TALLINN UNIVERSITY OF TECHNOLOGY

Faculty of Information Technology

Department of Computer Engineering

IAY40LT

Иван Бочаров 112660

РЕШАТЕЛЬ СУДОКУ

Диссертация бакалавра

Руководитель: Peeter Ellervee

Степень руководителя: Ph.D., Dipl.Eng.

Должность руководителя: professor

Tallinn 2015

Декларация автора об уникальности работы

Настоящим я подтверждаю, что я являюсь единственным автором этой дипломной работы. Все используемые материалы, ссылки на литературу и работы других авторов были представлены в ссылочных материалах. Данная работа нигде до этого не была представлена на рассмотрение.

Автор: Иван Бочаров

18.05.2015

Аннотация

Главной целью представленной работы было создание автономного кроссплатформенного решателя судоку. При этом для создания решателя нужно было объединить существующие методики решения головоломки судоку в одну программу. Данная программа должна была автоматически решать предложенные пользователем головоломки и выводить полученное решение.

В ходе работы было приведено подробное описание устройства головоломки судоку с пояснением того, что является целью при её решении и какие правила при этом существуют. Также приводилась применяемая в данной работе терминология вместе с пояснением к каждому термину.

В дальнейшем было представлено множество способов того, каким образом происходит поиск решений. К каждому способу было представлено подробное пояснение с иллюстрацией применяемого метода. Также производился разбор работы программы. В итоге была получена рабочая программа, полностью и правильно решающая все варианты стандартной головоломки судоку (поле 9x9).

В дополнение к основной функции решателя были разработаны дополнительные возможности приложения:

- простую проверку возможности решения
- поиск всех возможных значений для каждой пустой клетки
- вывод значений только тех клеток, которые были отмечены
- проверку уникальности решения и количества ответов
- возможность использовать файловую систему и загружать заранее сохраненные судоку.

Итоговая работа была написана на русском языке и содержит в себе 52 страницы текста, 18 глав и 32 рисунка.

Annotatsioon

Sudoku lahendaja

Esitatud töö peamine eesmärk oli luua autonoomne platvormiülene sudoku lahendaja. Samal ajal loodud lahenduseks oli vaja ühendada olemasolevate sudoku meetoditemõistatusi ühte programmi. See programm peaks automaatselt lahendada kasutaja poolt pakutud mõistatusi ja kuvada lahendust.

Töö käigus on tehtud teegid üksikasjalikkuks kirjelduseks seade Sudoku koos selgitusega, mis on selle eesmärk otsuste ja milliseid reegleid on olemas. Samuti kasutatud terminoloogia selgitab selles töös kasutatud mõistusi.

Hiljem näidati mitmeid viise, kuidas korraldatud lahendusi otsimine toimub. Iga meetodi esitati koos illustratsiooni üksikasjaliku selgitusega. Samuti oli tehtud programmitöö analüüs. Tulemuseks on saadud tööversioon programmist, mis täielikult ja õigesti lahendab kõik standartseid Sudoku pusle variante (ruut 9x9).

Lisaks põhifunktsioon lahendajale, on rakendanud lisavõimalusi. Nende hulka:

- lihtne lahenduse kontroll,
- võimalike väärtuste otsimine iga tühja rakki väljal,
- väljundväärtused valitud lahtrite põhjal,
- lahenduste ukinaalsuse kontroll koos mitmete otsingu vastustega,
- võime kasutada failisüsteemi ja alla laadida eelnevalt salvestatud Sudoku.

Lõputöö on kirjutatud vene keeles ning sisaldab teksti 52 leheküljel, 18 peatükki ja 32 joonist.

Abstract

Sudoku solver

The main purpose of this thesis was creation of autonomic cross-platform Sudoku solver. In the same time for creating of solver we needed to combine all existing methods of solving Sudoku in the one program. This program had to automatically solve all presented puzzles that can be input by user, and had to show a founded answer.

During the thesis was explained in detail how is arranged puzzle with the explaining what purpose exist during the solving of Sudoku and what rules existing. Furthermore, were explained a term's that were used during the writing of the thesis and program.

After that were described many methods of that how happens the searching of solutions. For the every way of solving, there was presented very detailed description with a figure of that how it works. Moreover, was made very detailed review of working program. Eventually, was obtained correctly working application that can find an answer for the every standard puzzle (table 9x9).

Except of the standard functions were developed for solver some additional opportunities of application:

- a simple checking of possibility of solving Sudoku,
- searching of all possible values for the every empty cell,
- outputting only of the cells that were selected,
- checking of uniqueness of the solution and the number of possible answers,
- opportunity of using file system for a loading of created in the past file.

