд.Использование памяти: %d байт"), elapsedSeconds,
MemoryUsageTotal);
        MessageBox(hMainWnd, bufferTimeCreate, _T("Информация"),
MB_OK);
```

Листинг 19 – Алгоритм вычисления скорости работы алгоритмов генерации поля и использование ими памяти

5.4 Алгоритм проверки игрового поля

В окне игры имеется кнопка «Проверить», с помощью которой проверяется правильность заполнения пустых ячеек пользователем (листинг 20). Они сравниваются с ячейками вектора otherData, в котором хранятся все начальные ячейки игрового поля. В случае успешной проверки, таймер останавливается и это значение сравнивается с лучшим. Если время меньше, то оно записывается в текстовый документ для дальнейшего вывода в новой игре. Далее во всплывающем окне выводится информация об успешном заполнении игрового поля и время пользователя.

```
case IDC_BUTTON_CHECK: {
            bool isValid = true;
            int row = 0;
            int col = 0;
            int i=0, j=0;
            for ( row = 0; row < Size; row++) {
                for (col = 0; col < Size; col++) {
                    TCHAR buffer[10];
             GetWindowText(hEditControls[row][col], buffer, 10);
                    int value = _wtoi(buffer);
                    if (value != int(otherData[row][col])) {
                        isValid = false;
                        break;
                    }
                }
                if (!isValid) break;
            }
            if (isValid) {
                KillTimer(hChildWnd, g_timerID);
                elapsedTime = g_elapsedSeconds;
                if (elapsedTime < bestTime || bestTime == 0.0) {
                    writeBestTime(Diff, Size, elapsedTime);
                    bestTime = elapsedTime;
                }
                TCHAR message[200];

                wsprintf(message, _T("Поле заполнено правильно,
молодец! Ваше время: %02d:%02d:%02d"),
                    g_elapsedSeconds / 3600, (g_elapsedSeconds %
3600) / 60, g_elapsedSeconds % 60);
```

