

Министерство образования Республики Беларусь
Учреждение образования «Белорусский государственный университет
информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных машин

Дисциплина: Программирование на языках высокого уровня

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовому проекту
на тему

ИГРА “Судоку”

БГУИР КП 1-40 02 01 214 ПЗ

Студент: гр. 250502 Кисель Д. Д.

Руководитель: Богдан Е. В.

МИНСК 2023

Учреждение образования
«Белорусский государственный университет информатики
и радиоэлектроники»

Факультет компьютерных систем и сетей

УТВЕРЖДАЮ
Заведующий кафедрой

(подпись)

2023 г.

ЗАДАНИЕ
по курсовому проектированию

Студенту Киселю Даниил Дмитриевичу

Тема проекта Игра “Судоку”

2. Срок сдачи студентом законченного проекта 15 декабря 2023 г.

3. Исходные данные к проекту Игра “Судоку”

4. Содержание расчетно-пояснительной записки (перечень вопросов, которые подлежат разработке)

1. Лист задания.

2. Введение.

3. Обзор литературы.

3.1. Обзор существующих аналогов.

3.2. Постановка задачи.

3.3. Обзор методов и алгоритмов решения поставленной задачи.

4. Функциональное проектирование.

4.1. Структура входных и выходных данных.

4.2. Разработка диаграммы классов.

4.3. Описание классов.

5. Разработка программных модулей.

5.1. Разработка схем алгоритмов (два наиболее важных метода)

5.2. Разработка алгоритмов (описание алгоритмов по шагам, для двух методов)

6. Результаты работы

7. Заключение

8. Литература

9. Приложения

5. Перечень графического материала (с точным обозначением обязательных чертежей и графиков)

1. Диаграмма классов.

2. Схема алгоритма annealing()

3. Схема алгоритма solve()

6. Консультант по проекту (с обозначением разделов проекта) Богдан Е. В.

7. Дата выдачи задания 15.09.2023г

8. Календарный график работы над проектом на весь период проектирования (с обозначением сроков выполнения и трудоемкости отдельных этапов):

1. Выбор задания. Разработка содержания пояснительной записки. Перечень графического материала 15 %;

разделы 2, 3 – 10 %;

разделы 4 к – 20 %;

разделы 5 к – 35 %;

раздел 6,7,8 – 5 %;

раздел 9 к – 5%;

оформление пояснительной записки и графического материала к 15.12.23 10 %

Защита курсового проекта с 21.12 по 28.12.23г.

РУКОВОДИТЕЛЬ Богдан Е. В.

(подпись)

Задание принял к исполнению _____

(дата и подпись студента)

СОДЕРЖАНИЕ

ВВЕДЕНИЕ.....	5
1 ОБЗОР ЛИТЕРАТУРЫ.....	8
1.1 Анализ существующих аналогов.....	8
1.1.1 Игра “Sudoku.com”.....	8
1.1.2 Игра “Sudoku Universe”	9
1.1.3 Игра “Sudoku Jigsaw”.....	10
1.1.4 Игра “Sudoku Killer”	10
1.2 Постановка задачи	11
1.3 Обзор методов и алгоритмов решения поставленной задачи	11
2 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ	15
2.1 Структура входных и выходных данных	15
2.2 Разработка диаграммы классов	16
2.3 Описание классов.....	16
2.3.1 Класс Field	16
2.3.2 Класс Cell	16
2.3.3 Класс Generator.....	17
2.3.4 Класс Solver	17
2.3.5 Класс Gamewidget	17
3 Разработка программных модулей.....	21
3.1 Разработка схем алгоритмов.....	21
3.2 Разработка алгоритмов	21
3.2.1 Разработка алгоритма функции annealing()	21
3.2.2 Разработка алгоритма метода solve().....	21
4 Результат работы	23
ЗАКЛЮЧЕНИЕ	26
СПИСОК ЛИТЕРАТУРЫ.....	27
ПРИЛОЖЕНИЕ А	28
ПРИЛОЖЕНИЕ Б.....	29
ПРИЛОЖЕНИЕ В	30
ПРИЛОЖЕНИЕ Г	31
ПРИЛОЖЕНИЕ Д	51

ВВЕДЕНИЕ

История Судoku проистекает от игры швейцарского математика 18-го века под названием “Латинские квадраты”, а некоторые из первых пазлов, появившихся в газетах, были опубликованы во Франции в 1895 году. Но современная игра Судoku, какой мы ее видим сегодня, была создана Говардом Гарнсом, независимым разработчиком головоломок из Коннерсвилла, штат Индиана, США, в 1979 году, когда она была опубликована в журналах “Dell Pencil Puzzles” и “Word Games”. Эта головоломка была известна под названием “Number Place”, поскольку в ней требовалось размещать недостающие числа в пустые места на сетке, состоящей из 9 x 9 клеток.

Игра впервые появилась в Японии в 1984 году, где ей было дано имя “Судoku”. Это выражение означает – “числа ограничены одним местонахождением”. Судoku и сейчас пользуется большой популярностью в Японии, где каждый месяц продается более 600 000 журналов.

Сложность Судoku зависит от количества изначально заполненных клеток и от методов, которые нужно применять для её решения. Самые простые решаются дедуктивно: всегда есть хотя бы одна клетка, куда подходит только одно число. Некоторые головоломки можно решить за несколько минут, на другие можно потратить несколько часов.

Правильно составленная головоломка имеет только одно решение. Тем не менее, на некоторых сайтах в интернете под видом усложнённых головоломок пользователю предлагаются варианты Судoku с несколькими вариантами решения, а также с ветвлениями самого хода решения.

Существует множество вариаций Судoku:

Нерегулярное Судoku (их ещё называют Судoku-пазл, Судoku-фигуры, Области). В этой головоломке вместо стандартных областей 3×3 используются области произвольной формы. Цифры не должны повторяться в каждой такой области.

Судoku разных размеров. Встречаются Судoku размером от 4×4 (такие уменьшенные головоломки называют “Шидoku” от японского shi– четыре) до 25×25. Наиболее распространены Судoku небольших размеров для детей.

Судoku с дополнительными областями. В этих задачах помимо стандартных областей (вертикалей, горизонталей и блоков) задаются дополнительные области, в которых цифры не могут повторяться. Наиболее распространены диагональные Судoku.

Судoku с дополнительными условиями. В таких задачах помимо стандартного условия неповторяемости цифр задаются ещё дополнительные условия на значения. Например:

Судoku “Суммы” и множество её вариаций (“Судoku-Произведения”, “Судoku-Арифметика”, “Квадросудoku”, “Суммы-сбоку”, “Суммы по диагонали” и тому подобное.), где всё поле делится на блоки, для которых сообщается сумма (произведение, разность и тому подобное) всех входящих в него цифр.

Судoku “Чёт-нечёт”. В них некоторые клетки изначально выделены цветом, в этих клетках находятся или только чётные, или только нечётные цифры.

Судoku “Больше-меньше”. Для некоторых соседних клеток указан знак, показывающий в какой из клеток цифра больше.

Судoku “Точки”, где между соседними клетками выводится точка белого или чёрного цвета, когда соседние цифры отличаются на 1 или в два раза.

Судoku “Перегородки” (“Судoku-Соседи”), где отмечены все места с цифрами в соседних клетках, отличающихся на 1.

Судoku, где вместо чисел используются японские символы или английские буквы.

“Шахматные” Судoku, где две или более клеток, которые связаны между собой ходом определённой шахматной фигуры (чаще всего коня), не могут содержать одинаковые цифры. Данное ограничение может налагаться на все возможные цифры или на некоторые из них.

Судoku другой формы. В головоломке используется не квадратная сетка, а треугольная, шестиугольная или более хитроумной формы. Сцепленные Судoku. “Судoku-Самурай” (“Судoku-Комбо”, “Судoku-Конструкции”, “Тудoku” и др.), в которых несколько отдельных Судoku имеют общие блоки и решаются взаимозависимо.

В данной курсовой работе мне предстоит реализовать классическое Судoku с использованием C++ и фреймворка Qt.

C++ является мощным и эффективным языком программирования с широкими возможностями. Его особенности, такие как высокая производительность, а также возможность работы с объектно-ориентированным и процедурным программированием, делают его идеальным выбором для разработки игр. C++ обеспечивает высокую скорость выполнения программы, что важно для быстрой обработки игровой логики.

Qt – это мощный фреймворк с открытым исходным кодом, который предоставляет разработчикам широкие возможности для создания кроссплатформенных графических приложений. Он включает в себя богатую библиотеку инструментов для работы с GUI (графическим интерфейсом пользователя), обработки событий, создания окон, кнопок, таблиц и других элементов интерфейса.

Qt также предоставляет удобные инструменты для создания пользовательского интерфейса игры. С его помощью можно легко создавать графические элементы, обрабатывать события взаимодействия пользователя с интерфейсом, анимировать элементы и отображать информацию в удобном для игроков виде.

Сочетание C++ и Qt предоставляет разработчикам мощный инструментарий для создания игры с высокой производительностью, эффективностью использования ресурсов компьютера и удобным пользовательским интерфейсом. Этот стек технологий позволяет легко

реализовать игровую логику, управление данными и создание приятного и интуитивно понятного интерфейса для игроков.

1 ОБЗОР ЛИТЕРАТУРЫ

1.1 Анализ существующих аналогов

Судоку является одной из самых известных головоломок.

Существует множество аналогов начиная от простых браузерных вариантов игры и заканчивая необычными вариантами на ПК с различными режимами.