The thesis is in Russian language and contains 52 pages of text, 18 chapters and 32 images.

Таблица аббревиатур и правил

IDE	Integrated Development Environment, Интегрированная среда разработки
ICHEC	Irish Centre for High-End Computing
Ячейка	Одна клетка на поле судоку
Квадрат	Одна из выделенных областей 3x3 ячейки поля судоку
Прямая X или Y	Область, покрывающая 9 ячеек слева направо или 9 квадратов сверху вниз соответственно
Действительное значение (или подтвержденное)	Число, при записи которого в ячейку была исключена возможность записи всех других чисел
Возможное значение	Число, которое может быть записано в ячейку
Область пересечения (или пересечение областей)	Сочетание трех областей (прямых X, Y и квадрата) в пределах которых ячейка должна иметь уникальное значение
Теоретическая линия	Линия, составленная из ячеек с возможными значениями, находящимися в одном квадрате и лежащими на одной прямой
Пара пустых ячеек (с возможными значениями)	Некоторое количество ячеек (не обязательно 2), имеющих одни и те же возможные значения и исключающих тем самым друг друга из списка проверки
Порядковый номер возможного значения	Число, обозначающее позицию используемого возможного значения в общем списке всех возможных значений всех ячеек головоломки в текущий момент

Содержание

Список рисунков

1. Вступление

Основной задачей данной работы являлось создание автономного и кроссплатформенного приложения решателя судоку. Была поставлена именно такая задача, так как на момент написания данной работы не было найдено ни одного известного автономного кроссплатформенного решателя судоку. Для решения поставленной задачи был выбран редактор кода Qt Creator. Данный редактор является кроссплатформенной свободной интегрированной средой разработки для таких языков программирования, как C, C++ и QML. Также данный редактор включает в себя графический интерфейс отладчика и визуальные средства разработки интерфейса, что является большой помощью для тех, кто не имел раньше дела с разработкой приложений. Именно по причине всего вышеперечисленного данный редактор кода был выбран для реализации представленной задачи.

Для построения алгоритма решения головоломки судоку было найдено несколько методик решения. Впоследствии они были представлены в виде отдельных функций в коде программы. Такое представление оказалось очень удобным в случае необходимости настройки программы и исправлении возникающих в ходе написании кода ошибок. Сама программа была написана на языке C++, так как данный язык программирования несколько раз был предметом изучения на специальности, на которой автор проходит обучение. Таким образом, основы данного языка оказались уже известны, что существенно облегчило понимание структур Qt Creator’а и написание программы. Кроме того, C++ является одним из наиболее популярных языков программирования, что означает, что в случае разбора кода приложения, написанного на данном языке, большему количеству людей он будет понятен.

Сама же данная работа (отчет) была написана на русском языке, так как автор владеет в совершенстве только им и хотел в точности передать и описать свои мысли и идеи относительно созданного им приложения. Кроме того, автор не имел до этого опыта написания подобных работ и хотел уменьшить возможное количество ошибок, поэтому использовал родной язык.

2. Решение головоломки судоку

В данной главе описаны проблемы, возникающие при решении головоломки судоку, и способы их разрешения. Способы решения представлена как в виде изображений с описанием под ними, так и в виде отрезков кода с подробным пояснением к каждому. В итоге, сумма всех описанных далее методик и функций и представляет основу решателя “Sudoku Solver”.

2.1. Описание головоломки судоку

В данном пункте будет описана терминология, применяемая для описания решения поставленных задач и проблем, возникающих в ходе работы, а также применяемая в коде самой программы.

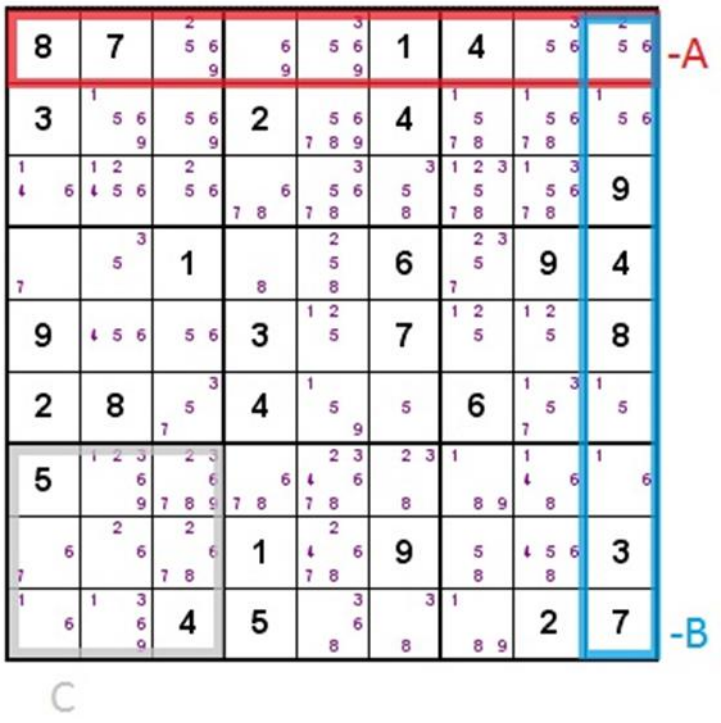

Рисунок 1. Области уникальности чисел.