```
                MessageBox(hChildWnd, message, L"Проверка",
MB_OK | MB_ICONINFORMATION);
                EnableWindow(hButtonCheck, FALSE);
            }
            else {
                MessageBox(hMainWnd, L"Поле содержит ошибки:(",
L"Проверка", MB_OK);
            }
            break;
            }
```

Листинг 20 – Алгоритм проверки игрового поля

6 Демонстрация работы приложения

В данном разделе мы рассмотрим работу всего приложения: используем различные настройки, сыграем в игру и посмотрим на практике как работают алгоритмы генерации поля.

При запуске программы появляется главное меню (рисунок 4), в котором есть 3 кнопки: «Играть», «Настройки игры» и «Выход».

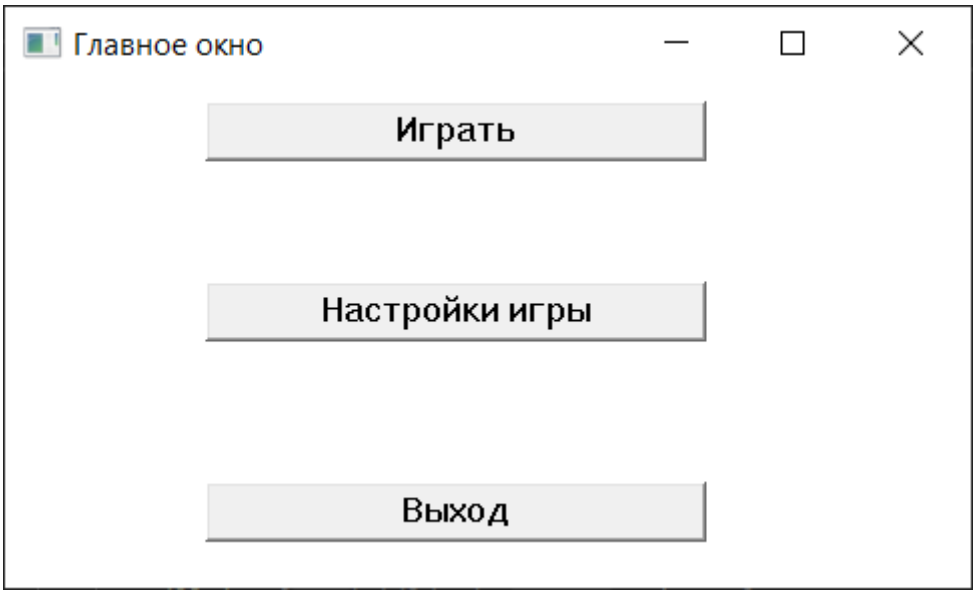

Рисунок 4 – Главное окно приложения

При запуске приложения изначально задается сложность 1, размеры поля 9 на 9 клеток и выбран первый алгоритм – Back tracking. При нажатии на кнопку «Играть» начинается генерация игрового поля при начальных параметрах или заданных пользователем. После завершения генерации появляется всплывающее окно (рисунок 5), в котором выводится информация, сколько времени потребовалось алгоритму для генерации игрового поля и сколько было использовано памяти.

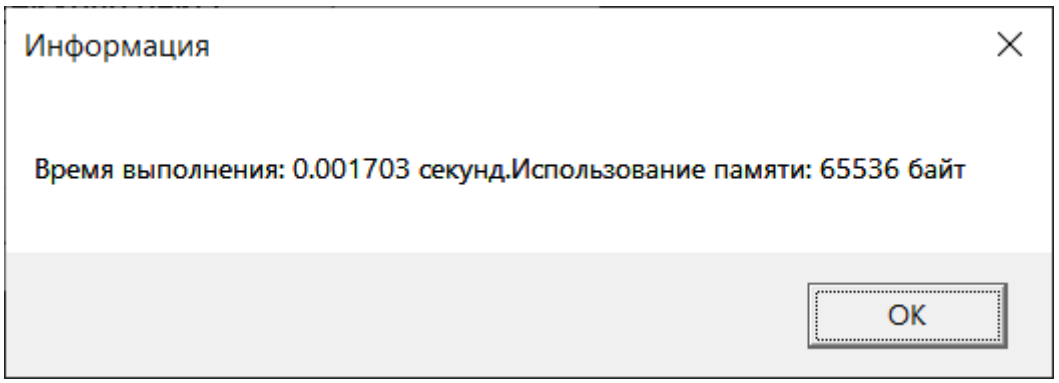

Рисунок 5 – Окно с информацией о времени работы алгоритма и использовании памяти

После нажатия кнопки «ОК», появляется окно «Игра» (рисунок 6). В данном окне присутствует само игровое поле, правила игры, информация о сложности игры, размерах игрового поля, таймер игры, лучшее время и 2 кнопки: «Проверить», «В главное меню». Описание работы кнопки было описано выше в пункте 5.5. При нажатии кнопки «В главное меню», данное окно закрывается и открывается главное окно.

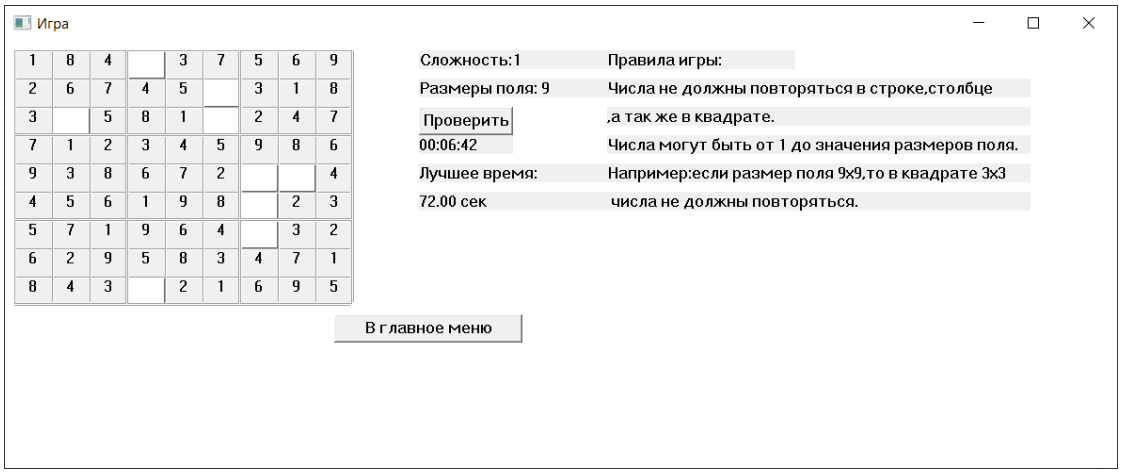

Рисунок 6 – Окно «Игра»

Игровым полем является матрица, в которой можно редактировать пустые клетки. Остальные клетки нельзя менять, так как иначе пользователь может случайно изменить начальную клетку. Каждую секунду таймер обновляется и таким образом выводит время игры пользователя.

Заполним пустые ячейки случайными числами (рисунок 7) и нажмем кнопку проверить, чтобы увидеть какой будет результат (рисунок 8). Программа должна вывести информацию о том, что поле заполнено неправильно.

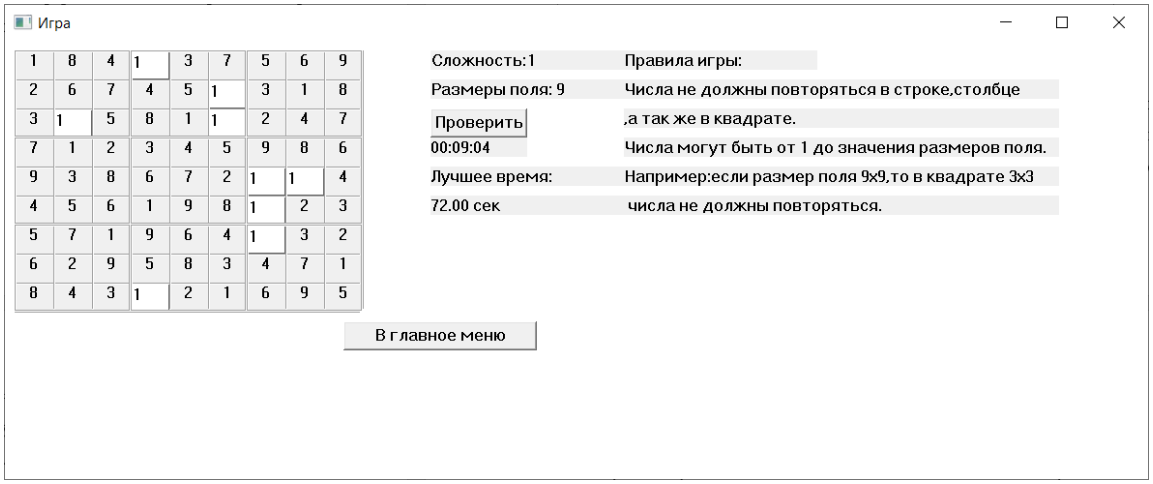

Рисунок 7 – Заполненное поле числами

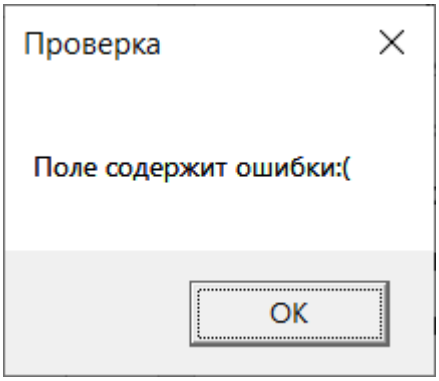

Рисунок 8 – Результат проверки заполнения поля

Как мы видим, алгоритм работает правильно. Теперь решим головоломку (рисунок 9) и произведем проверку еще раз (рисунок 10).

Рисунок 9 – Правильно заполненное поле

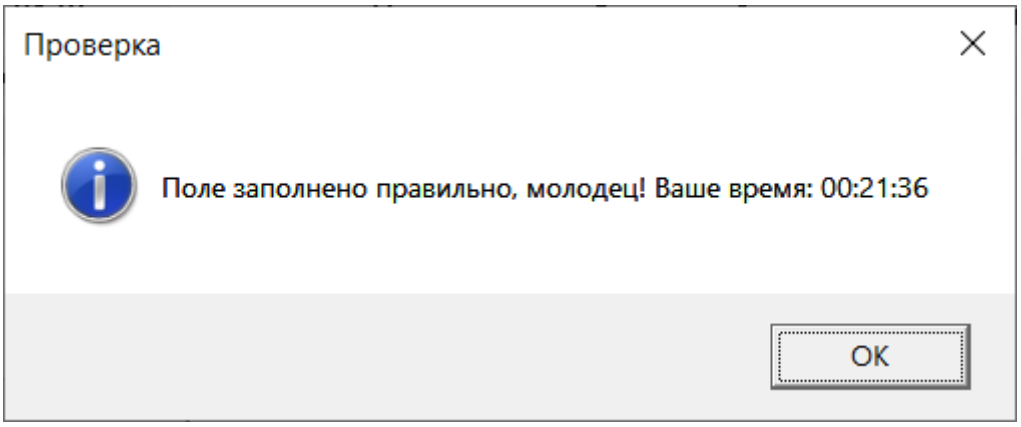

Рисунок 10 - Результат проверки правильно заполненного поля

После правильной проверки, таймер останавливается и кнопка «Проверить» становиться неактивной (рисунок 11).

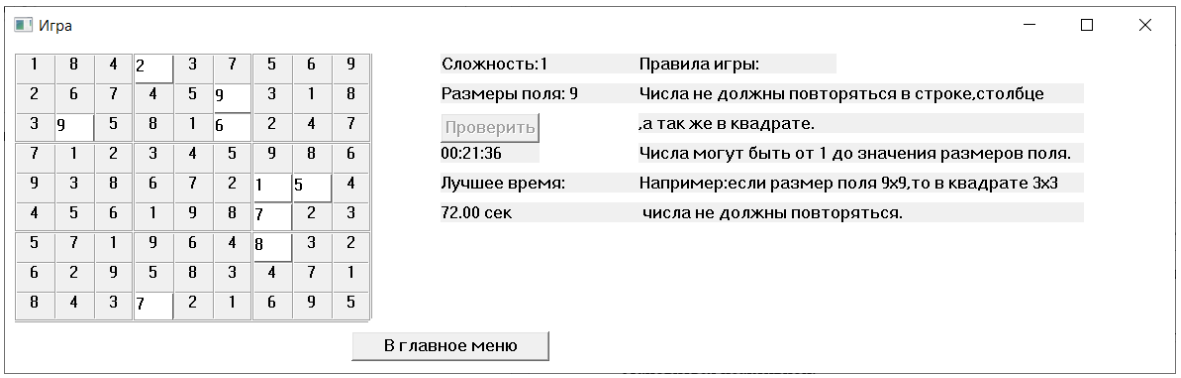

Рисунок 11 – Неактивная кнопка «Проверить»

Изменим настройки игрового поля: изменим сложность игры на 5, размеры поля увеличим до 16 на 16 клеток и выберем Генеративный алгоритм (рисунки 12 – 13). Далее посмотрим за какое время алгоритм сгенерирует игровое поле и сколько потребуется памяти на это (рисунок 14). Затем увидим новое сгенерированное поле (рисунок 15).

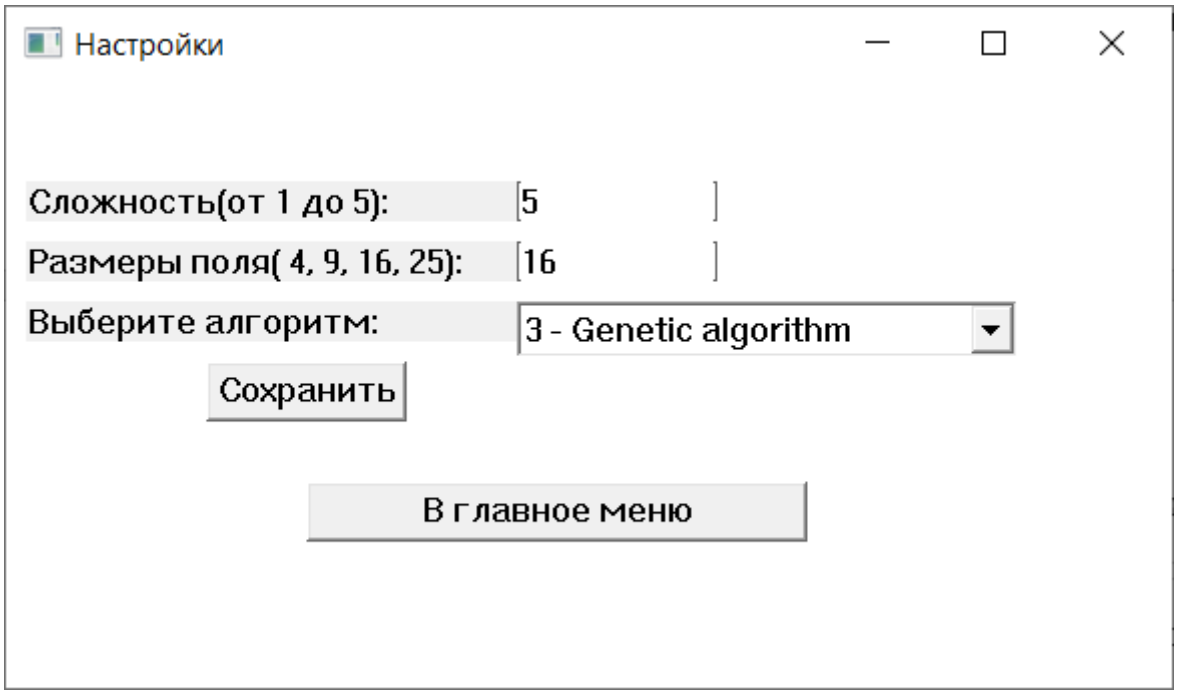

Рисунок 12 – Настройка игры с использованием своим параметров

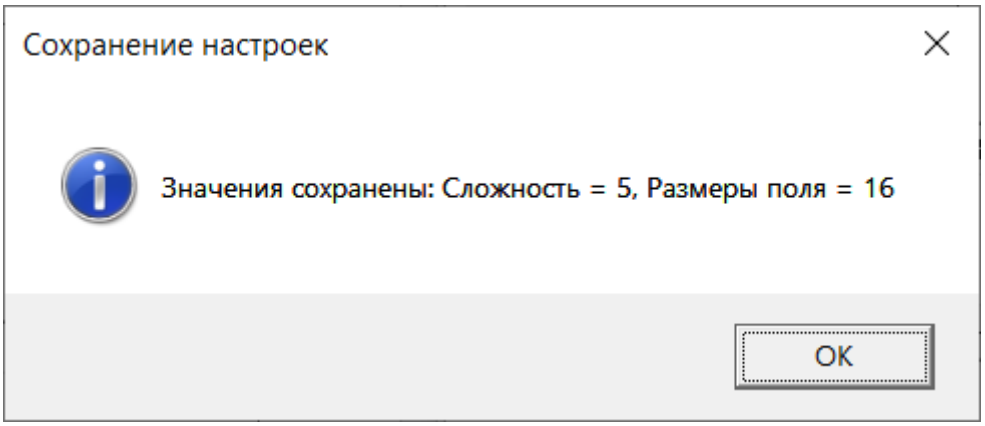

Рисунок 13 – Сохраненные настройки игры

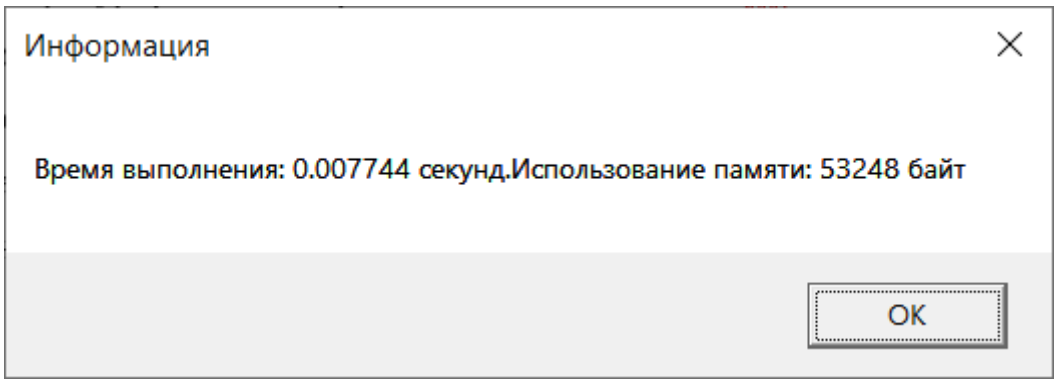

Рисунок 14 – Информация о работе алгоритма

Рисунок 15 – Сгенерированное поле 16 на 16

7 Анализ алгоритмов генерации игрового поля

В данном разделе проведем анализ работы алгоритмов. Будем сравнивать время, за которое алгоритмы генерируют различные поля, и использование памяти ими.

Сложность игры установим на 1, а размеры поля будут 4 на 4, 9 на 9 и 25 на 25. В таком порядке будем запускать алгоритмы. Каждый алгоритм будет запускать 5 раз и в конце высчитаем среднее значение времени и использования памяти. На основе этих данных и будем сравнивать алгоритмы. Запустим программу и выставим настройки сложности и размеры поля 4 на 4 (рисунки 16 – 17). Первым алгоритмом будет алгоритм Back tracking (рисунки 18 – 19).

Рисунок 16 – Выставленные настройки

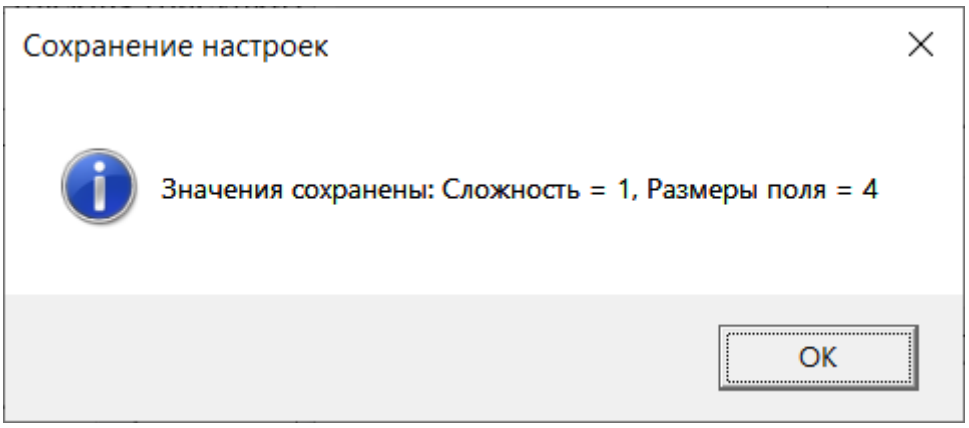

Рисунок 17 – Сохраненные настройки

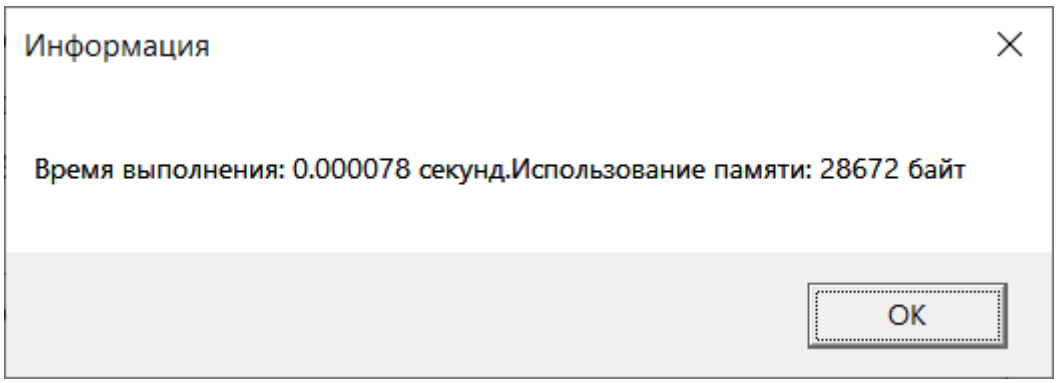

Рисунок 18 – Первый пример запуска алгоритма Back tracking

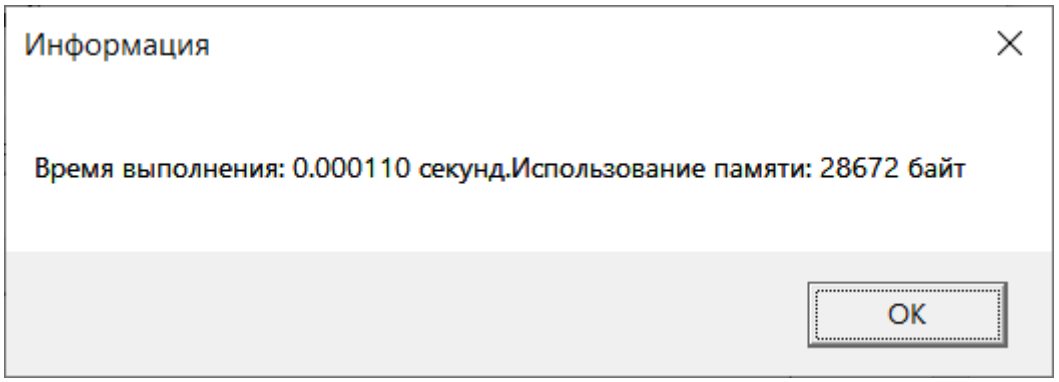

Рисунок 19 – Второй пример запуска алгоритма Back tracking

Следующим алгоритмом будет алгоритм Dancing links (рисунки 20 – 21).