1.1.1 Игра “Sudoku.com”

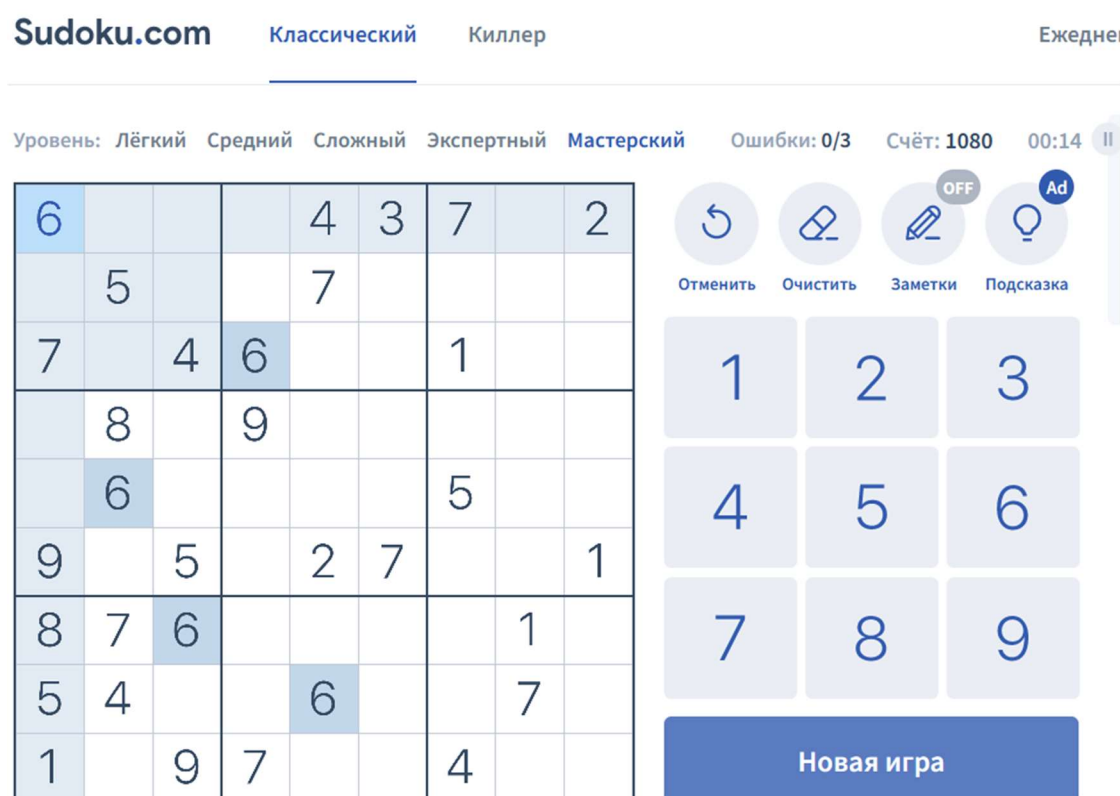


Рисунок 1.1.1 – Кадр из игры “Sudoku.com”

Данная игра представляет собой браузерный вариант игры Судоку. Здесь представлено несколько режимов игры. Можно поиграть в классическое Судоку с разными уровнями сложности, начиная от легкого уровня для новичков и заканчивая мастерским уровнем для настоящих профессионалов. Здесь также присутствует режим “Киллер”, который отличается от классического Судоку тем, что в качестве дополнительной подсказки здесь присутствуют области различной формы, в которых указана сумма всех чисел, находящихся внутри.

Уровень: Лёгкий Средний Сложный **Экспертный**

Ошибки: 0/3

Счёт: 0

00:03 II

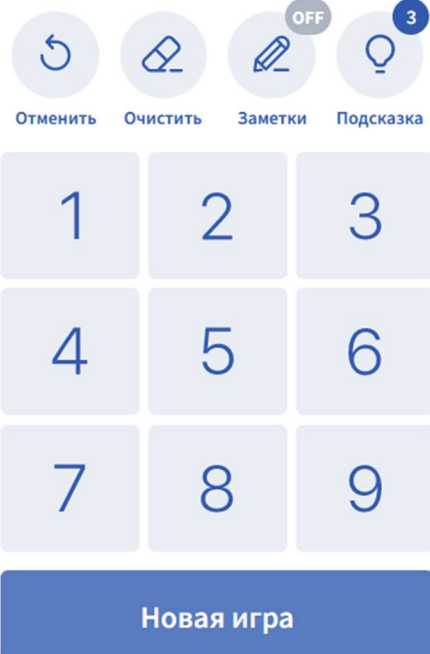
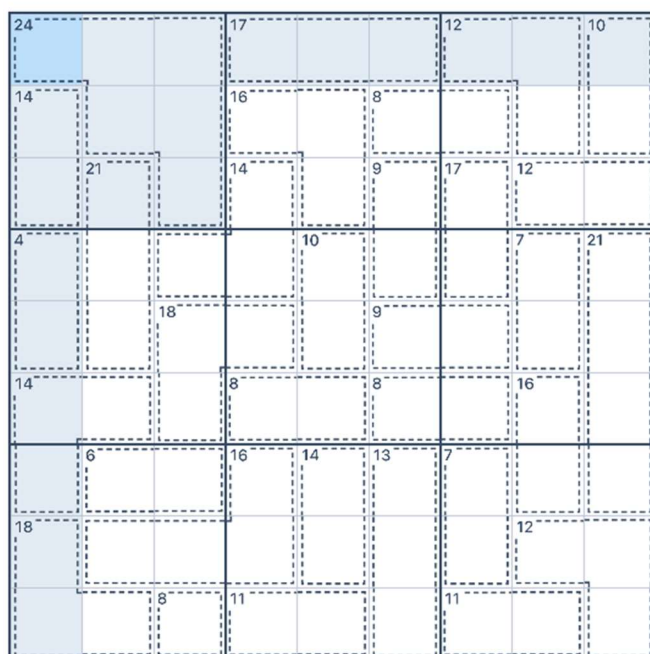


Рисунок 1.1.2 – Кадр из игры “Sudoku.com” в режиме “Киллер”

На самом сложном уровне сложности “Экспертный” здесь вовсе могут отсутствовать начальные подсказки в виде чисел.

1.1.2 Игра “Sudoku Universe”

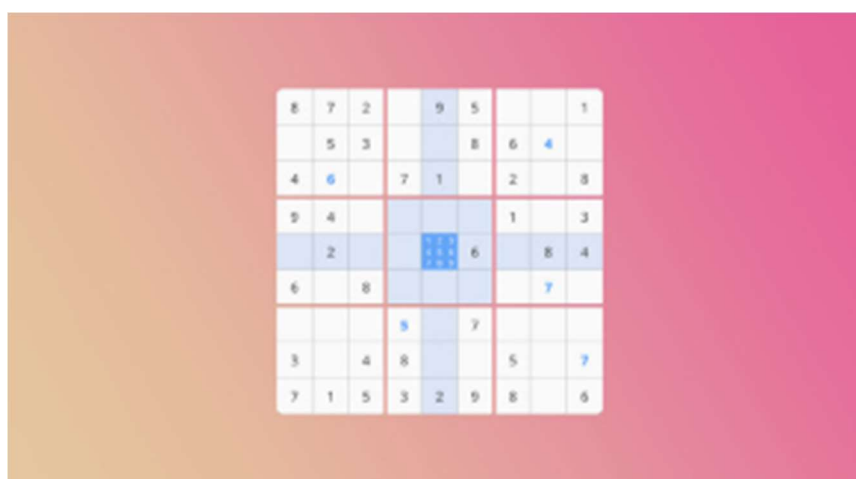


Рисунок 1.1.3 – Кадр из игры “Sudoku Universe”

Данная игра представляет собой реализацию классического Судоку на персональный компьютер. К выбору доступны несколько уровней сложности. Игра имеет приятный вид, функциональный визуальный интерфейс и большое количество уровней различной сложности.

1.1.3 Игра “Sudoku Jigsaw”

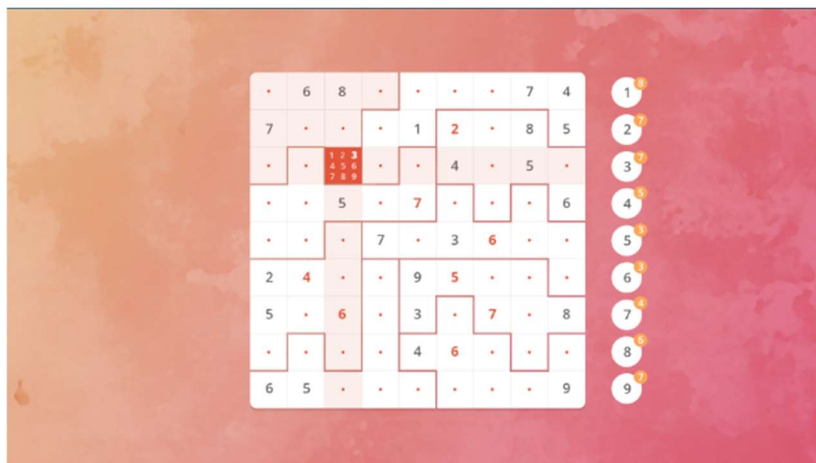


Рисунок 1.1.4 – Кадр из игры “Sudoku Jigsaw”

Правила игры “Jigsaw Sudoku” аналогичны стандартному Судоку, поскольку вы должны поместить каждое из чисел от 1 до 9 в каждую из строк и столбцов. Однако, в то время как в стандартном Судоку есть квадратные ячейки, каждая из которых должна содержать каждое число, в “Jigsaw Sudoku” эти ячейки заменены различными фигурами разной формы, каждая из которых должна содержать каждое число.