Стандартная головоломка судоку состоит из поля клеток 9 на 9, разделенного в свою очередь на 9 равных частей (квадратов). Каждая из 9 частей имеет в себе поле 3 на 3, ячейки которого, в конечном счете, после удачного решения должны будут иметь числа от 1 до 9. При этом за удачное решение является наличие в

каждом квадрате (C), горизонтальной прямой Y (A) и вертикальной прямой X (B) только одной ячейки с одним уникальным значением (каждое число должно встречаться только однажды).

Изначально в такой головоломке содержится от 16 начальных значений (далее любое значение, существование которого в конкретной ячейке было доказано, будем называть подтвержденным или действительным). Исходя из этих самых значений, начинает формироваться решение, которое возникает за счёт применения элементарных законов логики при поиске ответов для пустых ячеек. Если в некоторых ячейках в какой-то момент нет однозначного ответа, какое число должно быть туда записано, тогда можно представить все возможные на данный момент значения в данной ячейке, расписав их по периметру ячейки в виде небольших отметок (пример на рисунке 1). Далее в данной работе подобные значения будем называть возможными значениями.

Количество минимальных изначальных значений было установлено в 16, так как меньшее количество значений приведет к тому, что решение не будет уникальным, а головоломка в свою очередь может оказаться нерешаемой. Такое решение было принято исходя из расчетов, реализованных ирландскими математиками на суперкомпьютере в Ирландском центре современнейших вычислений в Дублине (ICHEC) и опубликованных в сети 1 января 2012 года[1], согласно которым минимальное количество изначально расставленных числе в головоломке для получения уникального решения должно равняться 17. Однако для полной демонстрации работы созданной программы в данном проекте были одобрены судоку с количеством изначально заполненных ячеек от 16 штук.

2.2. Логика реализации основной задачи

Для решения основной задачи было необходимо разработать общую стратегию для решения головоломки судоку. Впоследствии данный алгоритм необходимо было представить в виде кода, воссозданного при использовании избранной IDE. Для реализации основной задачи решателя судоку были разработаны несколько

[1] http://arxiv.org/abs/1201.0749

функций, отдельно реализующих каждый возможный алгоритм решения задачи и расположенных в порядке возрастающей сложности и затрачиваемом времени на свое полное выполнение. Для начала перечислим основные пункты и поясним основную логику, реализованную в каждой из этих функций.

2.2.1. Простейшая логика (функция firstLVL())

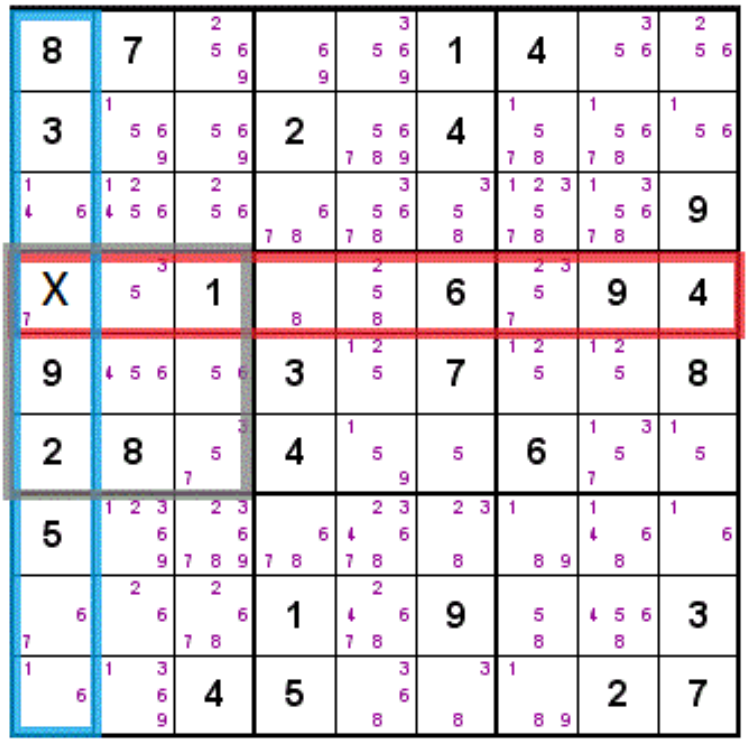

Рисунок 2. Пересечение областей(выделенные области).