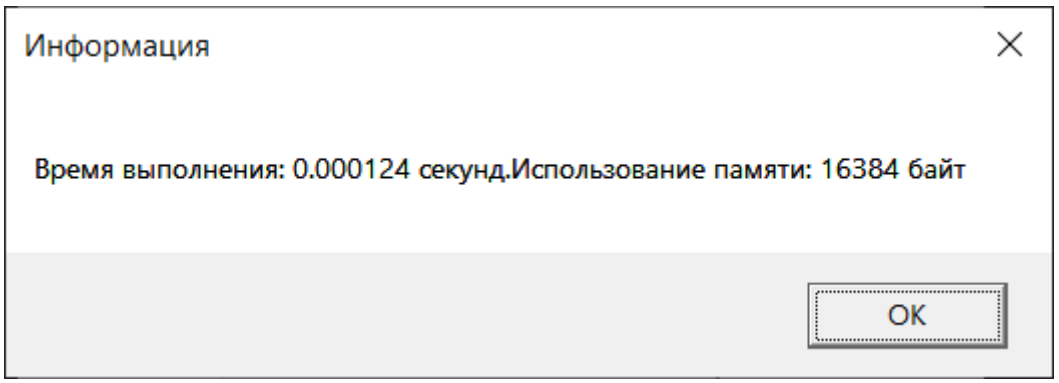

Рисунок 20 – Первый пример запуска алгоритма Dancing links

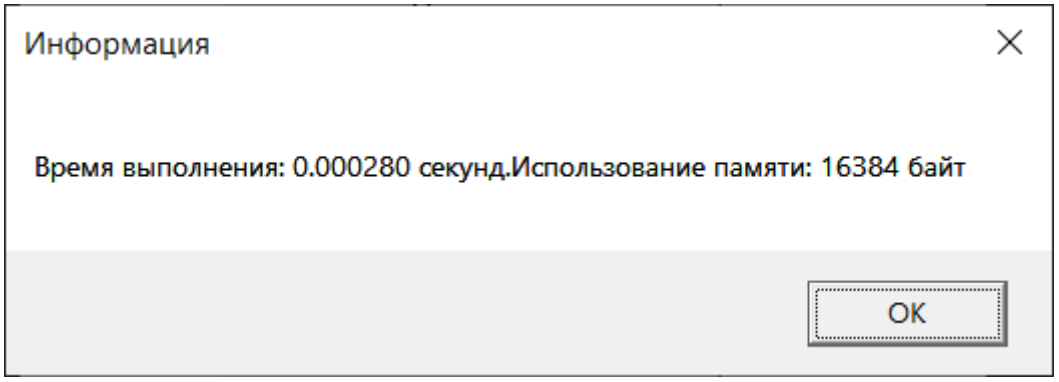

Рисунок 21 – Второй пример запуска алгоритма Dancing links

Теперь запускаем последний Генетический алгоритм (рисунки 22 – 23).

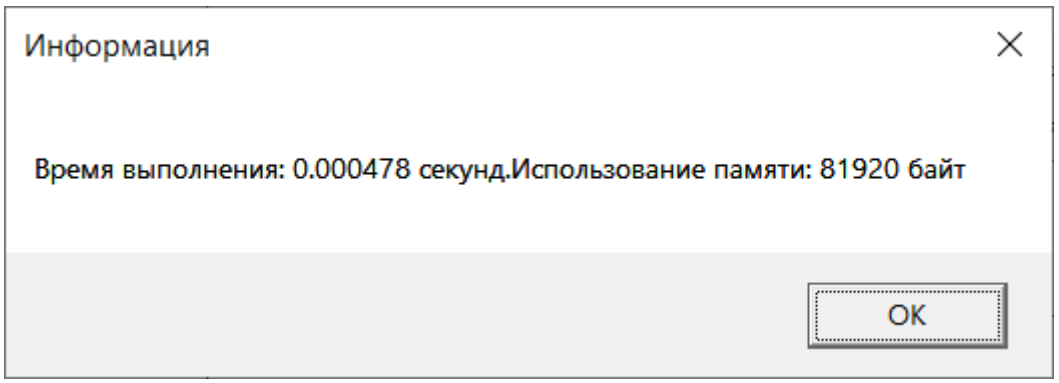

Рисунок 22 – Первый пример запуска Генетического алгоритма

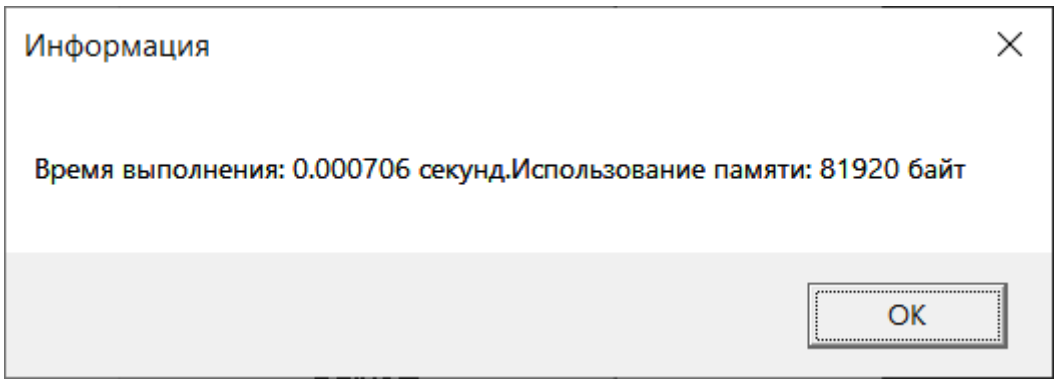

Рисунок 23 – Второй пример запуска Генетического алгоритма

Все полученные данные записываем в таблицы 1 – 2 и в конце рассчитываем средние значения.

Таблица 1 – Время работы алгоритмами Back tracking, Dancing links и Генетический при размерах поля 4 на 4

В секундах

Номер запуска	Back tracking	Dancing links	Генетический
1	0.000078	0.000124	0.000478
2	0.000110	0.000280	0.000706
3	0.000061	0.000231	0.000662
4	0.000100	0.000121	0.000600
5	0.000071	0.000278	0.000388
Среднее значение	0.000084	0.000207	0.000567

Таблица 2 – Использование памяти алгоритмами Back tracking, Dancing links и Генетический при размерах поля 4 на 4

В байт

Номер запуска	Back tracking	Dancing links	Генетический
1	28672	16384	81920
2	28672	16384	81920
3	28672	16384	90112

Продолжение таблицы 2

Номер запуска	Back tracking	Dancing links	Генетический
4	32768	16384	73728
5	28672	16384	77824
Среднее значение	29491	16384	81100

Изменим размеры поля до классических 9 на 9 (рисунки 24 – 25) и продолжим запускать алгоритмы.

Рисунок 24 – Именные настройки игры

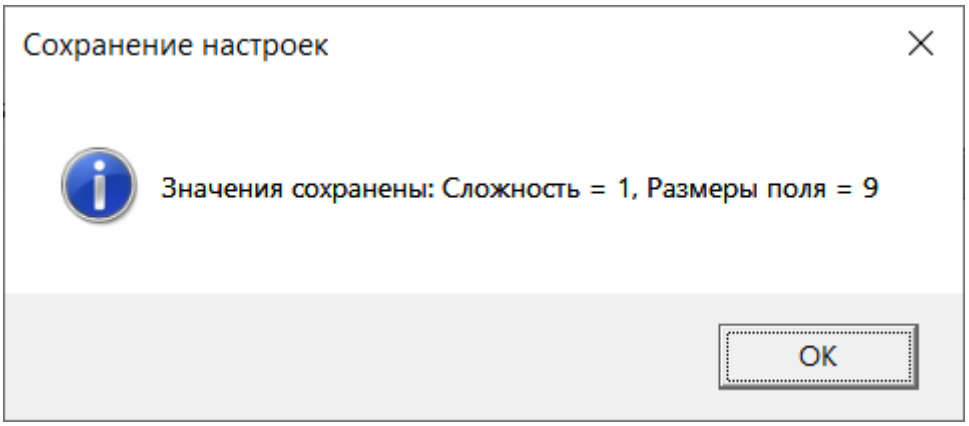

Рисунок 25 – Сохранение настроек игры

Первым алгоритмом также будет Back tracking (рисунки 26 – 27).

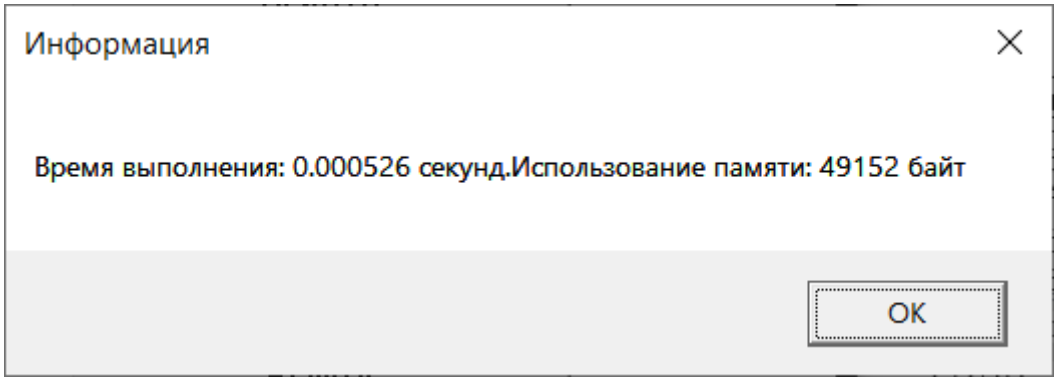

Рисунок 26 – Первый пример запуска алгоритма Back tracking

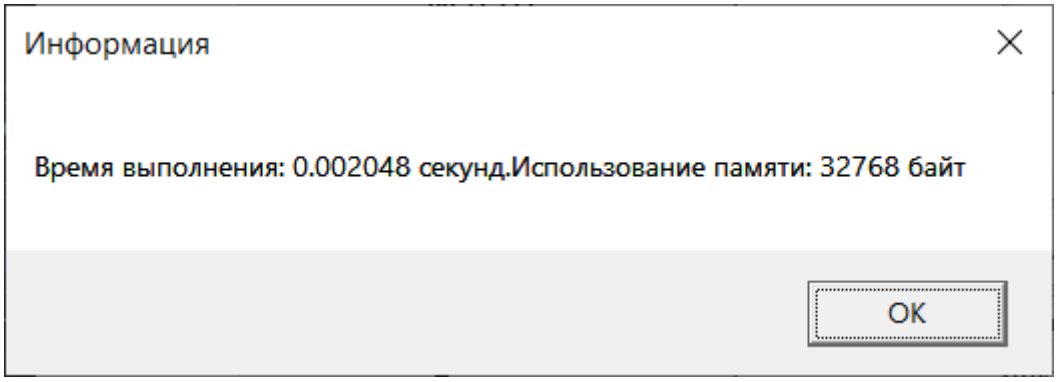

Рисунок 27 – Второй пример запуска алгоритма Back tracking

Вторым алгоритмом также будет Dancing links (рисунки 28 – 29).

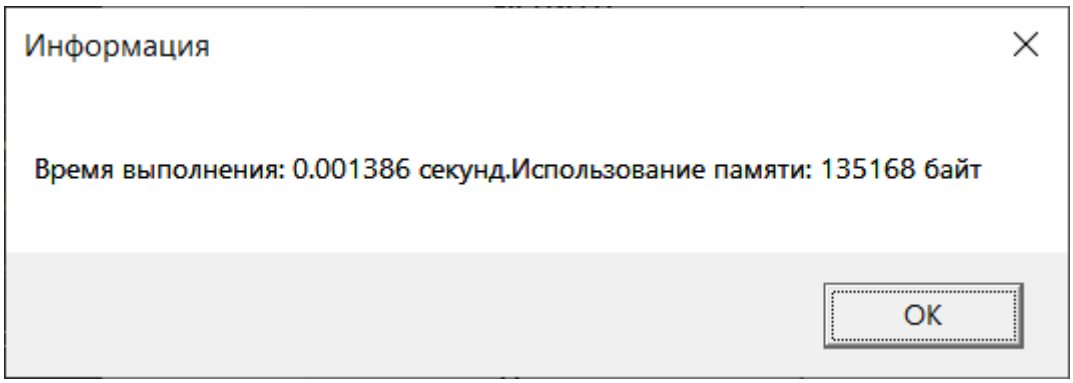

Рисунок 28 – Первый пример запуска алгоритма Dancing links

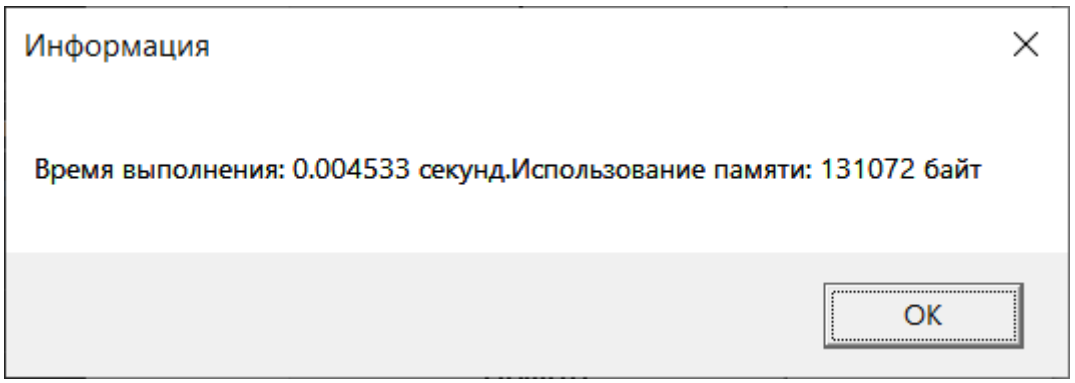

Рисунок 29 – Второй пример запуска алгоритма Dancing links

Теперь запуски последнего Генетического алгоритма (рисунки 30 – 31).

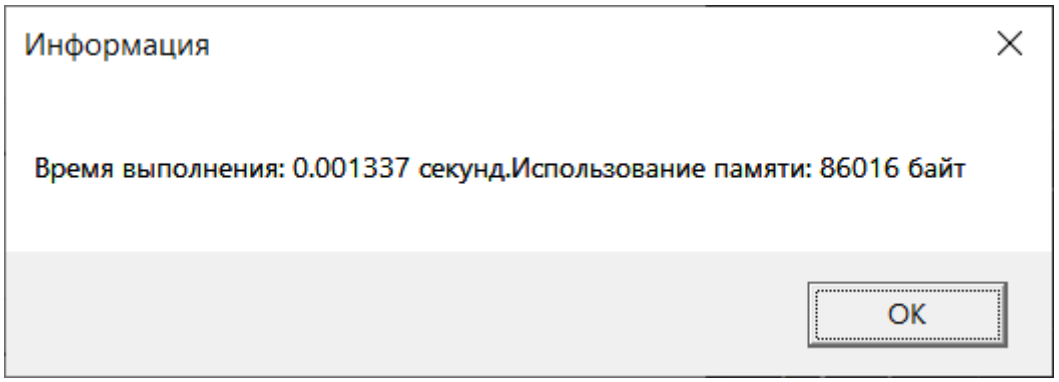

Рисунок 30 – Первый пример запуска Генетического алгоритма

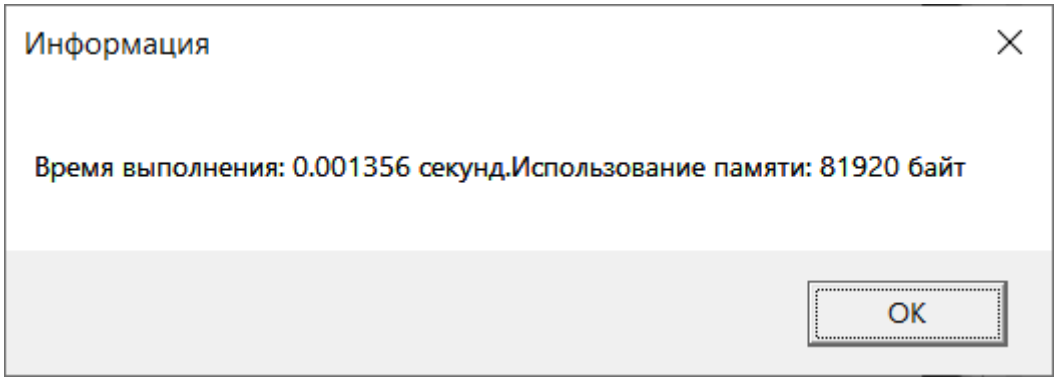

Рисунок 31 – Второй пример запуска Генетического алгоритма

Теперь полученные данные запишем в таблицы 3 – 4 и посчитаем средние значения.

Таблица 3 – Время работы алгоритмами Back tracking, Dancing links и Генетический при размерах поля 9 на 9

В секундах

Номер запуска	Back tracking	Dancing links	Генетический
1	0.000526	0.001386	0.001337
2	0.002048	0.004533	0.001356
3	0.000190	0.002010	0.003404
4	0.001814	0.003135	0.001290
5	0.000273	0.001877	0.002114
Среднее значение	0.000970	0.002588	0.001900

Таблица 4 – Использование памяти алгоритмами Back tracking, Dancing links и Генетический при размерах поля 9 на 9

В байт

Номер запуска	Back tracking	Dancing links	Генетический
1	49152	135168	86016
2	32768	131072	81920
3	69632	135168	135168
4	57344	139264	86016
5	53248	139264	114688
Среднее значение	52429	135987	100761

Для последнего анализа увеличим размеры поля до 25 на 25 (рисунки 32 – 33).

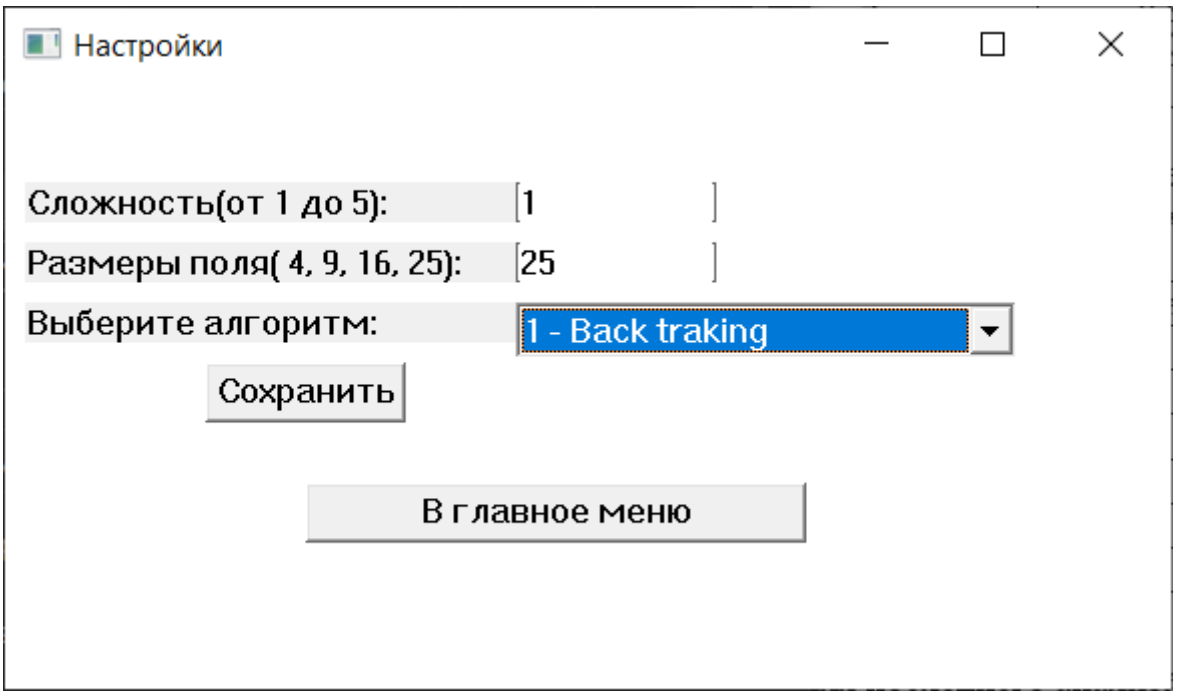

Рисунок 32 – Измененные размеры поля

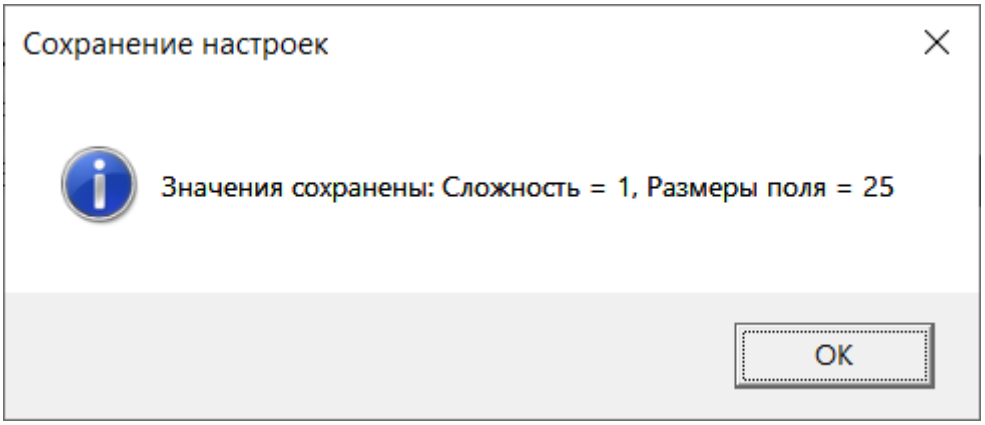

Рисунок 33 – Сохраненные настройки

При данных настройках алгоритм Back tracking тратит большое количество времени для нахождения решения. Также программа перестает работать и в следствии этого ее анализ невозможен. Алгоритмы Dancing links и Генетический могут сгенерировать такое большое игровое поле. Поэтому дальше будем проверять только их (рисунки 34 – 37).

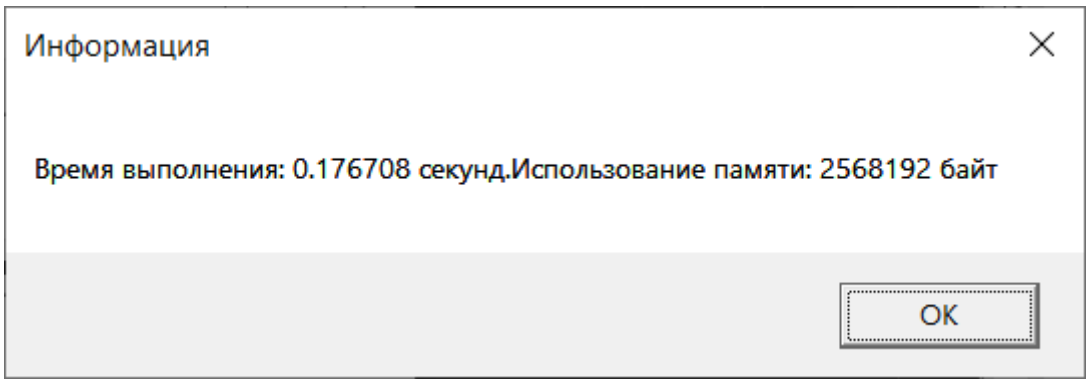

Рисунок 34 – Первый пример запуска алгоритма Dancing links

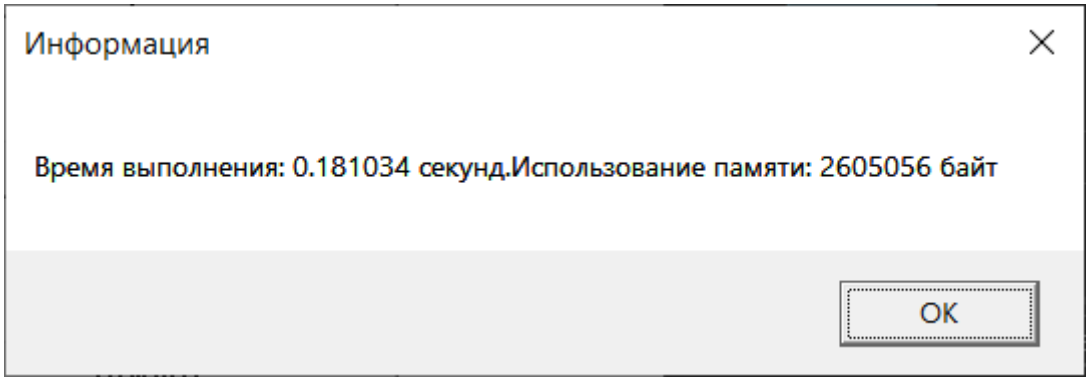

Рисунок 35 – Второй пример запуска алгоритма Dancing links

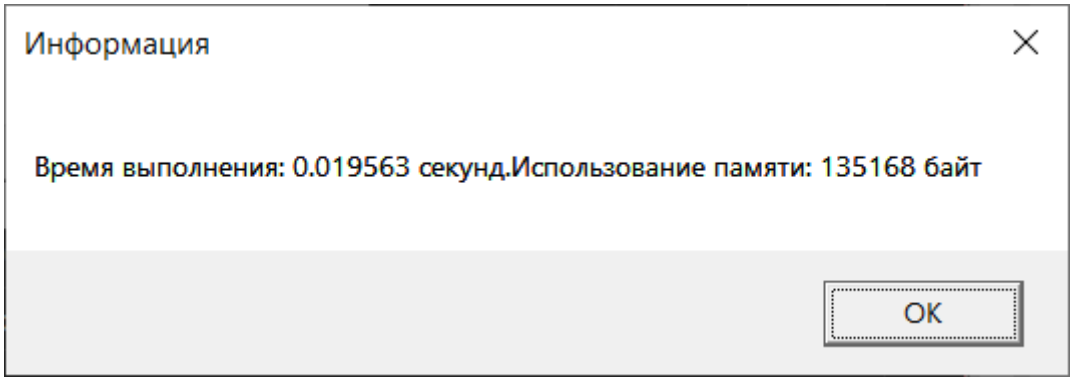

Рисунок 36 – Первый пример запуска Генетического алгоритма

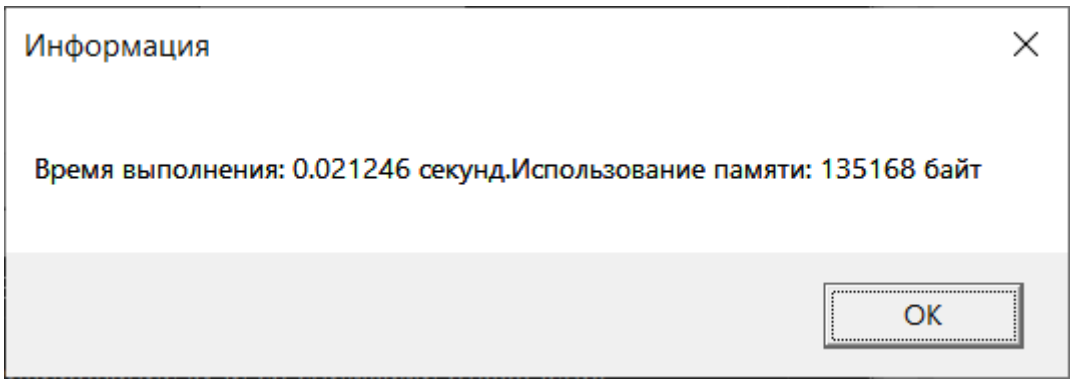

Рисунок 37 – Второй пример запуска Генетического алгоритма

Полученные данные запишем в таблицы 5 – 6.

Таблица 5 – Время работы алгоритмами Back tracking, Dancing links и Генетический при размерах поля 25 на 25

В секундах

Номер запуска	Dancing links	Генетический
1	0.176708	0.019563
2	0.181034	0.021246
3	0.224745	0.021069
4	0.183448	0.020306
5	0.349694	0.042215
Среднее значение	0.223125	0.024879

Таблица 6 – Использование памяти алгоритмами Back tracking, Dancing links и Генетический при размерах поля 25 на 25

В байт

Номер запуска	Dancing links	Генетический