1.1.4 Игра “Sudoku Killer”

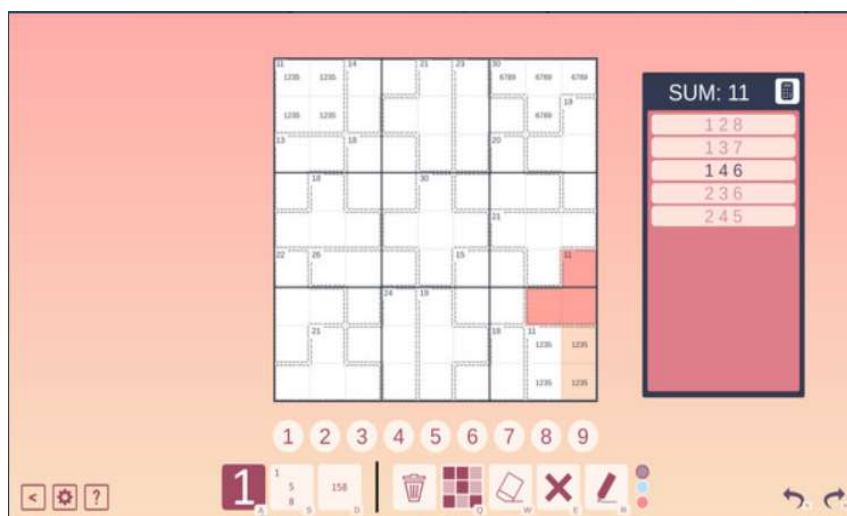


Рисунок 1.1.5 – Кадр из игры “Sudoku Killer”

Данная игра представляет собой реализацию классического Судоку с небольшой особенностью. Если в классическом Судоку у нас изначально известны некоторые клетки, то в данной реализации у нас изначально все поле пустое. Единственной подсказкой являются контуры, которые покрывают все поле. В данных контурах показывается сумма всех чисел, находящихся внутри. Данная игра будет хорошим аналогом классического Судоку для людей, которым надоело обычное Судоку.

1.2 Постановка задачи

“Судоку” – логическая игра, представляющая собой игровое поле, размером 9 на 9 клеток, некоторые клетки которого заполнены числами от 1 до 9; поле разделено на 9 секторов, представляющих собой квадраты, размером 3 на 3 клетки. Цель игрока: заполнить игровое поле, не нарушив правила игры.

Игровое поле представляет собой квадрат размером 9×9 , разделённый на меньшие квадраты со стороной в 3 клетки. Таким образом, всё игровое поле состоит из 81 клетки. В них уже в начале игры стоят некоторые числа (от 1 до 9), называемые подсказками. От игрока требуется заполнить свободные клетки цифрами от 1 до 9 так, чтобы в каждой строке, в каждом столбце и в каждом малом квадрате 3×3 каждая цифра встречалась бы только один раз.

Программа должна иметь удобный визуальный интерфейс. В программе должна быть предусмотрена возможность загрузить заранее сгенерированный уровень выбранной сложности, получить подсказку в решении уровня.

1.3 Обзор методов и алгоритмов решения поставленной задачи

Лучший метод решения – записывать числа-кандидаты в вершине левого угла ячейки, а затем вычёркивать невозможные по правилам игры числа из данной ячейки. После этого можно увидеть именно те числа, которые могут занимать данную ячейку. Играть в Судоку рекомендуется медленно, так как это расслабляющая игра.

Сначала смотрят на ряды, столбцы и блоки 3×3 с наиболее заполненными квадратами: легче решить там, где вариантов меньше. При заполнении ячейки нужно проверить столбец, ряд и блок 3×3 . Нужно проверить, что все другие 8 чисел не дублируются.

Когда в Судоку осталось несколько открытых ячеек в блоке 3×3 и только одна ячейка подходит для данного числа, то именно это число нужно записать в данную ячейку. Перед заполнением следует удостовериться, что вписываемое в ячейку число не будет встречаться в другой ячейке в том же столбце, строке или в блоке 3×3 .

Когда в одном столбце, строке или блоке 3×3 три любых ячейки имеют числа-кандидаты $\{1,2; 1,2; 1,3\}$, то число для третьей ячейки должно быть 3. Потому что если бы это было число 1, то в одной из первых двух ячеек было

бы число 2, а в другой не было бы ничего, но такого быть не может, поскольку все клетки должны быть заполнены.

Для хранения поля игры используется двумерный массив. Процесс генерации поля состоит из двух этапов: заполнение поля числами от 1 до 9 и удаление чисел из поля.

В начале генерации поля, двумерный массив заполняется случайными числами таким образом, чтобы в каждой строке не было повторяющихся чисел. Для приведения поля к состоянию, в котором нет повторяющихся чисел в каждом столбце и секторе, был разработан алгоритм, основанный на методе имитации отжига.

Метод имитации отжига работает следующий образом: пусть имеется некоторая функция $f(x)$ от состояния x , которую мы хотим минимизировать или максимизировать. Возьмём в качестве базового решения какое-то состояние x_0 и будем пытаться его улучшать.

Введём температуру t – какое-то действительное число, которое будет изменяться в течении оптимизации и влиять на вероятность перейти в соседнее состояние.

Пока не придём к оптимальному решению, будем повторять следующие шаги:

- уменьшим температуру;
- выберем случайного соседа x – то есть какое-то состояние y , которое может быть получено из x каким-то минимальным изменением;
- с вероятностью $p(f(x), f(y))$ сделаем присвоение $x = y$.

Стоит отметить, что в каждом шаге есть свобода реализации. Основные эвристические соображения следующие:

- в начале оптимизации наше решение плохое, поэтому мы можем позволить себе риск перейти в состояние хуже;
- в конце наоборот, наше решение почти оптимальное и мы не хотим терять прогресс, следовательно не можем позволить себе риск перейти в состояние хуже;
- температура должна быть высокой в начале и медленно уменьшаться к концу;
- алгоритм будет работать лучше, если функция $f(x)$ будет изменяться плавно.

Опишем более подробно детали, которые были изменены специально для текущей задачи: в качестве точки x подразумевается текущее расположение чисел в строке. Значение функции $f(x)$ в точке x равняется числу повторяющихся чисел в каждом столбце и секторе.

Переход от точки x к точке y происходит путем перестановки двух чисел в строке.

Если $f(y) < f(x)$, то выполняется присвоение $x = y$. Иначе, присвоение не выполняется.

Второй этап, удаление чисел, основан на поиске с возвратом.

Имеем какое-то число n – количество клеток, которые мы должны удалить. Выбирается и освобождается случайная клетка в игровом поле. Далее мы пробуем подставить в клетку все числа от 1 до 9.

Если подставленное число удовлетворяет правилам игры, то мы делаем данную клетку пустой и заново повторяем данный ход действий пока не освободим n клеток, но уже с полем, в котором отсутствует клетка, освобожденная на предыдущем этапе алгоритма.

Если на каком-то этапе алгоритма нам не удастся подставить число в пустую клетку так, чтобы она удовлетворяла правилам игры, то мы возвращаемся к клетке, которая была освобождена до этой, и пытаемся подставить другое число, которое также удовлетворяло бы правилам игры, и повторяем данный алгоритм, пока не освободим n клеток.

Моя реализация не поддерживает генерацию уровней в реальном времени из-за особенности моего алгоритма реализации. Для генерации требуются две функции: функция, генерирующая игровое поле, и функция, которая способна удалить выбранное число клеток.

В случае с функцией, генерирующей игровое поле, проблем не возникает так как функция работает по алгоритму, имитирующему отжиг. Функция оптимизирована и работает быстро.

В случае с функцией, которая удаляет выбранное число клеток, все сложнее. Данная функция является рекурсивной, что представляет собой большую проблему. Сами по себе рекурсивные функции довольно длинные и их применение не желательно. И моя функция не исключение. Для числа удаленных клеток около 30 функция работает пару секунд, что не так долго. Но при увеличении числа удаляемых клеток выше 40 начинаются проблемы со временем из-за того, что функция может попасть в “плохое” состояние.

“Плохим” состоянием является состояние, из-за которого моя функция может уйти очень глубоко в рекурсию. За каждый вызов рекурсии мое финальное время увеличивается экспоненциально. Поэтому для данной курсовой работы я не стал реализовывать генерацию в реальном времени.

При возникновении трудностей с прохождением уровня игрок может воспользоваться кнопкой “Подсказка”, которая открывает одну случайную клетку, или кнопкой “Решение” для полного решения всего уровня.

Для генерации 10 уровней сложности с количеством удаленных клеток равным примерно 55 мне бы понадобилось огромное количество времени. Однако здесь можно прибегнуть к двум хитростям.

Во-первых, мы можем воспользоваться полезным инструментом языка C++. А именно работа с потоками. Мы можем запустить рекурсивную функцию в отдельном потоке. На первый взгляд преимущество не очевидно. Но C++ позволяет нам прекратить работу с потоком тогда, когда мы этого захотим. И я решил воспользоваться такой возможностью. Я запускал мою функцию по удалению клеток в отдельном потоке и заставлял основной поток подождать 500 миллисекунд. По прошествии данного времени возможно два варианта: функция либо закончила свою работу, либо нет.

Если функция закончила свою работу вовремя, то мы просто сохраняем поле.

Если же функция не успела завершить свою работу вовремя, то мы принудительно завершаем ее выполнение. Для отслеживания работы функции в ней была предусмотрена переменная, выполняющая роль флага. Если функция завершила работу, то данный флаг принимал значение true. Если не завершила, то значение флага оставалось тем же, что и по умолчанию – false.

После принудительного завершения функции мы возвращаемся к полю, которое было сгенерировано ранее и повторяем выполнение функции удаления клеток.

Во-вторых, мы можем в качестве начального поля отталкиваться уже от ранее сгенерированных. Это нам позволит значительно сэкономить время.

Также рассмотрим человеческий фактор, который следует учитывать во внимание при генерации поля. Моя реализация имеет небольшой недостаток. В ней не предусмотрен человеческий фактор. При решении Судоку человек не может опираться на машиноориентированные алгоритмы. И при генерации поля может возникнуть случай, когда удаленных клеток хватает для решения компьютером данного уровня, а человеку может не хватать начальной информации. Из-за этого возникает ряд случаев, при которых человек не может решить уровень по предоставленной информации. Мой алгоритм не предусматривает данный фактор, так как это приведет к существенному усложнению алгоритма генерации поля и значительному увеличению времени генерации, что не является желательным. С целью экономии времени, ресурсов компьютера и оптимизации алгоритма генерации мною не был принят данный фактор во внимание.