В качестве простейшей логики возьмем логику для нахождения значения ячейки при наличии в пересекаемых ячейкой X (рисунок 2) областях всех возможных значений, кроме одного. В таком случае единственно не задействованное значение и является ответом для данной ячейки X. В нашем случае этим значением является цифра 7. Все остальные цифры присутствуют на горизонтальной и вертикальной прямых и в квадрате, к которому принадлежит ячейка. Далее будем называть сочетание областей, в пределах которых рассматриваемая ячейка должна иметь уникальное значение областью пересечения (или пересечением областей) рассматриваемой ячейкой.

Такая логика является основной для нахождения значений ячеек на протяжении всего решения судоку. Также данная идея реализована в кодовом виде как первая

функция решателя firstLVL() и впоследствии постоянно происходит возврат к ней, как к основе решателя.

```
if(Number[x][y]==0)
    NCounter=0;
    for(NoS=1;NoS<=N;NoS++)
        counterZ=1;
        if(x<9)
            counterX=3;
        if(x<6)
            counterX=2;
        if(x<3)
            counterX=1;
        if(y<9)
            counterY=3;
        if(y<6)
            counterY=2;
        if(y<3)
            counterY=1;
        for(X=counterX*3-3;X<counterX*3;X++)
            for(Y=counterY*3-3;Y<counterY*3;Y++)
                 if(Number[X][Y]==NoS)
                     counterZ=0;
        for(a=0;a<N;a++)
            if(Number[x][a]==NoS)
                 counterZ=0;
            if(Number[a][y]==NoS)
                 counterZ=0;
        if(counterZ==1)
            Numb=NoS;
            NCounter++;
     if(NCounter<2 and status!=1)
        Done=1;
        Number[x][y]=Numb;
```

Рисунок 3. Выполнение основной задачи первой функции

Для решения задачи в ходе работы данной функции происходит перебор ячеек, начиная с ячейки Number[0][0] и заканчивая ячейкой Number[8][8]. Для подобного перебора используется 2 цикла по X и по Y, расположенные один в другом и представляющие собой перемещение по ячейкам слева-направо и сверху-вниз. Дойдя до последней ячейки в первой горизонтальной строке, начав перемещение слева-направо, начиная с ячейки Number[0][0] (впоследствии

перемещение по Y), и проверив её, происходит перемещение на следующую строку (впоследствии перемещение по X) и проверка возобновляется, начиная с ячейки Number[1][0]. Перебор будет завершен по достижении ячейки Number[8][8] и результат работы функции будет возвращен.

В ходе перебора происходит проверка на заполнение ячейки некоторым значением. В случае если ячейка оказывается пустой, происходит переход к рассмотрению всех возможных значений (переменная NoS). Осуществляется их полный перебор, начиная с 1 и до 9.

Сразу определяется квадрат расположения данной ячейки на поле судоку. Происходит это посредством выполнения или невыполнения некоторых условий для значений X и Y (координат рассматриваемой ячейки). После уточнения квадрата определяем наличие или отсутствие в зоне пересечения данной ячейкой (Рис. 2) рассматриваемого в данный момент возможного значения NoS. При совпадении с одним из уже имеющихся значений в области пересечения происходит переход к следующему возможному значению, данное же отбрасывается, как невозможное. Если же данное возможное значение ни разу не было найдено, то оно временно сохраняется в переменной Numb, которое будет записано в рассматриваемую ячейку впоследствии после просмотра всех возможных значений, если при этом другие возможные значения NoS были отброшены. В случае если было подтверждено 2 или более возможных значений, запись не производится.

Однако помимо выше описанной задачи данная функция решает ещё несколько проблем, не столь очевидных на данном этапе, но возникающих впоследствии. В список возможностей данной функции помимо её основной функциональности входят:

1) Вычисление всех возможных значений для каждой пустой ячейки головоломки и их вывод

2) Вычисление общего количества возможных значений и их выборочная проверка (посредством добавления каждого в качестве правильного варианта в ответ и последующего решения)

3) Обнаружение вариантов головоломок, не имеющих решения

Вычисление всех возможных значений и их вывод

Данные вычисления и вывод реализуется по ходу работы функции и представляет собой реализацию задачи, поставленной для увеличения возможностей приложения. Таким образом, имеется возможность автоматически вычислить все возможные значения и отобразить их на интерфейсе приложения.

```
.
for(a=0;a<10;a++)
   NVariants[a]=0;
SVariants="";
MyPrintingV(SVariants,x,y);
.
.
if(counterZ==1)
   NVariants[NCounter]=NoS;
   NCounter++;
.
.
if(status==1)
   while(NCounter>0)
      NCounter--;
      if(NVariants[NCounter]!=0)
         SVariants=SVariants +
QString::number(NVariants[NCounter]);
    MyPrintingV(SVariants,x,y);
.
```

Рисунок 4. Выполнение задачи по выводу возможных значений

В ходе работы функции firstLVL() первый участок кода, отображенные на рисунке 4, воспроизводятся для каждой ячейки. Все остальные участки реализуются только для пустых ячеек.