1	2568192	135168
2	2605056	135168
3	2580480	188416
4	2596864	139264

Продолжение таблицы 6

Номер запуска	Dancing links	Генетический
5	2596864	139264
Среднее значение	2589491	147456

Из полученных данных мы можем видеть, насколько различаются алгоритмы, которые выполняют один и тот же процесс. Каждый алгоритм уникален и больше подходит для конкретных задач. Если игровое поле маленькое (размеры 4 на 4), тогда алгоритм Dancing links подходит лучше всего за счет своей скорости и малого использования памяти. Однако данный алгоритм сложен в реализации и понимании. Алгоритм Back tracking лучше всего себя показывает при классических настройках игрового поля из – за его простоты. Генетический алгоритм уступает по показателям двум другим при размерах поля 4 на 4. С увеличением данных данный алгоритм начинает работать лучше других. И при размерах больше 9 на 9 клеток, алгоритм работает быстро и не тратит большое количество памяти. Данный алгоритм сложен в реализации. Если из этих алгоритмов выбирать один, то лучшем выбором будет Генетический алгоритм за счет своей гибкости, скорости работы и использования памяти.

ЗАКЛЮЧЕНИЕ

В ходе выполнения выпускной квалификационной работы было разработано игровое приложение «Судоку», в котором было реализовано 3 алгоритма, с помощью которых генерируются игровые поля для игры. Также были решены следующие поставленные задачи:

изучены правила игры «Судоку», конкретно как должны располагаться числа на игровом поле и каким оно может быть по размерам;

были выбраны и реализованы алгоритмы для генерации игрового поля для дальнейшего анализа;

были реализованы алгоритмы и методы для работы всего приложения;

выполнен анализ алгоритмов для генерации игрового поля на основе скорости выполнения и использования памяти;

Разработанное приложение позволяет обычным пользователем запускать игру, содержащую только нужную информацию для игры, а также функционал позволяет настроить игровое поле так, как они хотят.

При проведении анализа мы увидели, с какой скоростью алгоритмы сгенерировали игровое поле и сколько потребовалось памяти на это им. Данная информация наглядно показала важность выбора алгоритмов и методов при проектировании различного ПО.

Данное приложение можно использовать как наглядный пример при обучении программистов в момент изучения тем, связанных с алгоритмами. На наглядном примере будет видна разница между алгоритмами, которые решают одну задачу. Эта информация может будущим разработчикам сделать ПО более эффективное и в некоторый случаях более стабильное.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Кнут, Дональд. Искусство программирования. Том 4А. Комбинаторные алгоритмы. Часть 1 / Дональд Кнут. — М.: Вильямс, 2016. — 864 с.
2. Фелинг, Джим. Совершенная судоку / Джим Фелинг. — М.: АСТ, 2019. — 288 с.
3. Прата, Стив. Алгоритмы на C++ / Стив Прата. — СПб.: Питер, 2018. — 688 с.
4. Петцольд, Чарльз. Программирование для Windows / Чарльз Петцольд. — СПб.: Питер, 2021. — 1584 с.
5. Страуструп, Бьерн. Язык программирования C++ / Бьерн Страуструп. — М.: Бином, 2018. — 1328 с.
6. Липпман, Стэнли Б., Лажойе, Жози, Му, Барбара Э. Язык программирования C++ / Стэнли Б. Липпман, Жози Лажойе, Барбара Э. Му. — М.: Вильямс, 2020. — 1120 с.
7. История Судоку [Электронный ресурс]. – URL: https://sudoku.com/ru/kak-igrat/istoria-sudoku/ (дата обращения: 15.05.2024).
8. История появления судоку. Кто первым придумал игру судоку [Электронный ресурс]. – URL: https://istorygames.info/golovolomki/istoriya-sudoku.html (дата обращения: 15.05.2024).
9. Алгоритм backtracking / Хабр [Электронный ресурс]. – URL: https://habr.com/ru/companies/otus/articles/746408/ (дата обращения: 16.05.2024).
10. Алгоритм X или что общего между деревянной головоломкой и танцующим Линком? / Хабр [Электронный ресурс]. – URL: https://habr.com/ru/articles/194410/ (Дата обращения: 16.05.2024).
11. Генетический алгоритм. Просто о сложном. / Хабр [Электронный ресурс]. – URL: https://habr.com/ru/articles/128704/ (Дата обращения: 18.05.2024).

12. Заголовочные файлы стандартной библиотеки C ++ [Электронный ресурс]. – URL: https://learn.microsoft.com/en-us/cpp/standard-library/cpp-standard-library-header-files?view=msvc-170 (Дата обращения: 18.05.2024).
13. The Algorithm X and the Dancing Links | NP-Incompleteness [Электронный ресурс]. - URL: https://kunigami.wordpress.com/2013/04/28/the-algorithm-x-and-the-dancing-links/170 (Дата обращения: 18.05.2024).
14. Панченко, Т.В. Генетические алгоритмы [Текст] / Т.В. Панченко. - М.: Бином. Лаборатория знаний, 2007. - 278 с.

ПРИЛОЖЕНИЕ А. Листинг файла functionsetpole.cpp