2 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ

2.1 Структура входных и выходных данных

На вход программы, в зависимости от выбранного уровня сложности, поступает игровое поле с некоторым количеством свободных клеток. В зависимости от выбранного уровня сложности, программа загрузит случайный уровень из десяти. Отличие уровней заключается в количестве пустых клеток: для легкого уровня – 35 - 40 пустых клеток; для среднего – 45 - 50; для сложного – 50 - 60. Если пользователь передумает проходить уровень выбранной сложности, предусмотрена возможность возврата к этапу выбора сложности и выбора более легкого уровня.

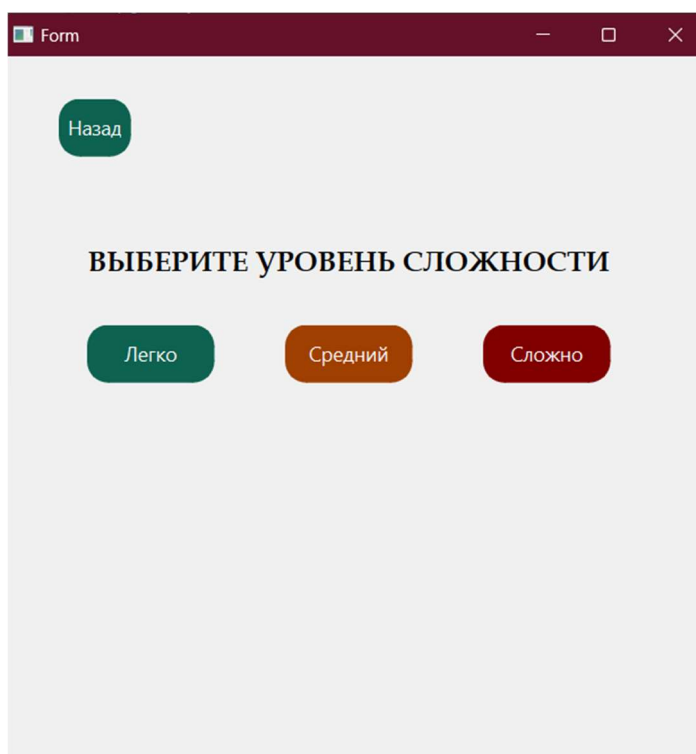
The image shows a screenshot of a software application window titled "Form". The window has a dark red title bar with standard Windows window controls (minimize, maximize, close). The main content area has a light gray background. In the top-left corner, there is a green rounded rectangular button with the text "Назад" (Back) in white. In the center of the screen, the text "ВЫБЕРИТЕ УРОВЕНЬ СЛОЖНОСТИ" (Choose difficulty level) is displayed in a bold, black, sans-serif font. Below this text, there are three rounded rectangular buttons arranged horizontally: a green button labeled "Легко" (Easy), an orange button labeled "Средний" (Medium), and a red button labeled "Сложно" (Hard). All buttons have white text.

Рисунок 2.1 – Выбор уровня сложности

9	8	2	4	0	0	1	0	0
6	0	7	1	9	2	8	0	4
0	0	4	7	8	0	2	0	9
0	0	8	0	4	3	5	9	0
4	0	9	5	1	7	0	8	6
0	6	0	0	2	0	7	4	0
8	0	0	2	6	1	0	7	5
2	0	0	8	5	0	0	1	3
0	4	1	0	7	9	0	0	0

Рисунок 2.2 – Файл игрового поля “NFieldP.txt”

2.2 Разработка диаграммы классов

Диаграмма классов приведена в приложении А.

2.3 Описание классов

2.3.1 Класс Field

Данный класс отвечает за хранение всей игры.

Описание полей класса:

Cell** field – игровое поле

Solver* slv – поле ответственное за решение головоломки

Generator* gen – поле ответственное за генерацию головоломки

Описание методов класса:

Field() – конструктор класса

~Field() – деструктор класса

bool check_field() – проверка на корректность игрового поля

bool load(std::string file_name) – загрузить уровень с заданным названием

bool save(std::string file_name) – сохранить уровень с заданным названием

2.3.2 Класс Cell

Данный класс отвечает за клетку игрового поля.

Описание полей класса:

int value – значение клетки

`int is_locked` – отвечает за возможность изменения значения клетки

Описание методов класса:

`Cell()` – конструктор класса

2.3.3 Класс Generator

Данный класс отвечает за генерацию игрового поля.

Описание методов класса:

`Generator()` – конструктор класса

`void generate(Field::Cell** &field, int difficulty)` – генерация игрового поля заданного уровня сложности.

2.3.4 Класс Solver

Данный класс отвечает за решение головоломки.

Описание методов класса:

`Solver()` – конструктор класса

`bool solve(Field::Cell** &field)` – метод, который решает головоломку

2.3.5 Класс Gamewidget

Данный класс представляет собой графический интерфейс, а также описывает некоторые методы игрового процесса, которые должны отображаться на экране.

Все использованные виджеты наследуются от встроенного в Qt класса `QWidget`. В данных виджетах вместе с модификаторами `public`, `private`, `protected` используются методы, имеющие модификатор `signals` (сигналы) и модификатор `public/private slots` (слоты). Сигнал вырабатывается, когда происходит определенное событие. Слот – это функция, которая вызывается в ответ на определенный сигнал.

Описание методов класса:

`explicit gamewidget(QWidget *parent = nullptr)` – конструктор класса

`~gamewidget()` – деструктор класса

`void set_value()` – установка значения клетки

`void onCellClicked()` – запоминание последней нажатой клетки

`void load()` – загрузка игрового уровня

`void check()` – проверка на корректность поля

`void solve()` – решение уровня

`void reset()` – сброс решения

`void timerSlot()` – отвечает за работу секундомера

`void play()` – начало игры
`void exit()` – выход из игры
`void easy()` – установить уровень сложности на легкий
`void medium()` – установить уровень сложности на средний
`void hard()` – установить уровень сложности на сложный
`void back_to_menu()` – вернуться в главное меню
`void back_to_difficulty()` – вернуться к выбору уровня сложности
`void hint()` – получить подсказку

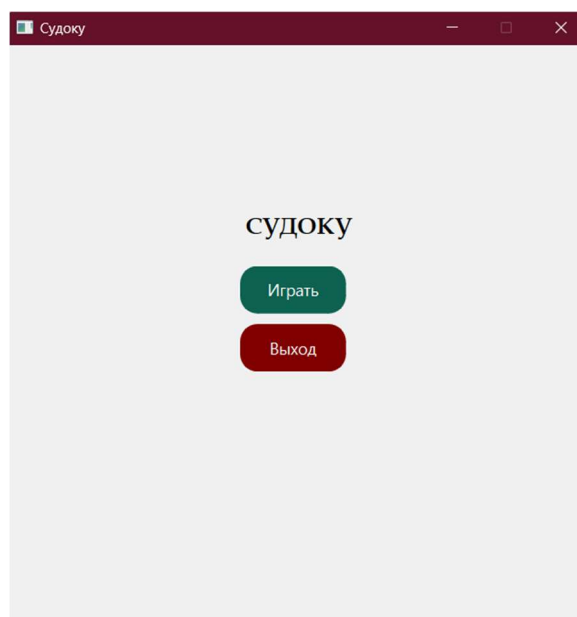


Рисунок 2.3 – Интерфейс главного меню

В главном меню присутствует две кнопки. Первая кнопка – “Играть”. Данная кнопка позволяет перейти к разделу выбора уровня сложности и дальнейшей игре. Вторая кнопка – “Выход”. Позволяет выйти из игры.

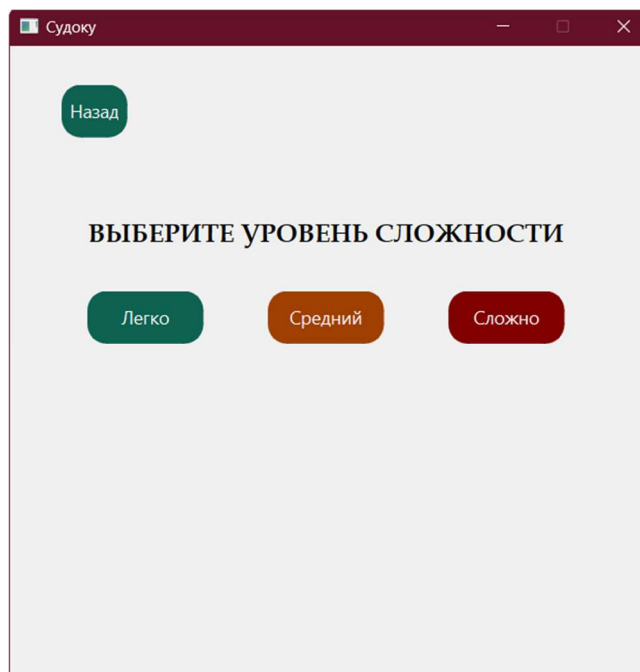


Рисунок 2.4 – Интерфейс раздела выбора уровня сложности

В меню выбора сложности присутствует три кнопки. Кнопка “Легко” позволяет перейти к игре на легком уровне сложности. Легкий уровень сложности определяет количество удаленных клеток в диапазоне от 35 до 40 клеток. Кнопка “Средне” позволяет перейти к игре на среднем уровне сложности. Средний уровень сложности определяет количество удаленных клеток в диапазоне от 45 до 50 клеток. Кнопка “Сложно” позволяет перейти к игре на сложном уровне сложности. Сложный уровень сложности определяет количество удаленных клеток в диапазоне от 50 до 60 клеток. После выбора конкретного уровня сложности игрок переходит к игровому полю. На первый взгляд при выборе разных уровней сложности игровые поля визуально не отличаются. Однако, изменяется функционал кнопки “Запуск”, описанной далее. В зависимости от выбранного уровня сложности кнопка будет загружать разные уровни.