В случае если пользователь захотел узнать возможные значения ячеек, он должен был отметить данное пожелание посредством выбора одного из вариантов поиска на пользовательском интерфейсе данной программы. При наличии этой отметки, после начала вычислений на данную функцию при её запуске будет передан

сигнал о том, что необходимо вычислить именно все возможные значения. Данный сигнал будет представлен в качестве переменной со значением 1 (status=1). Если же никакого сигнала получено не было, основная функция продолжит свою работу в стандартном режиме, описанном прежде.

Изначально для каждой ячейки данной функции предусмотрено удаление старой информацией, оставленной прежде в ячейках судоку. Также происходит сброс переменных для более корректной работы программы.

Далее описан процесс последовательной записи в массив возможных значений в рассматриваемой ячейке. Данная запись производится только в том случае, если области пересечения не было обнаружено рассматриваемого в данный момент возможного значения.

В итоге, после проверки всех возможных значений для рассматриваемой ячейки и перед переходом к следующей, в случае, если переменная status равна 1, в цикле производится запись из массива в строку по убывающей (с наибольшего до наименьшего) всех ранее записанных значений. Записанная же строка переписывается в только что рассмотренную ячейку.

Вычисление количества возможных значений и их проверка

Данные вычисление и проверка производятся только в случае, если ни одна из основных рабочих функций не может вычислить более ни одного действительного значения для рассматриваемой головоломки, а количество пустых ячеек все ещё не равно 0. Вследствие этого происходит переход на перебор оставшихся возможных значений и поиск ответа с их помощью. Однако данный вопрос будет рассмотрен позже в пункте 2.3., а сейчас вернемся к реализации, расположенной в основной функции firstLVL().

```
.
if(fail>1)
    Op=fail-1;
.
.
if(counterZ==1)
    NumbOfOp++;
    if(Op==NumbOfOp)
        Number[x][y]=Numb;
        return (x+1)*N+y+1;
.
.
if(fail==1)
    return NumbOfOp+1;
.
```

Рисунок 5. Выполнение задачи по вычислению суммы возможных значений и их запись

При запуске основной функции производится проверка того, что необходимо найти. В случае если была поставлена задача найти количество возможных значений, при запуске функции переменной fail будет присвоено значение 1. Если необходимо произвести проверку-запись нового значения, переменная fail должна быть больше 1. Сам же порядковый номер значения, которое необходимое записать как подтвержденное, выражается в переменной Op=fail-1. Если никакой из перечисленных вариантов производить не надо было, значение переменной fail будет равно 0 и работа функции продолжится в стандартном режиме.

Далее производится одновременный подсчет количества найденных возможных значений (переменная NumbOfOp) и сравнение текущего количества с искомым порядковым номером, применяемым для записи возможного значения в качестве подтвержденного. Если порядковый номер Op был ранее записан, тогда при его достижении возможное значение будет записано в ячейку Number[x][y] и будет произведен возврат функцией firstLVL() координат, куда была произведена запись. Работа функции будет завершена.

Однако если было необходимо найти количество возможных значений, будет произведен возврат переменной NumbOfOp, увеличенной на 1. Данное действие необходимо для правильной интерпретации получаемого результата работы функции firstLVL() в вызывающих функциях.

Обнаружение вариантов, не имеющих решения

Для обнаружения вариантов, не имеющих решение, используется следующая логика. Каждая ячейка при правильно составленной головоломке должна иметь как минимум одно возможное значение. Однако если возникает ситуация, когда не одно значение не может быть подставлено в пустую ячейку, а в области пересечения рассматриваемой ячейки присутствуют все возможные значения, ячейка должна остаться пустой. Обнаруживается, что решения для данной головоломки нет. Чтобы реализовать отсутствие решения без привлечения каких-либо излишних средств, при отсутствии какого-либо возможного значения (NCounter=0) для текущей ячейки происходит запись номера, установленного по умолчанию (Numb=1) или же любого другого, присвоенного данной переменной в ходе работы программы. В ходе нескольких итераций функции firstLVL() получаем полностью заполненное поле судоку. В ходе же итоговой проверки выясняется, что поле заполнено неверно, что означает, что решения для данного судоку нет.