```
#include "functionsetpole.h"
#include <iostream>
// Функция для проверки, можно ли разместить число num в
заданной позиции
bool isSafe(std::vector<std::vector< long int>>& arr, int
row, int col, int num, int h) {
    // Проверяем строку
    for (int i = 0; i < h; i++) {
        if (arr[row][i] == num)
            return false;
    }

    // Проверяем столбец
    for (int i = 0; i < h; i++) {
        if (arr[i][col] == num)
            return false;
    }

    // Проверяем квадрат
    int sqrtH = sqrt(h);
    int boxRowStart = row - row % sqrtH;
    int boxColStart = col - col % sqrtH;

    for (int i = 0; i < sqrtH; i++) {
        for (int j = 0; j < sqrtH; j++) {
            if (arr[i + boxRowStart][j + boxColStart] == num
&& (i + boxRowStart != row || j + boxColStart != col))
                return false;
        }
    }

    return true;
}

// Функция для случайного заполнения части массива
void baseInit(std::vector<std::vector<long int>>& arr, int
h) {
    srand(time(NULL));
    int numCells = h * h / 3;
    for (int i = 0; i < numCells; i++) {
        int row = rand() % h;
        int col = rand() % h;
        if (arr[row][col] == 0) {
            int num = rand() % h + 1;
            if (isSafe(arr, row, col, num, h)) {
                arr[row][col] = num;
            }
        }
    }
}
```

```
    // Функция для заполнения оставшихся пустых ячеек
    bool solveRemaining(std::vector<std::vector<long int>>& arr,
int h) {
        int row = -1;
        int col = -1;
        bool isEmpty = true;

        // Находим следующую пустую ячейку
        for (int i = 0; i < h; i++) {
            for (int j = 0; j < h; j++) {
                if (arr[i][j] == 0) {
                    row = i;
                    col = j;
                    isEmpty = false;
                    break;
                }
            }
            if (!isEmpty)
                break;
        }

        // Если пустых ячеек нет, значит массив заполнен
        if (isEmpty)
            return true;

        // Пробуем разместить числа от 1 до h
        for (int num = 1; num <= h; num++) {
            if (isSafe(arr, row, col, num, h)) {
                arr[row][col] = num;
                if (solveRemaining(arr, h))
                    return true;
                arr[row][col] = 0;
            }
        }

        return false; // Если невозможно разместить число,
возвращаем false
    }
    // Функция для заполнения массива числами от 1 до h
    bool backtracking(std::vector<std::vector<long int>>& arr,
int h) {
        bool solved = false;

        std::vector<std::vector<long int>> temp(h,
std::vector<long int>(h, 0)); // Создаем временный массив

        baseInit(temp, h); // Заполняем случайно часть
временного массива

        if (solveRemaining(temp, h)) {
           arr = temp; // Копируем решение во внешний массив
           solved = true;
```

```
        }
        return solved;
    }
```

ПРИЛОЖЕНИЕ Б. Листинг файла dancingLinks.cpp

```
#include <random>
#include "dancingLinks.h"

// Структура узла в списке Dancing Links
struct Node;
struct Header;

std::vector<Node*> nodes;
std::mt19937 gen(std::random_device{}()); // Определение
генератора случайных чисел
struct Node {
    int row, col, value;
    Node* up, * down, * left, * right;
    Header* head;

    Node(int r, int c, int v) : row(r), col(c), value(v),
    up(nullptr), down(nullptr), left(nullptr),
right(nullptr), head(nullptr) {}
};

// Структура заголовка узла
struct Header {
    int size;
    Node* head;

    Header() : size(0), head(nullptr) {}
};

// Вспомогательная функция для связывания узла с заголовком
void linkNode(Header& header, Node* node) {
    node->head = &header;
    if (header.head == nullptr) {
        header.head = node->left = node->right = node;
    }
    else {
        Node* oldHead = header.head;
        node->left = oldHead;
        node->right = oldHead->right;
        oldHead->right->left = node;
        oldHead->right = node;
    }
    header.size++;
}

void removeNode(Node* node) {
    if (node == nullptr || node->left == nullptr || node-
>right == nullptr || node->head == nullptr) {
        return; // Проверка на корректность инициализации
узла и его указателей
    }
```

```
        node->left->right = node->right;
        node->right->left = node->left;
        node->head->size--;

        for (Node* curr = node->up; curr != nullptr && curr !=
node; curr = curr->up) {
            Node* temp = curr->left;
            while (temp != curr) {
                if (temp->head == nullptr) {
                    // Проверка на корректность указателя head
                    return;
                }
                temp->head->size--;
                temp = temp->left;
                if (temp == nullptr) {
                    // Проверка на корректность указателя left
                    return;
                }
            }
        }
    }

    void recoverNode(Node* node) {
        if (node == nullptr || node->left == nullptr || node-
>right == nullptr || node->head == nullptr) {
            return; // Проверка на корректность инициализации
узла и его указателей
        }

        if (node->up != nullptr) {
            for (Node* curr = node->up; curr != node; curr =
curr->up) {
                if (curr != nullptr && curr->left != nullptr) {
                    for (Node* temp = curr->left; temp != curr;
temp = temp->left) {
                        if (temp != nullptr && temp->head !=
nullptr) {
                            // Проверка на корректность
указателя head
                            temp->head->size++;
                        }
                    }
                }
            }
        }

        if (node->head != nullptr) {
            // Проверка на корректность указателя head
            node->head->size++;
            node->left->right = node;
            node->right->left = node;
        }
```

```
    }

    // Функция для заполнения двумерного вектора случайными
числами с использованием Dancing Links
    std::vector<std::vector<long int>> generateDancingLinks(int
h) {
        int size = h;
        int headerCount = 4 * size + 1; // Количество заголовков
(строки, столбцы, квадраты, дополнительный заголовок)

        std::vector<Header> headers(headerCount); // Вектор
заголовков

        // Проверка корректности инициализации заголовков
        for (const Header& header : headers) {
            if (header.head != nullptr) {
                // Обработка ошибки некорректной инициализации
заголовков
                return {}; // Возвращаем пустой вектор
            }
        }

        // Создаем узлы и связываем их с соответствующими
заголовками
        for (int row = 0; row < size; row++) {
            for (int col = 0; col < size; col++) {
                for (int value = 1; value <= size; value++) {
                    nodes.push_back(new Node(row, col, value));
// Создаем новый узел
                    Node* node = nodes.back();

                    // Связываем узел с заголовками строки,
столбца и квадрата
                    linkNode(headers[row], node);
                    linkNode(headers[size + col], node);
                    linkNode(headers[2 * size + (row / h) * h +
col / h], node);
                }
            }
        }

        // Создаем дополнительный заголовок и связываем его со
всеми узлами
        Header& rootHeader = headers.back();
        for (Node* node : nodes) {
            linkNode(rootHeader, node);
        }


        std::vector<std::vector<long int>> result(size,
std::vector<long int>(size, 0));
        std::uniform_int_distribution<long int> dist(1, size);
        for (int row = 0; row < size; row++) {
```

```
            for (int col = 0; col < size; col++) {
                std::vector<bool> used(size + 1, false); //
Вектор для отслеживания использованных чисел
                int value;
                do {
                    value = dist(gen);
                } while (used[value]); // Продолжаем генерацию,
пока не найдем уникальное число
                result[row][col] = value;
                used[value] = true;

                // Удаляем узлы, конфликтующие с выбранным
числом
                for (Node* node : nodes) {
                    if ((node->row == row || node->col == col ||
                        (node->row / h) * h + node->col / h ==
(row / h) * h + col / h)
                        && node->value != value) {
                        removeNode(node);
                    }
                }
            }
        }

        // Восстанавливаем удаленные узлы
        for (Node* node : nodes) {
            recoverNode(node);
        }

        return result;
    }
```

ПРИЛОЖЕНИЕ В. Листинг файла solveWithGeneticAlgorithm.cpp

```
#include "solveWithGeneticAlgorithm.h"
#include <random>
#include <algorithm>

using namespace std;

bool isSafe2(std::vector<std::vector< long int>>& arr, int
row, int col, int num, int h) {
    // Проверяем строку
    for (int i = 0; i < h; i++) {
        if (arr[row][i] == num)
            return false;
    }

    // Проверяем столбец
    for (int i = 0; i < h; i++) {
        if (arr[i][col] == num)
            return false;
    }