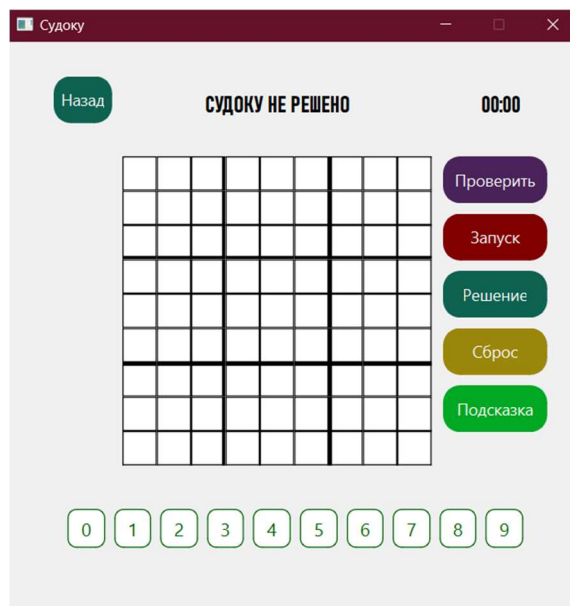


Рисунок 2.5 – Игровой раздел

Игровой раздел состоит из 3 частей: поле, числа и служебные кнопки.

Поле представляет собой набор 81 кнопки, которые формируют игровое поле игры. Назначение данных кнопок заключается в том, что при нажатии на клетку поля запоминается последняя нажатая кнопка. Чтобы заполнить клетку поля нужно нажать на раздел чисел, который находится ниже.

Раздел чисел представляет собой набор кнопок со значением от 0 до 9. Кнопки со значением от 1 до 9 предназначены для вставки в игровое поля выбранного числа. Кнопка “0” предназначена для очистки игрового поля.

Раздел со служебными кнопками предназначен для процесса игры. Кнопка “Проверить” проверяет игровое поле на соблюдение правил. Если игровое поле удовлетворяет правилам, то текстовое поле над игровым полем принимает значение “Судоку решено!”. В противном случае – “Судоку не решено!”.

Кнопка “Запуск” загружает в игровое поле случайный уровень заданной сложности из 10 заранее сгенерированных.

Кнопка “Решение” загружает в поле уже решенный уровень, который ранее был загружен.

Кнопка “Сброс” убирает все введенные пользователем числа из клеток, оставляя только поле, которое было ранее загружено. Кнопка “Подсказка” заполняет случайную клетку правильным значением.

3 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ

3.1 Разработка схем алгоритмов

Функция `annealing()` производит генерацию игрового поля. Схема алгоритма приведена в приложении Б.

Метод `solve()` класса `Solver` решает уровень. Схема алгоритма приведена в приложении В.

3.2 Разработка алгоритмов

3.2.1 Разработка алгоритма функции `annealing()`

Функция `annealing()` производит генерацию игрового поля методом отжига.

Шаг 1. Начало.

Шаг 2. Устанавливаем начальное значение температуры, конечное значение температуры, коэффициент остывания.

Шаг 3. Инициализируем игровое поле с помощью функции `field_init(curr)`.

Шаг 4. Создаем временное хранилище игрового поля, которое будем изменять в процессе алгоритма, и помещаем в него текущее игровое поле.

Шаг 5. Запоминаем количество совпадающих чисел в игровом поле.

Шаг 6. Перемешиваем два случайных числа в случайной строке и сравниваем количество совпадающих чисел.

Шаг 7. Если оно меньше предыдущего, то переходим в это состояние поля. Если больше или равно – оставляем изначальное состояние поля.

Шаг 8. Уменьшаем значение температуры на коэффициент остывания.

Шаг 9. Повторяем шаги 5 – 8 пока текущая температура не станет равна конечной.

Шаг 10. Сохраняем текущее состояние игрового поля.

Шаг 11. Конец.

3.2.2 Разработка алгоритма метода `solve()`

Метод `solve()` класса `Solver` решает уровень. В данной функции также можно было реализовать возможность возврата `true`, в случае если данное поле возможно решить, и `false`, если невозможно. Однако так как данный алгоритм в основном предназначен для генерации поля, в реализации данной функции нет нужды.

Шаг 1. Начало.

Шаг 2. Находим пустую клетку на поле.

Шаг 3. Пытаемся подставить каждое число от 1 до 9.

Шаг 4. Если игровое поле с данным числом в клетке не нарушает правил игры, то оставляем его и повторяем шаг 2 и 3, пока не заполним все пустые клетки или же не сможем подставить все числа от 1 до 9 так, чтобы не нарушить правила.

Шаг 5. Конец.

4 РЕЗУЛЬТАТ РАБОТЫ

После нажатия кнопки “Играть” в главном меню, мы попадаем в меню выбора сложности, где игрок может выбрать один из трех предложенных уровней сложности.

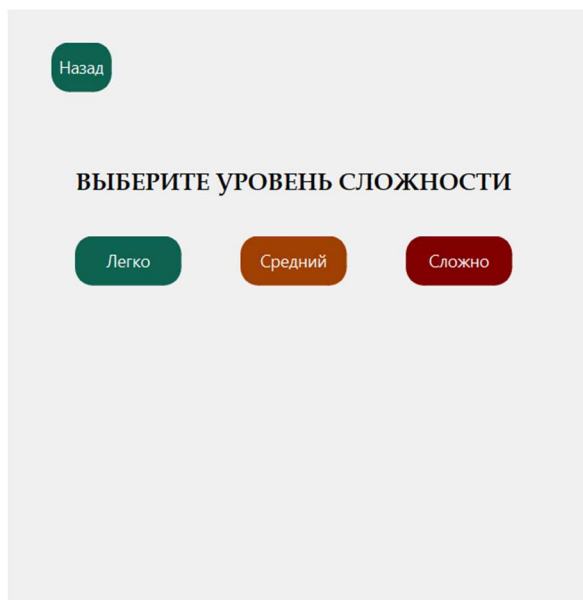


Рисунок 4.1 – Меню выбора уровня сложности

В меню выбора сложности присутствует три кнопки. Кнопка “Легко” позволяет перейти к игре на легком уровне сложности. Легкий уровень сложности определяет количество удаленных клеток в диапазоне от 35 до 40 клеток. Кнопка “Средне” позволяет перейти к игре на среднем уровне сложности. Средний уровень сложности определяет количество удаленных клеток в диапазоне от 45 до 50 клеток. Кнопка “Сложно” позволяет перейти к игре на сложном уровне сложности. Сложный уровень сложности определяет количество удаленных клеток в диапазоне от 50 до 60 клеток.

После выбора уровня сложности игрок попадает в игровую область (рисунок 4.2).

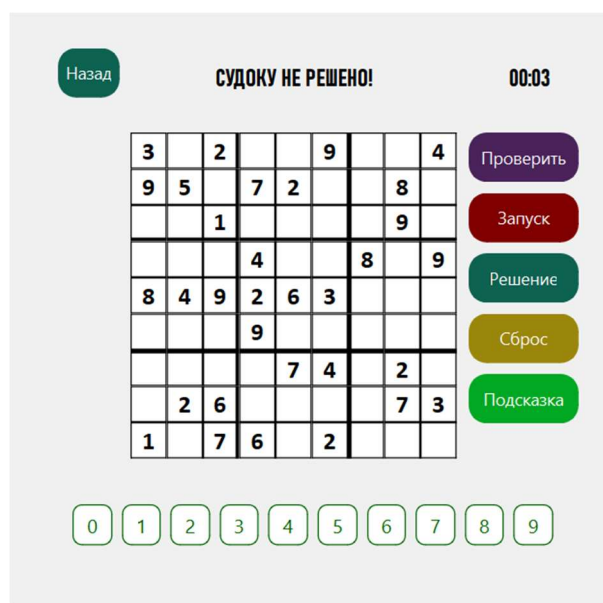


Рисунок 4.2 – Игровая область

Заполнение поля происходит следующим образом. Игрок нажимает на клетку, в которую хочет вставить число и нажимает на числа снизу. При полном заполнении игрового поля, игрок может нажать кнопку “Проверка”. Если игрок правильно решил уровень, то надпись над игровым полем покажет “Судоку решено!”. Иначе – “Судоку не решено!”.

Кнопка “Запуск” позволяет загрузить уровни, сгенерированные заранее. Моя реализация не поддерживает генерацию игровых уровней в реальном времени по причинам, описанных ранее.

В случае если игрок хочет начать уровень заново, он может воспользоваться кнопкой “Сброс”. Данная кнопка сохраняет загруженный уровень и восстанавливает его в случае надобности.

На игровом поле также присутствует секундомер, чтобы игрок мог отслеживать свой прогресс по скорости в прохождении уровня.

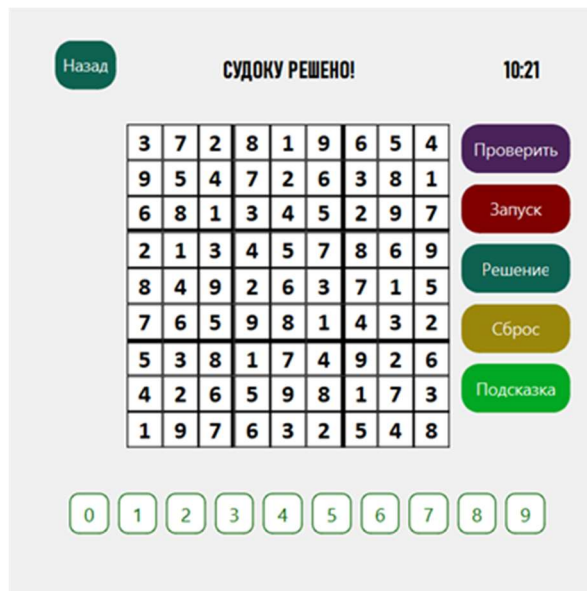


Рисунок 4.3 – Пример пройденного уровня

В случае если игрок заполнил все игровое поле, он может нажать кнопку проверить. В случае если Судоку решено верно, то текстовое поле выше покажет надпись “Судоку решено!”. В случае если Судоку решено неверно, то текстовое поле покажет надпись “Судоку не решено”. После прохождения игрок может вернуться в меню выбора сложности, пройти данный уровень заново, нажав кнопку “Сброс”.

ЗАКЛЮЧЕНИЕ

В процессе выполнения данной курсовой работы была разработана игра Судoku на языке программирования C++ с использованием фреймворка Qt. В процессе работы были изучены основы создания графического интерфейса с помощью Qt, а также реализованы алгоритмы генерации и решения игрового поля.