2.2.2. Простейшая логика для пересечения областей (функция secondLVL())

Следующая функция secondLVL() представляет собой усовершенствование основной работы предыдущей рассмотренной функции. На подобии функции firstLVL() используется проверка на наличие в области пересечения пустой ячейкой возможных значений, однако теперь делается упор не на общее количество найденных возможных значений в каждой пустой ячейке, а на количество ячеек, имеющих проверяемое возможное значение на рассматриваемых прямых X или Y или в квадрате.

```
a=1;
  while(a<4)
    b=1;
    while(b<4)
      NCounter=0;
      for(counterX=a*3-3;counterX<a*3;counterX++)
        for(counterY=b*3-3;counterY<b*3;counterY++)
           if(Number[counterX][counterY]==NoS)
              NCounter=9;
              counterX=N;
              counterY=N;
           if(Number[counterX][counterY]==0)
              for(square=0;square<9;square++)
                 if(Number[counterX][square]==NoS)
                    square=10;
                 if(Number[square][counterY]==NoS)
                    square=10;
              if(square==9)
                 X=counterX;
                 Y=counterY;
                 NCounter++;
      if(NCounter==1)
          Done=2;
          Number[X][Y]=NoS;
      b++;
    a++;
```

Рисунок 6. Выполнение основной задачи второй функции для квадрата

Для разбора можно взять одну из трех частей второй функции, так как все части работают по одному сценарию, за исключением применения некоторых специфичных конструкций для конкретных областей. Поэтому для разбора возьмем первую и одновременно наиболее компактную часть функции.

Применение такой конструкции происходит для каждого возможного значения NoS, которые перебираются в цикле с 1 до 9. При этом внутри цикла перебора значений происходит перебор квадратов, начиная с левого верхнего и заканчивая правым нижним. Переменная a отвечает за порядковый номер квадрата, начиная с первого верхнего и считая сверху вниз, в свою очередь переменная b - за порядковый номер квадрата, начиная с первого левого и считая слева направо. Выбрав квадрат для рассмотрения, программа высчитывает координаты ячеек, используемых в данном квадрате.

Далее для каждой ячейки проверяется, во-первых, имеется ли в ней значение, равное проверяемому в данный момент возможному значению, а во-вторых, является ли данная ячейка пустой. В случае если ячейка имеет проверяемое значение, то происходит переход к следующему квадрату для проверки по тем же пунктам. Если имеется значение, отличное от проверяемого, но в то же время не равное 0, происходит переход к следующей ячейке квадрата. Переход между ячейками квадрата происходит в том же порядке, что и переход между квадратами судоку. В итоге, если программа находит пустую ячейку, то к ней применяется проверка на наличие на пересеченных ею прямых X и Y проверяемых возможных значений. Если на данных прямых не найдено ни одного рассматриваемого возможного значения, тогда координаты прошедшей проверку ячейки сохраняются, а количество прошедших проверку ячеек в данном квадрате по рассматриваемому возможному значению увеличивается на 1.

Произведя проверку для всех ячеек в текущем квадрате, программа проверяет полученное количество прошедших проверку. Если прошли проверку более одной ячейки, результаты и записанные данные игнорируются. Происходит переход к следующему, не рассмотренному квадрату. В случае если проверку прошла одна ячейка, возможное значение становится действительным для полученной с помощью сохраненных координат (X и Y) ячейки (Number[X][Y]=NoS).

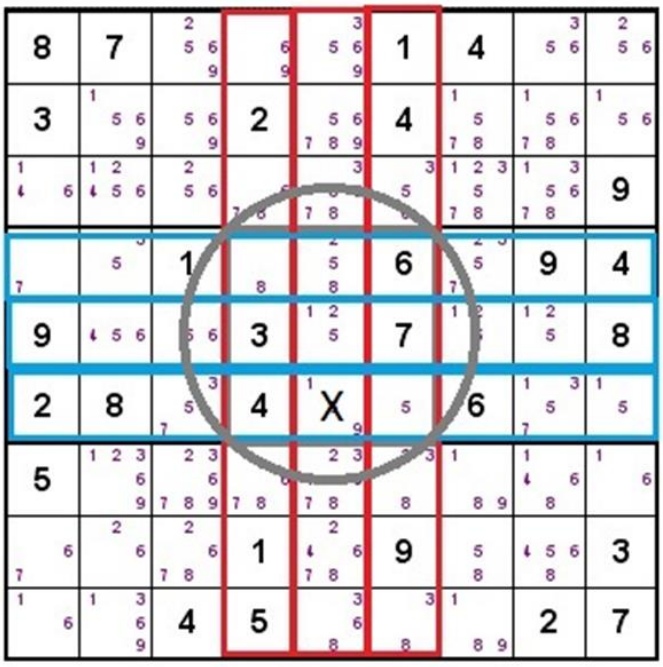

Рисунок 7. Пересечение областей для всех клеток квадрата.