    // Проверяем квадрат
    int sqrtH = sqrt(h);
    int boxRowStart = row - row % sqrtH;
    int boxColStart = col - col % sqrtH;

    for (int i = 0; i < sqrtH; i++) {
        for (int j = 0; j < sqrtH; j++) {
            if (arr[i + boxRowStart][j + boxColStart] == num
&& (i + boxRowStart != row || j + boxColStart != col))
                return false;
        }
    }

    return true;
}

void fillRandomCells2(std::vector<std::vector<long int>>&
arr, int h) {
    srand(time(NULL)); // Инициализируем генератор случайных
чисел

    // Заполняем случайно примерно треть ячеек массива
    int numCells = h * h / 3;
    for (int i = 0; i < numCells; i++) {
        int row = rand() % h;
        int col = rand() % h;

            int num = rand() % h + 1;
            if (isSafe2(arr, row, col, num, h)) {
```

```
                    arr[row][col] = num;
                }

            }
        }

        // Функция оценки индивидуального решения
        int fitness(const vector<vector<long int>>& solution, int h)
{
            int conflicts = 0;

            // Проверка строк
            for (int row = 0; row < h; row++) {
                vector<bool> seen(h + 1, false);
                for (int col = 0; col < h; col++) {
                    if (solution[row][col] != 0) {
                        if (seen[solution[row][col]]) {
                            conflicts++;
                        }
                        else {
                            seen[solution[row][col]] = true;
                        }
                    }
                }
            }

            // Проверка столбцов
            for (int col = 0; col < h; col++) {
                vector<bool> seen(h + 1, false);
                for (int row = 0; row < h; row++) {
                    if (solution[row][col] != 0) {
                        if (seen[solution[row][col]]) {
                            conflicts++;
                        }
                        else {
                            seen[solution[row][col]] = true;
                        }
                    }
                }
            }

            // Проверка квадратов
            int sqrtH = static_cast<int>(sqrt(h));
            for (int sqRow = 0; sqRow < sqrtH; sqRow++) {
                for (int sqCol = 0; sqCol < sqrtH; sqCol++) {
                    vector<bool> seen(h + 1, false);
                    for (int row = sqRow * sqrtH; row < (sqRow + 1)
* sqrtH; row++) {
                        for (int col = sqCol * sqrtH; col < (sqCol +
1) * sqrtH; col++) {
                            if (solution[row][col] != 0) {
                                if (seen[solution[row][col]]) {
                                    conflicts++;
```

```
                            }
                            else {
                                seen[solution[row][col]] = true;
                            }
                        }
                    }
                }
            }

            return conflicts;
        }

        // Функция для создания начальной популации
        vector<vector<vector<long int>>> initPopulation(int size,
const vector<vector<long int>>& initialBoard, int h) {
            vector<vector<vector<long int>>> population(size);
            for (auto& solution : population) {
                solution = initialBoard;
            }
            return population;
        }

        // Функция для скрещивания двух решений
        vector<vector<long int>> crossover(const vector<vector<long
int>>& parent1, const vector<vector<long int>>& parent2, int h)
{
            vector<vector<long int>> child(h, vector<long int>(h,
0));
            int sqrtH = static_cast<int>(sqrt(h));

            // Копируем неизменные части из родительских решений
            for (int row = 0; row < h; row++) {
                for (int col = 0; col < h; col++) {
                    if (parent1[row][col] != 0) {
                        child[row][col] = parent1[row][col];
                    }
                    else if (parent2[row][col] != 0) {
                        child[row][col] = parent2[row][col];
                    }
                }
            }

            // Заполняем пустые ячейки, избегая повторений
            for (int row = 0; row < h; row++) {
                for (int col = 0; col < h; col++) {
                    if (child[row][col] == 0) {
                        vector<bool> seen(h + 1, false);
                        // Проверяем строку
                        for (int i = 0; i < h; i++) {
                            if (child[row][i] != 0) {
                                seen[child[row][i]] = true;
                            }
```

```
                    }
                    // Проверяем столбец
                    for (int i = 0; i < h; i++) {
                        if (child[i][col] != 0) {
                            seen[child[i][col]] = true;
                        }
                    }
                    // Проверяем квадрат
                    int sqRow = row / sqrtH;
                    int sqCol = col / sqrtH;
                    for (int i = sqRow * sqrtH; i < (sqRow + 1)
* sqrtH; i++) {
                        for (int j = sqCol * sqrtH; j < (sqCol +
1) * sqrtH; j++) {
                            if (child[i][j] != 0) {
                                seen[child[i][j]] = true;
                            }
                        }
                    }
                    // Находим первое незанятое число и
заполняем ячейку
                    for (int num = 1; num <= h; num++) {
                        if (!seen[num]) {
                            child[row][col] = num;
                            break;
                        }
                    }
                }
            }
        }

        return child;
    }
    // Функция для мутации решения
    void mutate(vector<vector<long int>>& solution, int h) {
        random_device rd;
        srand(time(NULL)); // Инициализируем генератор случайных
чисел
        mt19937 gen(rd());
        uniform_int_distribution<> rowDist(0, h - 1);
        uniform_int_distribution<> colDist(0, h - 1);
        uniform_int_distribution<long int> numDist(1, h);

        // Выбираем случайную ячейку для мутации
        int row = rowDist(gen);
        int col = colDist(gen);

        // Проверяем, можно ли безопасно изменить значение
ячейки
        vector<bool> seen(h + 1, false);
        bool canMutate = true;

        // Проверяем строку
```

```
        for (int i = 0; i < h; i++) {
            if (solution[row][i] != 0) {
                seen[solution[row][i]] = true;
            }
        }
        // Проверяем столбец
        for (int i = 0; i < h; i++) {
            if (solution[i][col] != 0) {
                seen[solution[i][col]] = true;
            }
        }
        // Проверяем квадрат
        int sqrtH = static_cast<int>(sqrt(h));
        int sqRow = row / sqrtH;
        int sqCol = col / sqrtH;
        for (int i = sqRow * sqrtH; i < (sqRow + 1) * sqrtH;
i++) {
            for (int j = sqCol * sqrtH; j < (sqCol + 1) * sqrtH;
j++) {
                if (solution[i][j] != 0) {
                    seen[solution[i][j]] = true;
                }
            }
        }

        // Если в строке, столбце или квадрате есть
повторяющиеся числа, отказываемся от мутации
        for (int num = 1; num <= h; num++) {
            if (seen[num]) {
                canMutate = false;
                break;
            }
        }

        // Выполняем мутацию, если это возможно
        if (canMutate) {
            long int newValue;
            do {
                newValue = rand() % h + 1;;
            } while (seen[newValue]);
            solution[row][col] = newValue;
        }
    }

    // Основной алгоритм
    vector<vector<long int>> solveWithGeneticAlgorithm( int
populationSize, int generations, int h) {
        std::vector<std::vector<long int>> arr(h,
std::vector<long int>(h, 0));
        fillRandomCells2(arr, h);
        vector<pair<int, int>> fitnessSolutions;
        vector<vector<vector<long int>>> population =
initPopulation(populationSize, arr, h);
```

```
        for (int generation = 0; generation < generations;
generation++) {
            // Оценка пригодности каждого решения
            for (int i = 0; i < populationSize; i++) {

fitnessSolutions.emplace_back(fitness(population[i], h), i);
            }
            sort(fitnessSolutions.begin(),
fitnessSolutions.end());
            // Выбор родителей для скрещивания
            vector<vector<vector<long int>>> newPopulation;
            for (int i = 0; i < populationSize / 2; i++) {
                int parent1Index = fitnessSolutions[i].second;
                int parent2Index =
fitnessSolutions[populationSize - i - 1].second;

newPopulation.push_back(crossover(population[parent1Index],
population[parent2Index], h));

newPopulation.push_back(crossover(population[parent2Index],
population[parent1Index], h));
            }
            // Мутация
            for (auto& solution : newPopulation) {
                mutate(solution, h);
            }
            population = move(newPopulation);
        }
        // Возврат решения с наилучшим значением пригодности
        int bestIndex = fitnessSolutions[0].second;
        return population[bestIndex];
    }
```

ПРИЛОЖЕНИЕ Г. Листинг файла functiondeletecells.cpp

```
#include "functiondeletecells.h"
#include <random>

std::vector<std::vector<longint>>
deleteRandomCells(std::vector<std::vector<long int>>& arrF, int
h,int d) {
        srand(time(NULL)); // Инициализируем генератор случайных
чисел
        std::vector<std::vector<long int>> arr= arrF;
        std::random_device rd;
        std::mt19937 gen(rd());
        std::uniform_int_distribution<> dis(0, h * h - 1);
        int numCells = d * h ;
        if (h == 4) {
            numCells = 10;
        }
        for (int i = 0; i < numCells; i++) {
            int index = dis(gen);
            int row = index / h;
            int col = index % h;
            if (arr[row][col] != 0) {
                arr[row][col] = 0;
            }
            else {
                i--;
            }
        }
        return arr;
    }
```

ПРИЛОЖЕНИЕ Д. Листинг файла Diplom.cpp

```
#include "functionsetpole.h"
#include "functiondeletecells.h"
#include "dancingLinks.h"
#include "solveWithGeneticAlgorithm.h"

using namespace std;

vector<vector<long int>> mains(int size,int numAlgor) {
    int h = size;
    vector<vector<long int>> arrFull(h, vector<long int>(h,
0));

    if (numAlgor == 0) {
        backtracking(arrFull, h);
    }
    if (numAlgor == 1) {
        arrFull = generateDancingLinks(h);

    }
    if (numAlgor == 2) {
        arrFull = solveWithGeneticAlgorithm( 2, 1, h);
    }
    return arrFull;
}
```

ПРИЛОЖЕНИЕ Е. Листинг файла WindowsProject2.cpp

```
#include <windows.h>
#include <tchar.h>
#include <vector>
#include <fstream>
#include <sstream>
#include <Psapi.h>
#include "Resource.h"
#include "Diplom.h"
#include "functiondeletecells.h"

using namespace std;

#define ID_BTN_OPEN_WINDOW 9001
#define ID_BTN_ACTION2 9002
#define ID_BTN_OPEN_WINDOW2 9003
#define ID_BTN_EXIT 2
#define ID_BTN_SAVE 3
#define IDC_BUTTON_CHECK 4
#define CELL_WIDTH 40
#define CELL_HEIGHT 30
UINT_PTR g_timerID = 0;

LRESULT CALLBACK MainWndProc(HWND hwnd, UINT msg, WPARAM
wParam, LPARAM lParam);
LRESULT CALLBACK SecondWndProc(HWND hChildWnd, UINT msg,
WPARAM wParam, LPARAM lParam);
LRESULT CALLBACK ThirdWndProc(HWND hChildWnd2, UINT msg,
WPARAM wParam, LPARAM lParam);

std::vector<std::vector<long int>> matrixData;
std::vector<std::vector< long int>> otherData;

int Diff = 1, Size = 9, numAlgor = 0, bestDifficulty =
0,bestSize = 0;
int screenWidth = GetSystemMetrics(SM_CXSCREEN);
int screenHeight = GetSystemMetrics(SM_CYSCREEN);
RECT windowRect = { 0, 0, 500, 300 };

int windowWidth = windowRect.right - windowRect.left;
int windowHeight = windowRect.bottom - windowRect.top;
int windowX = (screenWidth - windowWidth) / 2;
int windowY = (screenHeight - windowHeight) / 2;

int** userInputData;
HWND** hEditControls = nullptr;
int g_elapsedSeconds = 0;
HWND hTimerControl = nullptr;
double bestTime = 0.0;
```

```
    long long MemoryUsageBefore,
MemoryUsageAfter,MemoryUsageTotal;

    long long getTotalMemoryUsage() {
        PROCESS_MEMORY_COUNTERS pmc;
        if (GetProcessMemoryInfo(GetCurrentProcess(), &pmc,
sizeof(pmc))) {
            return static_cast<long long>(pmc.WorkingSetSize);
        }
        else {
            return 0LL; // Если не удалось получить информацию,
возвращаем 0
        }
    }

    double readBestTime(int difficulty, int fieldSize) {
        string filename = "best_times.txt";
        ifstream file(filename);
        if (!file) {
            return 0.0;
        }

        string line;
        while (getline(file, line)) {
            istringstream iss(line);
            int diff1, size1;
            double time;
            if (iss >> diff1 >> size1 >> time && diff1 ==
difficulty && size1 == fieldSize) {
                file.close();
                return time;
            }
        }
        file.close();
        return 0.0;
    }

    void writeBestTime(int difficulty, int fieldSize, double
time) {
        string filename = "best_times.txt";
        ofstream file(filename, ios::app);
        if (file) {
            file << difficulty << " " << fieldSize << " " <<
time << endl;
            file.close();
        }
    }

    LRESULT CALLBACK MainWndProc(HWND hwnd, UINT umsg, WPARAM
wParam, LPARAM lParam) {
        HWND hChildWnd, hChildWnd2;
```

```
        static HINSTANCE hInstance;
        AdjustWindowRect(&windowRect, WS_OVERLAPPEDWINDOW,
FALSE);
        switch (umsg) {
        case WM_CREATE:
            hChildWnd = (HWND)lParam;
            hChildWnd2 = (HWND)lParam;
            hInstance = ((LPCREATESTRUCT)lParam)->hInstance;
            break;

        case WM_COMMAND:
            switch (LOWORD(wParam)) {
            case ID_BTN_OPEN_WINDOW:

                hChildWnd = CreateWindowEx(0,
L"SecondWindowClass", L"Игра", WS_OVERLAPPEDWINDOW, windowX,
windowY, windowWidth * 2 , windowHeight * 2, hwnd, NULL,
hInstance, hwnd);

                if (!hChildWnd) {
                    MessageBox(NULL, _T("Не удалось создать
второе окно!"), _T("Ошибка"), MB_OK | MB_ICONERROR);
                }
                else {