Программа предоставляет пользователю удобный и интуитивно понятный интерфейс для игры в Судoku, позволяя выбирать уровень сложности и решать игровые уровни. Были реализованы функции генерации и решения уровней, проверки правильности заполнения игрового поля и подсказок для помощи игроку в решении головоломки.

Данная работа позволила углубить знания в области программирования на C++, ознакомиться с применением библиотеки Qt для создания графических приложений и практически применить алгоритмы решения головоломки “Судoku”.

Реализованный проект также предоставляет пользователю возможность не только играть в “Судoku”, но и изучать методы решения головоломки, следить за своими успехами и улучшать навыки логического мышления.

В целом, данная работа позволила получить новые знания о языке программирования C++, понять особенности работы с фреймворком Qt и научиться применять полученные знания для разработки программ, которые я смогу применить для создания более сложных программ.

Код программы приведен в приложении Г.

СПИСОК ЛИТЕРАТУРЫ

Стивен Прата. Язык программирования C++. Лекции и упражнения: учеб. пособие / С. Прата – СПб.: ООО «ДиаСофтЮП», 2003. – 1104с.

Программирование на C++ [Электронный ресурс]. - Режим доступа: <https://metanit.com/cpp/tutorial/>. - Дата доступа: 23.11.2023.

Qt Documentation [Электронный ресурс]. -Электронные данные. -Режим доступа: <https://doc.qt.io/> Дата доступа: 15.11.2023

А. В.Чеботарев Библиотека Qt 4. Программирование прикладных приложений в среде Linux / А. В.Чеботарев - Л.: Наука, 2013. - 821 с.

Страуструп, Б. Язык программирования C++ / Б. Страуструп. - М.: БИНОМ, 2004.- 1098 с.

ПРИЛОЖЕНИЕ А
(обязательное)
Диаграмма классов

ПРИЛОЖЕНИЕ Б
(обязательное)
Схема функции `annealing()`

ПРИЛОЖЕНИЕ В

(обязательное)

Схема алгоритма метода `solve()`

ПРИЛОЖЕНИЕ Г

(обязательное)

Код программы

```
Файл Cell.h:
#ifndef CELL_H
#define CELL_H
#include "Field.h"
class Field::Cell
{
public:
    Cell() {
        value = 0;
        is_locked = 0;
    }
    int value, is_locked;
};
#endif // CELL_H
```

```
Файл Field.h:
#ifndef FIELD_H
#define FIELD_H
#include <iostream>
class Cell;
class Field {
public:
    class Cell;
    class Generator;
    class Solver;
    Field();
    ~Field();
    bool check_field();
    bool load(std::string file_name);
    void save(std::string file_name);
    Cell** field;
    Solver* slv;
    Generator* gen;
};
#endif // FIELD_H
```

```
Файл Gamewidget.h:
#ifndef GAMEWIDGET_H
#define GAMEWIDGET_H
#include <QWidget>
#include <QPushButton>
#include <QTimer>
#include "Field.h"
namespace Ui {
class gamewidget;
}
class gamewidget : public QWidget
```

```

{
    Q_OBJECT
public:
    explicit gamewidget(QWidget *parent = nullptr);
    ~gamewidget();
private:
    Ui::gamewidget *ui;
    QPushButton* selectedCell;
    Field* _field;
    QTimer* timer;
    int sec, min, difficulty;
    std::string filename;
private slots:
    void set_value();
    void onCellClicked();
    void generate();
    void load();
    void check();
    void solve();
    void reset();
    void timerSlot();
    void play();
    void exit();
    void easy();
    void medium();
    void hard();
    void back_to_menu();
    void back_to_difficulty();
    void hint();
};
#endif // GAMEWIDGET_H

Файл Generator.h:
#ifndef GENERATOR_H
#define GENERATOR_H
#include "Field.h"
#include "Field.h"
class Field::Generator
{
public:
    Generator() = default;
    void generate(Field::Cell** &field, int difficulty);
};
#endif // GENERATOR_H

Файл Solver.h:
#ifndef SOLVER_H
#define SOLVER_H
#include "Cell.h"
#include <iostream>
#include <fstream>
#include <sstream>

```



```

class Field::Solver {
public:
    Solver() = default;
    bool solve(Field::Cell** &field);
};
#endif // SOLVER_H

Файл Cell.cpp:
#include "cell.h"

Файл Field.cpp:
#include "Field.h"
#include "Cell.h"
#include "Solver.h"
#include "Generator.h"
#include <iostream>
#include <iomanip>
#include <fstream>
#include <sstream>
#include <string>
#include <QFile>
#include <QTextStream>
#include <QStringList>
#include <QDebug>
Field::Field()
{
    field = new Field::Cell * [9];
    for (int i = 0; i < 9; i++) {
        field[i] = new Field::Cell [9];
    }
    gen = new Generator;
    slv = new Solver;
}
Field::~Field()
{
    for (int i = 0; i < 9; i++) {
        delete[] field[i];
    }
    delete[] field;
}
bool Field::load(std::string file_name) {
    std::ifstream file(file_name, std::ios::in);
    if(file.is_open()){
        std::string line;
        int row = 1, num;
        int i = 0;
        while (getline(file, line) && row <= 11) {
            int j = 0;
            if (row % 4 == 0) {
                row++;
                continue;
            }

```

```

        }
        std::istringstream str(line);
        while (str >> num && j < 9) {
            field[i][j].value = num;
            if(field[i][j].value!=0)
                field[i][j].is_locked=1;
            j++;
        }
        i++;
        row++;
    }
    file.close();
}
else{
    return false;
}
}

void Field::save(std::string file_name) {
    std::ofstream file;
    file.open(file_name);
    for (int i = 0; i < 3; i++) {
        for (int a = 0; a < 3; a++) {
            for (int j = 0; j < 3; j++) {
                for (int k = 0; k < 3; k++) {
                    file << std::setw(3) << field[i * 3 + a][j * 3 +
k].value;
                }
                file << " ";
            }
            file << "\n";
        }
        file << "\n";
    }

    file.close();
}

int column_duplicates0(Field::Cell** (&arr)) {
    int count = 0;
    int size = 9;
    for (int i = 0; i < size; ++i) {
        for (int k = 0; k < 8; k++) {
            for (int j = k + 1; j < size; ++j) {
                if (arr[k][i].value == arr[j][i].value) {
                    count++;
                    break;
                }
            }
        }
    }
    return count;
}

```

```

int check_sector0(int ind, int jnd, Field::Cell** mas) {
    int f = 0;
    for (int m = 1; m <= 9; m++) {
        int f1 = -1;
        for (int i = ind; i < ind + 3; i++) {
            for (int j = jnd; j < jnd + 3; j++) {
                if (mas[i][j].value == m) {
                    f1++;
                    if (f1 > 0)
                        f++;
                }
            }
        }
    }
    return f;
}

int check_sectors0(Field::Cell** mas) {
    int f = 0;
    for (int i = 0; i < 9; i += 3) {
        for (int j = 0; j < 9; j += 3) {
            f += check_sector0(i, j, mas);
        }
    }
    return f;
}

int matches0(Field::Cell** mas) {
    int f = 0;
    f += column_duplicates0(mas);
    f += check_sectors0(mas);
    return f;
}

bool Field::check_field(){
    for(int i=0;i<9;i++){
        for(int j=0; j<9;j++){
            if(field[i][j].value == 0)
                return false;
        }
    }
    for(int i=0;i<9;i++){
        for(int val=1; val<10;val++){
            int f=0;
            for(int j=0;j<9;j++){
                if(field[i][j].value == val)
                    f++;
            }
            if(f>1)
                return false;
        }
    }
    if(matches0(field)>0)
        return false;
    return true;
}

```

```
}
```

Файл gamewidget.cpp:

```
#include "gamewidget.h"
#include "ui_gamewidget.h"
#include "cell.h"
#include "generator.h"
#include "solver.h"
class Field;
class Cell;
class Generator;
class Solver;
gamewidget::gamewidget(QWidget *parent) :
    QWidget(parent),
    ui(new Ui::gamewidget)
{
    this->setFixedSize(500,500);
    timer = new QTimer(this);
    ui->setupUi(this);
    ui->playWidget->hide();
    ui->difficultyWidget->hide();
    connect(ui->back_to_menu_button, SIGNAL(clicked(bool)), this,
    SLOT(back_to_menu()));
    connect(ui->back_to_difficulty_button, SIGNAL(clicked(bool)),
    this, SLOT(back_to_difficulty()));
    connect(ui->pushButton_number_0, SIGNAL(clicked(bool)), this,
    SLOT(set_value()));
    connect(ui->pushButton_number_1, SIGNAL(clicked(bool)), this,
    SLOT(set_value()));
    connect(ui->pushButton_number_2, SIGNAL(clicked(bool)), this,
    SLOT(set_value()));
    connect(ui->pushButton_number_3, SIGNAL(clicked(bool)), this,
    SLOT(set_value()));
    connect(ui->pushButton_number_4, SIGNAL(clicked(bool)), this,
    SLOT(set_value()));
    connect(ui->pushButton_number_5, SIGNAL(clicked(bool)), this,
    SLOT(set_value()));
    connect(ui->pushButton_number_6, SIGNAL(clicked(bool)), this,
    SLOT(set_value()));
    connect(ui->pushButton_number_7, SIGNAL(clicked(bool)), this,
    SLOT(set_value()));
    connect(ui->pushButton_number_8, SIGNAL(clicked(bool)), this,
    SLOT(set_value()));
    connect(ui->pushButton_number_9, SIGNAL(clicked(bool)), this,
    SLOT(set_value()));
    connect(ui->play_button, SIGNAL(clicked(bool)), this,
    SLOT(play()));
    connect(ui->exit_button, SIGNAL(clicked(bool)), this,
    SLOT(exit()));
    connect(ui->easy_button, SIGNAL(clicked(bool)), this,
    SLOT(easy()));
}
```

```

    connect(ui->medium_button, SIGNAL(clicked(bool)), this,
    SLOT(medium()));
    connect(ui->hard_button, SIGNAL(clicked(bool)), this,
    SLOT(hard()));
    connect(ui->load_button, SIGNAL(clicked(bool)), this,
    SLOT(load()));
    connect(ui->solve_button, SIGNAL(clicked(bool)), this,
    SLOT(solve()));
    connect(ui->check_button, SIGNAL(clicked(bool)), this,
    SLOT(check()));
    connect(ui->reset_button, SIGNAL(clicked(bool)), this,
    SLOT(reset()));
    connect(ui->hint_button, SIGNAL(clicked(bool)), this,
    SLOT(hint()));
    connect(timer, SIGNAL(timeout()), this, SLOT(timerSlot()));
    for(int i=0;i<=8;i++){
        QString buttonName = "pushButton_0" + QString::number(i);
        QPushButton *cellButton = findChild<QPushButton
*>(buttonName);
        if (cellButton) {
            connect(cellButton, SIGNAL(clicked(bool)), this,
    SLOT(onCellClicked()));
        }
    }
    for (int i = 10; i <= 88; i++) {
        if(i%10==9)
            continue;
        QString buttonName = "pushButton_" + QString::number(i);
        QPushButton *cellButton = findChild<QPushButton
*>(buttonName);
        if (cellButton) {
            connect(cellButton, SIGNAL(clicked(bool)), this,
    SLOT(onCellClicked()));
        }
    }
    _field = new Field();
    selectedCell = NULL;
    sec = 0;
    min = 0;
}
gamewidget::~gamewidget()
{
    delete ui;
}
void gamewidget::set_value()
{
    QPushButton *clickedButton =
qobject_cast<QPushButton*>(sender());
    QString buttonText = selectedCell->objectName();
    int i = buttonText[11].unicode() - '0';
    int j = buttonText[12].unicode() - '0';
    if(selectedCell != NULL && _field->field[i][j].is_locked==0){