На рисунке 7 можно увидеть живой пример работы данной функции. Проведя проверку по возможному значению 9 в выделенном квадрате, можно заметить, что в области пересечения всех пустых ячеек, кроме той, что отмечена X, на пересекаемых прямых находятся ячейки со значением 9. Таким образом, возможность поставить 9 в рассматриваемом квадрате присутствует только для ячейки, отмеченной X, что и следует сделать.

Подобным же образом работает проверка для прямых X и Y. Разница состоит лишь в том, что проверка производится не для каждого квадрата, а для каждой прямой. При этом передвигаясь и проверяя каждое значение на прямой Y (Y меняется от 0 до 8), производится поиск на совпадение с проверяемым возможным значением на прямой X (для каждого значения Y значение X меняется от 0 до 8) и в квадрате (координаты ячеек, расположенных в квадрате формируются каждый раз для нового X), в котором в данный момент находится проверяемая пустая ячейка прямой X. Те же самые действия производится при проверке по прямой X. Только в отличие от описанной выше модели переменные X и Y меняются местами и ролями.

2.2.3. Логика использования теоретических линий (функция thirdLVL())

Теперь разберем функцию thirdLVL(), работающую в соответствии со следующей логикой. Для примера разберем ситуацию на рисунке 8.

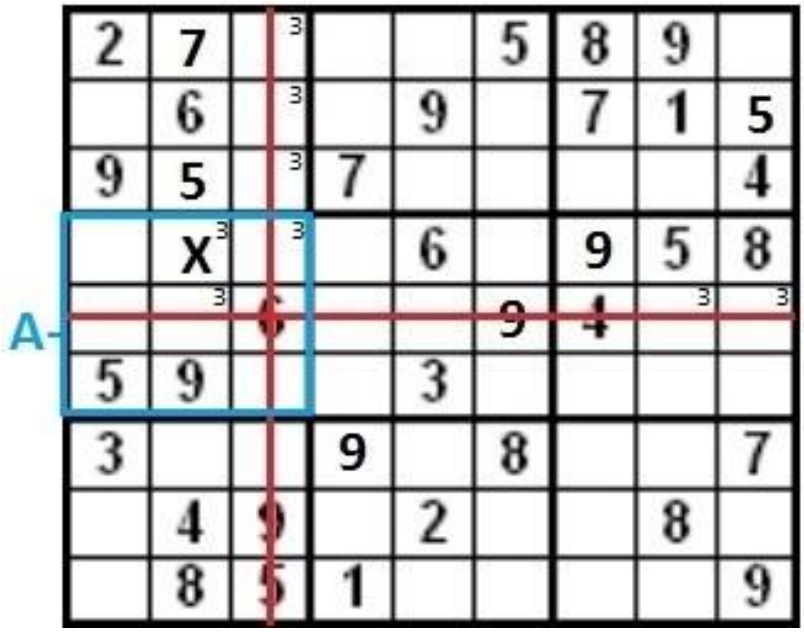

Рисунок 8. Использование теоретических линий.

Согласно правилам, применяемым при решении головоломки судоку, запрещается ставить в пустые ячейки те значения, что уже имеются в области пересечения пустой ячейки. Таким образом, легко вычислить, где могут стоять те или иные значения для всего поля. Однако данное вычисление не всегда гарантирует окончательную подстановку хотя бы одного значения. В такой ситуации необходимо прибегнуть к использованию ещё не утвержденных значений в качестве действительных. Даже не присутствуя на поле в своем утвержденном виде, при удачной расстановке возможные значения могут помочь продолжить решение и оказать подобным образом существенную поддержку.

В качестве рассматриваемого теоретического значения возьмем 3. Как видно на рисунке 8, на всем поле судоку имеется всего две подтвержденных ячейки со значением 3. В каждом квадрате, кроме квадратов с уже имеющимися ячейками со значением 3, имеется, по меньшей мере, 2 ячейки, куда может быть подставлено рассматриваемое возможное значение. Однако если все ячейки одного квадрата с возможным значением 3 лежат на одной линии, то с уверенностью можно утверждать, что на оставшихся шести ячейках получившейся линии из 9 ячеек, исключая ячейки квадрата, откуда была образована линия, рассматриваемое возможное значение не может существовать. Данное утверждение следует из того, что одна из пустых ячеек, содержащих возможное значение 3, наверняка, в конечном счете, должна будет получить это возможное значение в качестве действительного. Таким образом, можно представить использование сочетания возможных значений, образующих вместе подобие линии (далее будем называть теоретической линией).