                    ShowWindow(hChildWnd, SW_SHOW);
                    ShowWindow(hwnd, SW_HIDE);
                }
                break;

            case ID_BTN_OPEN_WINDOW2:
                hChildWnd2 = CreateWindowEx(0,
L"ThirdWindowClass", L"Настройки",WS_OVERLAPPEDWINDOW, windowX,
windowY,windowWidth+100, windowHeight+50, hwnd, NULL, hInstance,
hwnd);

                if (!hChildWnd2) {
                    MessageBox(NULL, _T("Не удалось создать
третье окно!"), _T("Ошибка"), MB_OK | MB_ICONERROR);
                }
                else {
                    CreateWindow(_T("BUTTON"), _T("В главное
меню"), WS_CHILD | WS_VISIBLE | BS_PUSHBUTTON, 150, 200, 250,
30, hChildWnd2, (HMENU)ID_BTN_EXIT, hInstance, NULL);

                    ShowWindow(hChildWnd2, SW_SHOW);
                    ShowWindow(hwnd, SW_HIDE);
                }
                break;

            case ID_BTN_ACTION2:
```

```
                PostQuitMessage(0);
                break;
            }
            break;
        case WM_CLOSE:
            ShowWindow(hwnd, SW_HIDE);
            PostQuitMessage(0);
            break;

        case WM_DESTROY:
            ShowWindow(hwnd, SW_HIDE);
            PostQuitMessage(0);
            break;
        default:
            return DefWindowProc(hwnd, umsg, wParam, lParam);
        }
        return 0;
    }

    LRESULT CALLBACK SecondWndProc(HWND hChildWnd, UINT umsg,
WPARAM wParam, LPARAM lParam) {
        static HWND hMainWnd, hButtonCheck = nullptr;
        static HINSTANCE hInstance;
        int NUM_ROWS = Size;
        int NUM_COLS = Size;

        double elapsedTime = 0;

        HWND** hEditControls1 = nullptr;
        AdjustWindowRect(&windowRect, WS_OVERLAPPEDWINDOW,
FALSE);
        int x = 10, y = 10;
        int x1 = 10, y1 = 10;
        int num = 1;
        int textX = 10 + NUM_COLS * CELL_WIDTH + 20;
        int textY = 10;
        int sqrtH = 0;
        TCHAR timeBuffer[100] = {0};
        LARGE_INTEGER startTime, endTime, freq;
        TCHAR bufferTimeCreate[100];
        double elapsedSeconds = 0;
        int sqrtSize = static_cast<int>(sqrt(Size));

        switch (umsg) {
        case WM_CREATE:
            g_elapsedSeconds = 0;
            elapsedSeconds = 0;
            hInstance = ((LPCREATESTRUCT)lParam)->hInstance;
            hMainWnd = (HWND)lParam;
            hMainWnd =
CreateWindowEx(0,_T("MainWindowClass"),_T("Главное
```

```
окно"),WS_OVERLAPPEDWINDOW,windowX, windowY,windowWidth,
windowHeight,NULL, NULL,hInstance,NULL);

            hEditControls = new HWND * [NUM_ROWS];
            for (int i = 0; i < NUM_ROWS; i++) {
                hEditControls[i] = new HWND[NUM_COLS];
                memset(hEditControls[i], 0, NUM_COLS *
sizeof(HWND));
            }


            QueryPerformanceCounter(&startTime);
            MemoryUsageBefore= getTotalMemoryUsage();
            otherData = mains( Size,numAlgor);
            MemoryUsageAfter= getTotalMemoryUsage();
            MemoryUsageTotal = MemoryUsageAfter -
MemoryUsageBefore;
            QueryPerformanceCounter(&endTime);
            QueryPerformanceFrequency(&freq);
            elapsedSeconds =
static_cast<double>(endTime.QuadPart - startTime.QuadPart) /
freq.QuadPart;
            _stprintf_s(bufferTimeCreate, _T("Время выполнения:
%.6f секунд.Использование памяти: %d байт"), elapsedSeconds,
MemoryUsageTotal);
            MessageBox(hMainWnd, bufferTimeCreate,
_T("Информация"), MB_OK);
            matrixData =deleteRandomCells(otherData,Size,Diff);

            NUM_ROWS = Size;
            NUM_COLS = Size;

            for (int row = 0; row < NUM_ROWS; row++) {
                for (int col = 0; col < NUM_COLS; col++) {
                    if (matrixData[row][col] == 0) {
                        hEditControls[row][col] =
CreateWindow(L"EDIT", NULL, WS_CHILD | WS_VISIBLE | WS_BORDER |
ES_NUMBER, x, y, CELL_WIDTH, CELL_HEIGHT, hChildWnd, NULL,
hInstance, NULL);
                    }
                    else {
                        hEditControls[row][col] =
CreateWindow(L"STATIC", NULL, WS_CHILD | WS_VISIBLE | SS_CENTER,
x, y, CELL_WIDTH, CELL_HEIGHT, hChildWnd, NULL, hInstance,
NULL);
                        TCHAR buffer[3];
                        _itow_s(matrixData[row][col], buffer, 3,
10);
                        SetWindowText(hEditControls[row][col],
buffer);
                    }
```

```
                    x += CELL_WIDTH;
                }
                x = 10;
                y += CELL_HEIGHT;
            }

            // Создание сетки
            x = 10;
            y = 10 + CELL_HEIGHT;
            for (int row = 0; row < NUM_ROWS; row++) {
                CreateWindow(L"STATIC", NULL, WS_CHILD |
WS_VISIBLE | SS_ETCHEDHORZ, x, y, NUM_COLS * CELL_WIDTH, 1,
hChildWnd, NULL, hInstance, NULL);
                y += CELL_HEIGHT;
            }

            x = 10 + CELL_WIDTH;
            y = 10;
            for (int col = 0; col < NUM_COLS; col++) {
                CreateWindow(L"STATIC", NULL, WS_CHILD |
WS_VISIBLE | SS_ETCHEDVERT, x, y, 1, NUM_ROWS * CELL_HEIGHT,
hChildWnd, NULL, hInstance, NULL);
                x += CELL_WIDTH;
            }

            x1 = 10;
            y1 = 10;
            for (int i = 0; i < NUM_ROWS; i += sqrtSize) {
                for (int j = 0; j < NUM_COLS; j += sqrtSize) {
                    CreateWindow(L"STATIC", NULL, WS_CHILD |
WS_VISIBLE | SS_ETCHEDFRAME,x1 + j * CELL_WIDTH, y1 + i *
CELL_HEIGHT,sqrtSize * CELL_WIDTH, sqrtSize *
CELL_HEIGHT,hChildWnd, NULL, hInstance, NULL);
                }
            }

            TCHAR diffStr[10], sizeStr[10];
            _itow_s(Diff, diffStr, 10, 10);
            _itow_s(Size, sizeStr, 10, 10);

            CreateWindow(L"STATIC",L"Правила игры:",WS_CHILD |
WS_VISIBLE,x+230, textY,200, 20,hChildWnd,NULL,hInstance,NULL);
            CreateWindow(L"STATIC",L"Сложность:", WS_CHILD |
WS_VISIBLE, x + 30, textY, 200, 20, hChildWnd, NULL, hInstance,
NULL);
            CreateWindow(L"STATIC",diffStr,WS_CHILD |
WS_VISIBLE,x + 130, textY,50, 20,hChildWnd,NULL,hInstance,NULL);

            textY += 30;
```

```
            CreateWindow(L"STATIC",L"Размеры поля:",WS_CHILD |
WS_VISIBLE,x+30, textY,200, 20,hChildWnd,NULL, hInstance,NULL);
            CreateWindow(L"STATIC", L"Числа не должны
повторяться в строке,столбце", WS_CHILD | WS_VISIBLE, x + 230,
textY, 450, 20, hChildWnd, NULL, hInstance, NULL);
            CreateWindow(L"STATIC", L",а так же в квадрате.",
WS_CHILD | WS_VISIBLE, x + 230, textY+30, 450, 20, hChildWnd,
NULL, hInstance, NULL);
            CreateWindow(L"STATIC", L"Числа могут быть от 1 до
значения размеров поля.", WS_CHILD | WS_VISIBLE, x + 230,textY +
60, 450, 20, hChildWnd, NULL, hInstance, NULL);
            CreateWindow(L"STATIC", L"Например:если размер поля
9x9,то в квадрате 3x3", WS_CHILD | WS_VISIBLE, x + 230,
textY+90, 450, 20, hChildWnd, NULL, hInstance, NULL);
            CreateWindow(L"STATIC", L" числа не должны
повторяться.", WS_CHILD | WS_VISIBLE, x + 230, textY + 120, 450,
20, hChildWnd, NULL, hInstance, NULL);
            CreateWindow(L"STATIC",sizeStr,WS_CHILD |
WS_VISIBLE,x + 160, textY,50, 20,hChildWnd,NULL,hInstance,NULL);

            hButtonCheck =
CreateWindow(L"BUTTON",L"Проверить",WS_CHILD | WS_VISIBLE, x+30,
textY+30, 100, 30,
hChildWnd,(HMENU)IDC_BUTTON_CHECK,hInstance,NULL);

            CreateWindow(_T("BUTTON"), _T("Играть"), WS_VISIBLE
| WS_CHILD, 100, 10, 250, 30, hMainWnd,
(HMENU)ID_BTN_OPEN_WINDOW, hInstance, NULL);
            CreateWindow(_T("BUTTON"), _T("Настройки игры"),
WS_VISIBLE | WS_CHILD, 100, 100, 250, 30, hMainWnd,
(HMENU)ID_BTN_OPEN_WINDOW2, hInstance, NULL);
            CreateWindow(_T("BUTTON"), _T("Выход"), WS_CHILD |
WS_VISIBLE | BS_PUSHBUTTON, 100, 200, 250, 30, hMainWnd,
(HMENU)ID_BTN_ACTION2, hInstance, NULL);

            hTimerControl = CreateWindow(L"STATIC", NULL,
WS_CHILD | WS_VISIBLE, x + 30, textY + 60, 100, 20, hChildWnd,
NULL, hInstance, NULL);

            bestTime = readBestTime(Diff, Size);
            bestDifficulty = Diff;
            bestSize = Size;

            TCHAR bestTimeStr[20];
            if (bestTime > 0.0) {
                _stprintf_s(bestTimeStr, 20, _T("%.2f сек"),
bestTime);
            }
            else {
                _tcscpy_s(bestTimeStr, 20, _T("Нет
результата"));
```

```
            }
            textY += 90;
            CreateWindow(L"STATIC", L"Лучшее время:", WS_CHILD |
WS_VISIBLE, x + 30, textY, 200, 20, hChildWnd, NULL, hInstance,
NULL);
            textY += 30;
            CreateWindow(L"STATIC", bestTimeStr, WS_CHILD |
WS_VISIBLE, x + 30, textY, 200, 20, hChildWnd, NULL, hInstance,
NULL);

            CreateWindow(_T("BUTTON"), _T("В главное меню"),
WS_CHILD | WS_VISIBLE | BS_PUSHBUTTON, x + 130, textY + 30, 200,
30, hChildWnd, (HMENU)ID_BTN_EXIT, hInstance, NULL);

            g_timerID = SetTimer(hChildWnd, 1, 1000, nullptr);

            break;
        case WM_TIMER:
            switch (wParam) {
            case 1:
            {
                g_elapsedSeconds++;
                wsprintf(timeBuffer, _T("%02d:%02d:%02d"),
g_elapsedSeconds / 3600, (g_elapsedSeconds % 3600) / 60,
g_elapsedSeconds % 60);
                SetWindowText(hTimerControl, timeBuffer);
            }
            break;
            }
            break;
        case WM_CLOSE:
            KillTimer(hChildWnd, g_timerID);
            delete[] hEditControls;
            hEditControls = nullptr;
            ShowWindow(hMainWnd, SW_SHOW);
            UpdateWindow(hMainWnd);
            DestroyWindow(hChildWnd);
            break;
        case WM_COMMAND:
            switch (LOWORD(wParam)) {
            case ID_BTN_EXIT:
                KillTimer(hChildWnd, g_timerID);
                SendMessage(hChildWnd, WM_CLOSE, 0, 0);
                DestroyWindow(hChildWnd);
                break;
            case IDC_BUTTON_CHECK: {
                bool isValid = true;
                int row = 0;
                int col = 0;
                int i=0, j=0;
                for ( row = 0; row < Size; row++) {
```

```
                    for (col = 0; col < Size; col++) {
                        TCHAR buffer[10];
                        GetWindowText(hEditControls[row][col],
buffer, 10);
                        int value = _wtoi(buffer);
                        if (value != int(otherData[row][col])) {
                            isValid = false;
                            break;
                        }
                    }
                    if (!isValid) break;
                }
                if (isValid) {
                    KillTimer(hChildWnd, g_timerID);
                    elapsedTime = g_elapsedSeconds;
                    if (elapsedTime < bestTime || bestTime ==
0.0) {
                        writeBestTime(Diff, Size, elapsedTime);
                        bestTime = elapsedTime;
                    }
                    TCHAR message[200];

                    wsprintf(message, _T("Поле заполнено
правильно, молодец! Ваше время: %02d:%02d:%02d"),
                        g_elapsedSeconds / 3600,
(g_elapsedSeconds % 3600) / 60, g_elapsedSeconds % 60);
                    MessageBox(hChildWnd, message, L"Проверка",
MB_OK | MB_ICONINFORMATION);
                    EnableWindow(hButtonCheck, FALSE);
                }
                else {
                    MessageBox(hMainWnd, L"Поле содержит
ошибки:(", L"Проверка", MB_OK);
                }
                break;
                }
            }
            break;
        case WM_DESTROY:
            KillTimer(hChildWnd, g_timerID);
            delete[] hEditControls;
            hEditControls = nullptr;
            DestroyWindow(hChildWnd);
            break;
        default:
            return DefWindowProc(hChildWnd, umsg, wParam,
lParam);
        }
        return 0;
    }
```