```

```

        if (clickedButton && selectedCell) {
            QString number = clickedButton->text();
            if(number == "0")
                selectedCell->setText("");
            else
                selectedCell->setText(number);
        }
    }
}

void gamewidget::onCellClicked() {
    QPushButton *clickedButton =
qobject_cast<QPushButton*>(sender());
    if (clickedButton) {
        selectedCell = clickedButton;
    }
}

void gamewidget::generate(){
    _field->gen->generate(_field->field, 30);
    for (int i = 0; i < 9; i++) {
        for(int j=0;j<9;j++){
            QString buttonName = "pushButton_" + QString::number(i) +
QString::number(j);
            QPushButton *cellButton =
findChild<QPushButton*>(buttonName);
            QString buttonText = QString::number(_field-
>field[i][j].value);
            if(buttonText == "0"){
                buttonText = "";
            }
            cellButton->setText(buttonText);
        }
    }
    QLabel *text_label = findChild<QLabel*>("text");
    text_label->setText("Судоку не решено!");
    min = 0;
    sec=0;
    timer->start(1000);
}

void gamewidget::load(){
    std::string baseFileName = "Field";
    std::string extension = ".txt";
    std::string fileName = std::to_string(difficulty) +
baseFileName + std::to_string(rand()%10) + extension;
    filename=fileName;
    if(_field->load(fileName)){
        for (int i = 0; i < 9; i++) {
            for(int j=0;j<9;j++){
                QString buttonName = "pushButton_" + QString::number(i) +
QString::number(j);
                QPushButton *cellButton =
findChild<QPushButton*>(buttonName);

```

```

        QString buttonText = QString::number(_field-
>field[i][j].value);
        if(buttonText == "0"){
            buttonText = "";
            _field->field[i][j].is_locked=0;
        }
        cellButton->setText(buttonText);
    }
}
QLabel *text_label = findChild<QLabel*>("text");
text_label->setText("Судoku не решено!");
min = 0;
sec=0;
timer->start(1000);
}
else{
QLabel *text_label = findChild<QLabel*>("text");
text_label->setText("проблема с файлом уровня!");
}
}
void gamewidget::solve() {
    _field->load(filename);
    _field->slv->solve(_field->field);
    for (int i = 0; i < 9; i++) {
        for(int j=0;j<9;j++){
            QString buttonName = "pushButton_" + QString::number(i) +
QString::number(j);
            QPushButton *cellButton =
findChild<QPushButton*>(buttonName);
            QString buttonText = QString::number(_field-
>field[i][j].value);
            if(buttonText == "0"){
                buttonText = "";
            }
            cellButton->setText(buttonText);
        }
    }
    QLabel *text_label = findChild<QLabel*>("text");
    text_label->setText("Судoku решено!");
    timer->stop();
}
void gamewidget::check() {
    for (int i = 0; i < 9; i++) {
        for(int j=0;j<9;j++){
            QString buttonName = "pushButton_" + QString::number(i) +
QString::number(j);
            QPushButton *cellButton =
findChild<QPushButton*>(buttonName);
            QString buttonText = cellButton->text();
            if(buttonText == "")
                _field->field[i][j].value = 0;
            else

```

```

        _field->field[i][j].value = buttonText.toInt();
    }
}
if(_field->check_field() == true){
    QLabel *text_label = findChild<QLabel*>("text");
    text_label->setText("Судоку решено!");
    timer->stop();
}
else {
    QLabel *text_label = findChild<QLabel*>("text");
    text_label->setText("Судоку не решено!");
}
}

void gamewidget::reset(){
    if(_field->load(filename)){
        for (int i = 0; i < 9; i++) {
            for(int j=0;j<9;j++){
                QString buttonName = "pushButton_" + QString::number(i) +
                QString::number(j);
                QPushButton *cellButton =
                findChild<QPushButton*>(buttonName);
                QString buttonText = QString::number(_field-
                >field[i][j].value);
                if(buttonText == "0"){
                    buttonText = "";
                }
                cellButton->setText(buttonText);
            }
        }
        QLabel *text_label = findChild<QLabel*>("text");
        text_label->setText("Судоку не решено!");
        min=0;
        sec=0;
        timer->start();
    }
    else{
        QLabel *text_label = findChild<QLabel*>("text");
        text_label->setText("проблема с файлом уровня!");
    }
}

void gamewidget::timerSlot(){
    sec++;
    if(sec>59){
        sec = 0;
        min++;
    }
    QLabel *timer_label = findChild<QLabel*>("timer");

    QString formattedTime = QString("%1:%2")
        .arg(min, 2, 10, QChar('0'))
        .arg(sec, 2, 10, QChar('0'));
}

```



```

    timer_label->setText(formattedTime);
}
void gamewidget::play() {
    ui->menuWidget->hide();
    ui->difficultyWidget->show();
}
void gamewidget::exit() {
    this->close();
}
void gamewidget::easy() {
    ui->playWidget->show();
    ui->difficultyWidget->hide();
    difficulty=1;
}
void gamewidget::medium() {
    ui->playWidget->show();
    ui->difficultyWidget->hide();
    difficulty=2;
}
void gamewidget::hard() {
    ui->playWidget->show();
    ui->difficultyWidget->hide();
    difficulty=3;
}
void gamewidget::back_to_menu() {
    ui->menuWidget->show();
    ui->difficultyWidget->hide();
}
void gamewidget::back_to_difficulty() {
    for(int i=0;i<9;i++){
        for(int j=0;j<9;j++){
            _field->field[i][j].is_locked=0;
        }
    }
    timer->stop();
    min=0;
    sec=0;
    QLabel *timer_label = findChild<QLabel*>("timer");

    QString formattedTime = QString("%1:%2")
        .arg(min, 2, 10, QChar('0'))
        .arg(sec, 2, 10, QChar('0'));
    for (int i = 0; i < 9; i++) {
        for(int j=0;j<9;j++){
            QString buttonName = "pushButton_" + QString::number(i) +
            QString::number(j);
            QPushButton *cellButton =
            findChild<QPushButton*>(buttonName);
            QString buttonText = QString::number(0);
            if(buttonText == "0"){
                buttonText = "";
            }
        }
    }
}

```

```

    }
    cellButton->setText(buttonText);
    }
}
timer_label->setText(formattedTime);
ui->playWidget->hide();
ui->difficultyWidget->show();
}
void gamewidget::hint() {
    int** temp = new int* [9];
    for(int i=0;i<9;i++){
        temp[i] = new int [9];
    }
    for(int i=0;i<9;i++){
        for(int j=0;j<9;j++){
            temp[i][j] = _field->field[i][j].value;
        }
    }
    int** solved = new int* [9];
    for(int i=0;i<9;i++){
        solved[i] = new int [9];
    }
    for(int i=0;i<9;i++){
        for(int j=0;j<9;j++){
            solved[i][j] = _field->field[i][j].value;
        }
    }
    _field->slv->solve(_field->field);
    for(int i=0;i<9;i++){
        for(int j=0;j<9;j++){
            std::swap(solved[i][j], _field->field[i][j].value);
        }
    }
    for (int i = 0; i < 9; i++) {
        for(int j=0;j<9;j++){
            QString buttonName = "pushButton_" + QString::number(i) +
            QString::number(j);
            QPushButton *cellButton =
            findChild<QPushButton*>(buttonName);
            QString buttonText = cellButton->text();
            if(buttonText == "")
                temp[i][j] = 0;
            else
                temp[i][j] = buttonText.toInt();
        }
    }
    int f=0;
    for(int i=0;i<9;i++){
        for(int j=0;j<9;j++){
            if(temp[i][j]!=solved[i][j]){
                f++;
            }
        }
    }
}