```
    LRESULT CALLBACK ThirdWndProc(HWND hChildWnd2, UINT umsg,
WPARAM wParam, LPARAM lParam) {
        static HWND hMainWnd, hwndDiffEdit,
hwndSizeEdit,hwndAlgorithmCombo;
        static HINSTANCE hInstance;

        switch (umsg) {
        case WM_CREATE:
            hInstance = ((LPCREATESTRUCT)lParam)->hInstance;
            hMainWnd = (HWND)lParam;
            hMainWnd =
CreateWindowEx(0,_T("MainWindowClass"),_T("Главное
окно"),WS_OVERLAPPEDWINDOW,windowX, windowY,windowWidth,
windowHeight,NULL, NULL,hInstance,NULL);

            CreateWindow(_T("STATIC"), _T("Сложность(от 1 до
5):"), WS_CHILD | WS_VISIBLE, 10, 50, 250, 20, hChildWnd2, NULL,
NULL, NULL);
            hwndDiffEdit = CreateWindow(_T("EDIT"), NULL,
WS_CHILD | WS_VISIBLE | WS_BORDER, 255, 50, 100, 20, hChildWnd2,
NULL, NULL, NULL);

            CreateWindow(_T("STATIC"), _T("Размеры поля( 4, 9,
16, 25):"), WS_CHILD | WS_VISIBLE, 10, 80, 250, 20, hChildWnd2,
NULL, NULL, NULL);
            hwndSizeEdit = CreateWindow(_T("EDIT"), NULL,
WS_CHILD | WS_VISIBLE | WS_BORDER, 255, 80, 100, 20, hChildWnd2,
NULL, NULL, NULL);

            CreateWindow(_T("STATIC"), _T("Выберите алгоритм:"),
WS_CHILD | WS_VISIBLE, 10, 110, 250, 20, hChildWnd2, NULL, NULL,
NULL);
            hwndAlgorithmCombo = CreateWindow(_T("COMBOBOX"),
NULL, WS_CHILD | WS_VISIBLE | CBS_DROPDOWNLIST, 255, 110, 250,
200, hChildWnd2, NULL, NULL, NULL);

            SendMessage(hwndAlgorithmCombo, CB_ADDSTRING, 0,
(LPARAM)_T("1 - Back traking"));
            SendMessage(hwndAlgorithmCombo, CB_ADDSTRING, 0,
(LPARAM)_T("2 - Dancing links"));
            SendMessage(hwndAlgorithmCombo, CB_ADDSTRING, 0,
(LPARAM)_T("3 - Genetic algorithm"));

            CreateWindow(_T("BUTTON"), _T("Сохранить"), WS_CHILD
| WS_VISIBLE | BS_PUSHBUTTON, 100, 140, 100, 30, hChildWnd2,
(HMENU)ID_BTN_SAVE, hInstance, NULL);

            CreateWindow(_T("BUTTON"), _T("Играть"), WS_VISIBLE
| WS_CHILD, 100, 10, 250, 30, hMainWnd,
(HMENU)ID_BTN_OPEN_WINDOW, hInstance, NULL);
```

```
            CreateWindow(_T("BUTTON"), _T("Настройки игры"),
WS_VISIBLE | WS_CHILD, 100, 100, 250, 30, hMainWnd,
(HMENU)ID_BTN_OPEN_WINDOW2, hInstance, NULL);
            CreateWindow(_T("BUTTON"), _T("Выход"), WS_CHILD |
WS_VISIBLE | BS_PUSHBUTTON, 100, 200, 250, 30, hMainWnd,
(HMENU)ID_BTN_ACTION2, hInstance, NULL);

            break;

        case WM_CLOSE:
            ShowWindow(hMainWnd, SW_SHOW);
            UpdateWindow(hMainWnd);
            DestroyWindow(hChildWnd2);
            break;
        case WM_COMMAND:
            switch (LOWORD(wParam)) {
            case ID_BTN_EXIT:
                SendMessage(hChildWnd2, WM_CLOSE, 0, 0);
                break;
            case ID_BTN_SAVE: {

                TCHAR diffStr[10], sizeStr[10];
                GetWindowText(hwndDiffEdit, diffStr, 10);
                GetWindowText(hwndSizeEdit, sizeStr, 10);
                Diff = _tstoi(diffStr);
                Size = _tstoi(sizeStr);
                numAlgor= SendMessage(hwndAlgorithmCombo,
CB_GETCURSEL, 0, 0);


                TCHAR message[100];
                if (Diff > 0 && Size > 0) {
                    _stprintf_s(message, _T("Значения сохранены:
Сложность = %d, Размеры поля = %d"), Diff, Size);
                }
                else {
                    _stprintf_s(message, _T("Некорректные
значения. Введите положительные числа."));
                }
                MessageBox(hChildWnd2, message, _T("Сохранение
настроек"), MB_OK | MB_ICONINFORMATION);
                break;
                }
            }
            break;
        case WM_DESTROY:
            ShowWindow(hMainWnd, SW_SHOW);
            UpdateWindow(hMainWnd);
            DestroyWindow(hChildWnd2);
            break;
        default:
```

```
            return DefWindowProc(hChildWnd2, umsg, wParam,
lParam);
        }
        return 0;

    }

    int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE
hPrevInstance, LPSTR lpCmdLine, int nCmdShow) {
        WNDCLASSEX wc;
        HWND hwnd;
        MSG msg;
        AdjustWindowRect(&windowRect, WS_OVERLAPPEDWINDOW,
FALSE);

        // Регистрация класса окна для главного окна
        wc.cbSize = sizeof(WNDCLASSEX);
        wc.style = CS_HREDRAW | CS_VREDRAW;
        wc.lpfnWndProc = MainWndProc;
        wc.cbClsExtra = 0;
        wc.cbWndExtra = 0;
        wc.hInstance = hInstance;
        wc.hIcon = LoadIcon(NULL, IDI_APPLICATION);
        wc.hCursor = LoadCursor(NULL, IDC_ARROW);
        wc.hbrBackground = (HBRUSH)(COLOR_WINDOW + 1);
        wc.lpszMenuName = NULL;
        wc.lpszClassName = _T("MainWindowClass");
        wc.hIconSm = LoadIcon(NULL, IDI_APPLICATION);

        if (!RegisterClassEx(&wc)) {
            MessageBox(NULL, _T("Не удалось зарегистрировать
класс главного окна!"), _T("Ошибка"), MB_OK | MB_ICONERROR);
            return 0;
        }

        // Регистрация класса окна для второго окна
        WNDCLASSEX wcChild;
        wcChild.cbSize = sizeof(WNDCLASSEX);
        wcChild.style = CS_HREDRAW | CS_VREDRAW;
        wcChild.lpfnWndProc = SecondWndProc;
        wcChild.cbClsExtra = 0;
        wcChild.cbWndExtra = 0;
        wcChild.hInstance = hInstance;
        wcChild.hIcon = LoadIcon(NULL, IDI_APPLICATION);
        wcChild.hCursor = LoadCursor(NULL, IDC_ARROW);
        wcChild.hbrBackground = (HBRUSH)(COLOR_WINDOW + 1);
        wcChild.lpszMenuName = NULL;
        wcChild.lpszClassName = _T("SecondWindowClass");
        wcChild.hIconSm = LoadIcon(NULL, IDI_APPLICATION);

        if (!RegisterClassEx(&wcChild)) {
```

```
            MessageBox(NULL, _T("Не удалось зарегистрировать
класс второго окна!"), _T("Ошибка"), MB_OK | MB_ICONERROR);
            return 0;
        }

        // Регистрация класса окна для третьего окна
        WNDCLASSEX wcChild2;
        wcChild2.cbSize = sizeof(WNDCLASSEX);
        wcChild2.style = CS_HREDRAW | CS_VREDRAW;
        wcChild2.lpfnWndProc = ThirdWndProc;
        wcChild2.cbClsExtra = 0;
        wcChild2.cbWndExtra = 0;
        wcChild2.hInstance = hInstance;
        wcChild2.hIcon = LoadIcon(NULL, IDI_APPLICATION);
        wcChild2.hCursor = LoadCursor(NULL, IDC_ARROW);
        wcChild2.hbrBackground = (HBRUSH)(COLOR_WINDOW + 1);
        wcChild2.lpszMenuName = NULL;
        wcChild2.lpszClassName = _T("ThirdWindowClass");
        wcChild2.hIconSm = LoadIcon(NULL, IDI_APPLICATION);

        if (!RegisterClassEx(&wcChild2)) {
            MessageBox(NULL, _T("Не удалось зарегистрировать
класс второго окна!"), _T("Ошибка"), MB_OK | MB_ICONERROR);
            return 0;
        }

        // Создание главного окна
        hwnd =
CreateWindowEx(0,_T("MainWindowClass"),_T("Главное окно"),
            WS_OVERLAPPEDWINDOW,windowX, windowY,windowWidth,
windowHeight,NULL, NULL,hInstance,NULL);
        if (!hwnd) {
            MessageBox(NULL, _T("Не удалось создать главное
окно!"), _T("Ошибка"), MB_OK | MB_ICONERROR);
            return 0;
        }

        HWND hChildWnd = NULL, hChildWnd2 = NULL;

        CreateWindow(_T("BUTTON"), _T("Играть"), WS_VISIBLE |
WS_CHILD, 100, 10, 250, 30, hwnd, (HMENU)ID_BTN_OPEN_WINDOW,
hInstance, NULL);
        CreateWindow(_T("BUTTON"), _T("Настройки игры"),
WS_VISIBLE | WS_CHILD, 100, 100, 250, 30, hwnd,
(HMENU)ID_BTN_OPEN_WINDOW2, hInstance, NULL);
        CreateWindow(_T("BUTTON"), _T("Выход"), WS_CHILD |
WS_VISIBLE | BS_PUSHBUTTON, 100, 200, 250, 30, hwnd,
(HMENU)ID_BTN_ACTION2, hInstance, NULL);

        ShowWindow(hwnd, nCmdShow);
        UpdateWindow(hwnd);
```

```
    while (GetMessage(&msg, NULL, 0, 0)) {
        if (!IsDialogMessage(hChildWnd, &msg)) {
            TranslateMessage(&msg);
            DispatchMessage(&msg);
        }
    }
    return (int)msg.wParam;
}
```

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное автономное образовательное учреждение высшего образования
«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

ОТЗЫВ РУКОВОДИТЕЛЯ НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ

на тему Игровое приложение «Судоку»

выполненную студентом группы № 1044

Чумиловым Ильей Андреевичем

фамилия, имя, отчество студента

по направлению подготовки/специальности 09.03.01 (код) Информатика и вычислительная техника (наименование направления подготовки/специальности)

(наименование направления подготовки/специальности)

Актуальность темы работы:

Актуальность темы работы заключается в создании игрового приложения, с помощью которого будет проведен анализ работы алгоритмов по показателям скорости работы и затрат памяти.

Цель и задачи работы:

Целью работы является изучение правил игры «Судоку», изучение расположение чисел на игровом поле, и какого оно может быть размера. Также выбор методов и реализация алгоритмов заполнения игрового поля, разработка игрового приложения и на его основе анализ алгоритмов генерации игрового поля, скорости работы и затрат памяти.

Общая оценка выполнения поставленной перед студентом задачи, основные достоинства и недостатки работы:

Поставленные перед И.А. Чумиловым задачи были выполнены. Были выбраны и реализованы алгоритмы для генерации игрового поля, а также другие алгоритмы работы всего приложения. Был выполнен анализ алгоритмов для генерации игрового поля на основе скорости выполнения и использования памяти.

Степень самостоятельности и способности к исследовательской работе студента (умение и навыки поиска, обобщения, анализа материала и формулирования выводов):

В ходе выполнения поставленных задач и написания квалификационной работы И.А. Чумилов проявил самостоятельность, умение ставить перед собой задачи, выполнять поиск актуальных технологий, изучение и анализ материалов.

Проверка текста выпускной квалификационной работы с использованием системы «Антиплагиат.ВУЗ», проводившаяся 6 июня 2024г. в 13:28, показывает оригинальность содержания на уровне 85,83%.

Степень грамотности изложения и оформления материала:

Материал пояснительной записки изложен последовательно. В работе присутствуют все необходимые поясняющие рисунки и примеры программного кода.

Оценка деятельности студента в период подготовки выпускной квалификационной работы (добросовестность, работоспособность, ответственность, аккуратность и т.п.):

За время работы над выпускной квалификационной работой И.А. Чумилов зарекомендовал себя как отличный специалист, организованный и ответственный работник, способный самостоятельно и добросовестно решать поставленные задачи.

Общий вывод:

Считаю, что И.А. Чумилов данной работой подтвердил свою квалификацию, и заслуживает присвоения квалификации бакалавра по направлению «09.03.01 Информатика и вычислительная техника».

В работе не содержится информация с ограниченным доступом и отсутствуют сведения, представляющие коммерческую ценность.

Руководитель

доц., к.т.н	10.06.24	А.В. Шахомиров
должность, уч. степень, звание	подпись, дата	инициалы, фамилия