```

```

    }
}
if(f>0){
    int row = 0, col = 0;
    do{
        row = rand()% 9;
        col = rand()% 9;
    }while(temp[row][col] == solved[row][col]);
    QString buttonName = "pushButton_" + QString::number(row) +
    QString::number(col);
    QPushButton* cellButton =
    findChild<QPushButton*>(buttonName);
    QString buttonText = QString::number(solved[row][col]);
    cellButton->setText(buttonText);
    if(f==1){
        QLabel *text_label = findChild<QLabel*>("text");
        text_label->setText("Судоку решено!");
        timer->stop();
    }
}
else{
    QLabel *text_label = findChild<QLabel*>("text");
    text_label->setText("Судоку решено!");
    timer->stop();
}
for(int i=0;i<9;i++){
    delete [] temp[i];
}
delete [] temp;
for(int i=0;i<9;i++){
    delete [] solved[i];
}
delete [] solved;
}

```

Файл Generator.cpp:

```

#include "Generator.h"
#include "Cell.h"
#include <chrono>
#include <thread>
#include <mutex>
#include <random>
#include <iomanip>
std::mutex fieldMutex;
void _print(Field::Cell** field)
{
    std::cout << "\n";
    for (int i = 0; i < 3; i++) {
        for (int a = 0; a < 3; a++) {
            for (int j = 0; j < 3; j++) {
                for (int k = 0; k < 3; k++) {

```

```

        std::cout << std::setw(3) << field[i * 3 + a][j * 3 +
k].value;
    }
    std::cout << " ";
}
std::cout << "\n";
}
std::cout << "\n";
}
std::cout << "\n";
}
bool is_valid(Field::Cell** field, int row, int col, int value)
{
    for (int c = 0; c < 9; c++)
    {
        if (field[row][c].value == value)
            return false;
    }
    for (int r = 0; r < 9; r++)
    {
        if (field[r][col].value == value)
            return false;
    }
    int startRow = floor(row / 3) * 3, startCol = floor(col / 3) *
3;
    for (int r = startRow; r < startRow + 3; r++)
    {
        for (int c = startCol; c < startCol + 3; c++)
        {
            if (field[r][c].value == value)
                return false;
        }
    }
    return true;
}
bool solve(Field::Cell** &field) {
    for (int i = 0; i < 9; i++) {
        for (int j = 0; j < 9; j++) {
            if (field[i][j].value == 0) {
                for (int value = 1; value <= 9; value++) {
                    int number_of_sol = 0;
                    if (is_valid(field, i, j, value)) {
                        number_of_sol++;
                        if (number_of_sol > 1)
                            return false;
                        field[i][j].value = value;
                        if (solve(field)) {
                            field[i][j].value = 0;
                            return true;
                        }
                    }
                }
            }
        }
    }
}

```

```

        return false;
    }
}
return true;
}
void deleting(Field::Cell** &field, int difficulty, bool
&finished) {
    std::random_device rd;
    std::mt19937 gen(rd());
    std::uniform_int_distribution<int> distribution(0, 8);
    int attemp = difficulty;
    int row, col, max = 0;
    while (attemp > 0) {
        int cnt = 0;
        row = distribution(gen);
        col = distribution(gen);
        while (field[row][col].value == 0) {
            row = distribution(gen);
            col = distribution(gen);
        }
        int temp = field[row][col].value;
        std::lock_guard<std::mutex> lock(fieldMutex);
        field[row][col].value = 0;
        if (solve(field)) {
            field[row][col].value = 0;

            for (int i = 0; i < 9; i++)
                for (int j = 0; j < 9; j++)
                    if (field[i][j].value == 0)
                        cnt++;
            if ((difficulty - attemp) > cnt) {
                field[row][col].value = temp;
                continue;
            }
            else {
                attemp--;
                max = cnt;
                std::cout << "\n";
                for (int i = 0; i < 3; i++) {
                    for (int a = 0; a < 3; a++) {
                        for (int j = 0; j < 3; j++) {
                            for (int k = 0; k < 3; k++) {
                                std::cout << std::setw(3) << field[i * 3 + a][j
* 3 + k].value;
                                    }
                                std::cout << " ";
                            }
                        std::cout << "\n";
                    }
                std::cout << "\n";
            }
        }
    }
}

```

```

        std::cout << "\n";
        std::cout << max << std::endl;
    }
}
else
    field[row][col].value = temp;
}
finished = true;
}
int check_sector(int ind, int jnd, Field::Cell** mas) {
    int f = 0;
    for (int m = 1; m <= 9; m++) {
        int fl = -1;
        for (int i = ind; i < ind + 3; i++) {
            for (int j = jnd; j < jnd + 3; j++) {
                if (mas[i][j].value == m) {
                    fl++;
                    if (fl > 0)
                        f++;
                }
            }
        }
    }
    return f;
}
int check_sectors(Field::Cell** mas) {
    int f = 0;
    for (int i = 0; i < 9; i += 3) {
        for (int j = 0; j < 9; j += 3) {
            f += check_sector(i, j, mas);
        }
    }
    return f;
}
int check(int val, int ind, Field::Cell* mas) {
    for (int i = 0; i < ind; i++) {
        if (mas[i].value == val)
            return 0;
    }
    return 1;
}
void mix_row(Field::Cell** (&field)) {
    int offset1, offset2, offset3;
    offset1 = rand() % 9;
    offset2 = rand() % 9;
    offset3 = rand() % 9;
    std::swap(field[offset3][offset1].value,
field[offset3][offset2].value);
}
void mixing(Field::Cell** (&field)) {
    mix_row(field);
}

```

```

void field_init(Field::Cell** (&mas)) {
    mas = new Field::Cell * [9];
    for (int i = 0; i < 9; i++) {
        mas[i] = new Field::Cell[9]{};
    }
    for (int i = 0; i < 9; i++) {
        for (int j = 0; j < 9; j++) {
            do {
                mas[i][j].value = rand() % 9 + 1;
            } while (!check(mas[i][j].value, j, mas[i]));
        }
    }
}

int column_duplicates(Field::Cell** (&arr)) {
    int count = 0;
    int size = 9;
    for (int i = 0; i < size; ++i) {
        for (int k = 0; k < 8; k++) {
            for (int j = k + 1; j < size; ++j) {
                if (arr[k][i].value == arr[j][i].value) {
                    count++;
                    break;
                }
            }
        }
    }
    return count;
}

int matches(Field::Cell** mas) {
    int f = 0;
    f += column_duplicates(mas);
    f += check_sectors(mas);
    return f;
}

void copy(Field::Cell** (&to), Field::Cell** from) {
    for (int i = 0; i < 9; i++) {
        for (int j = 0; j < 9; j++)
            to[i][j].value = from[i][j].value;
    }
}

void annealing(Field::Cell**& mas) {
    double initial_temp = 4000.0;
    double final_temp = 0.1;
    double alpha = 0.01;
    double current_temp = initial_temp;
    double num;
    int diff;
    int old_matches, new_matches;
    Field::Cell** curr;
    field_init(curr);
    copy(curr, mas);
    while (matches(curr) != 0) {
        current_temp = initial_temp;

```

```

    field_init(mas);
    copy(curr, mas);
    while (current_temp > final_temp) {
        copy(mas, curr);
        old_matches = matches(curr);
        mixing(mas);
        new_matches = matches(mas);
        diff = old_matches - new_matches;
        num = (double)rand() / (double)RAND_MAX;;
        if (exp(-1 * diff / current_temp) > num && diff >= 0) {
            copy(curr, mas);
        }
        current_temp -= alpha;
    }
}
copy(mas, curr);
}

void Field::Generator::generate(Field::Cell** &_field, int
_difficulty){
    bool finished = false;
    field_init(_field);
    annealing(_field);
    Field::Cell** temp;
    temp = new Cell * [9];
    for (int i = 0; i < 9; i++) {
        temp[i] = new Cell[9];
    }
    for (int i = 0; i < 9; i++) {
        for (int j = 0; j < 9; j++) {
            temp[i][j].value = _field[i][j].value;
        }
    }
    int it = 1;
    _print(_field);
    int f = 0;
    while (true) {
        const int maxExecutionTime = 500;
        std::thread functionThread(deleting, std::ref(_field),
_difficulty, std::ref(finished));
        std::this_thread::sleep_for(std::chrono::milliseconds(maxExecuti
onTime));
        if (!finished) {
            functionThread.detach();
            for (int i = 0; i < 9; i++) {
                for (int j = 0; j < 9; j++) {
                    _field[i][j].value = temp[i][j].value;
                }
            }
            it++;
        }
        else {
            functionThread.detach();

```



```

        f++;
        _print(_field);
        for (int i = 0; i < 9; i++) {
            delete[] temp[i];
        }
        delete[] temp;

        return;
    }
}
}

```

Файл main.cpp:

```

#include <iostream>
#include <ctime>
#include "Field.h"
#include <QtWidgets>
#include <QApplication>
#include "gamewidget.h"
#include "generator.h"
#include "solver.h"
int main(int argc, char *argv[])
{
    srand(time(NULL));
    QApplication a(argc, argv);
    gamewidget w;
    w.show();
    return a.exec();
}

```

Файл Solver.cpp:

```

#include "Solver.h"
#include <iostream>
#include <cmath>
bool is_valid1(Field::Cell** field, int row, int col, int value)
{
    for (int c = 0; c < 9; c++)
    {
        if (field[row][c].value == value)
            return false;
    }
    for (int r = 0; r < 9; r++)
    {
        if (field[r][col].value == value)
            return false;
    }
    int startRow = floor(row / 3) * 3, startCol = floor(col / 3) *
3;
    for (int r = startRow; r < startRow + 3; r++)
    {
        for (int c = startCol; c < startCol + 3; c++)
        {

```

```

        if (field[r][c].value == value)
            return false;
    }
}
return true;
}
bool Field::Solver::solve(Field::Cell** &_field){
    for (int i = 0; i < 9; i++) {
        for (int j = 0; j < 9; j++) {
            if (_field[i][j].value == 0) {
                for (int value = 1; value <= 9; value++) {
                    if (is_valid1(_field, i, j, value)) {
                        _field[i][j].value = value;
                        if (solve(_field)) {
                            return true;
                        }
                    }
                    else
                        _field[i][j].value = 0;
                }
            }
            return false;
        }
    }
}
return true;
}

```

ПРИЛОЖЕНИЕ Д
(обязательное)
Ведомость документов