Образованные теоретические линии помогают убрать возможные значения из ячеек других квадратов. Таким образом, рассматривая ситуацию с квадратом А можно отметить, что помимо изначально удаленных возможных значений 3 ячейками с действительными значениями 3, были также исключены ещё 2 возможных значения с помощью образованных теоретических линий. После этого осталась единственная не исключенная ячейка X с рассматриваемым теоретическим значением 3. В эту ячейку и следует записать цифру 3.

```
x=0;
while(x<N)
  y=0;
  while(y<N)
    if(Number[x][y]==NoS)
        xLine[x]=NoS;
        yLine[y]=NoS;
    y++;
  x++;
```

Рисунок 9. Предустановка теоретических линий для действительных значений.

Далее рассмотрим реализацию данного метода в программе. Представленные далее действия производятся, как и в случае с предыдущей функцией, при переборе всех возможных значений (NoS меняется от 1 до 9). Участок кода, изображенный на рисунке 9, реализует создание теоретических линий из изначально подтвержденных чисел. Из каждого действительного числа, которое равняется рассматриваемому в данный момент значению, создается 2 линии, перпендикулярные по отношению друг к другу и пересекающие друг друга в ячейке действительного числа, из которого были созданы. Одна из линий представлена переменной xLine[x], обозначающая принадлежность к прямой, на которой все значения X равны текущей координате x. Другая линия представлена переменной yLine[y], обозначающая принадлежность к прямой, на которой все значения Y равны текущей координате y.

```
for(counterX=a*3-3;counterX<a*3;counterX++)
      for(counterY=b*3-3;counterY<b*3;counterY++)
          if(Number[counterX][counterY]==NoS)
              ...
          if(Number[counterX][counterY]==0)
              ...
              if(square==9)
                  xSearch[i]=counterX;
                  ySearch[i]=counterY;
                  i++;
```

Рисунок 10. Поиск возможных значений ячеек квадрата.

После того, как все ячейки будут проверены на совпадение с рассматриваемым возможным значением, начнется поиск пустых ячеек с возможным значением в каждом квадрате поля. Как и в случае с предыдущей функцией secondLVL(), будет таким же образом происходить переход между квадратами. Перемещение

между ячейками также будет взято из предыдущей функции, как и проверка для каждой пустой ячейки области пересечения и наличия в ней текущего возможного значения. Отличие состоит в том, что координаты каждой прошедшей проверку ячейки будут записаны в отдельную переменную массива для последующего сравнения между собой.

```
X=xSearch[0];
Y=ySearch[0];
if(i==3 and NCounter==0)
  if(xSearch[0]==xSearch[1] and
xSearch[2]==xSearch[1])
      xLine[X]=NoS;
      xCube[a*3+b-4]=X;
  if(ySearch[0]==ySearch[1] and
ySearch[2]==ySearch[1])
      yLine[Y]=NoS;
      yCube[a*3+b-4]=Y;
if(i==2 and NCounter==0)
  if(xSearch[0]==xSearch[1])
      xLine[X]=NoS;
      xCube[a*3+b-4]=X;
  if(ySearch[0]==ySearch[1])
      yLine[Y]=NoS;
      yCube[a*3+b-4]=Y;
```

Рисунок 11. Составление из точек теоретических линий.

В итоге, когда в рассматриваемом квадрате все ячейки проходят проверку, программа переходит к составлению из полученных точек теоретических линий. Данные линии образуются только в том случае, если все ячейки квадрата, в которых было найдено рассматриваемое возможное значение, лежат на одной линии. Проверяется это путем сравнения сохраненных координат между собой. Если все они имеют одно и то же значение координаты X, тогда данные ячейки образуются теоретическую линию xLine[x]. Если же имеется совпадение по координате Y, образуется теоретическая линия yLine[y]. Реализуется только один из двух вариантов, так как ячейки должны лежать только на одной прямой для правильного применения данного правила. Помимо теоретических линий происходит запись переменной, обозначающей квадрат, откуда была образована созданная теоретическая линия, вместе со значением, обозначающим саму линию (xCube[]=X или yCube[]=Y). Данное действие необходимо для временного

исключения только что созданной линии при повторном рассмотрении только что проверенного квадрата.

Если имеется более 3 ячеек или если полученные ячейки не лежат на одной прямой, теоретические линии не образуются. Также на данном этапе не проводилась проверка на наличие одной ячейки в квадрате с проверяемым возможным значением, так как данная возможность уже проверялась в предыдущей функции, а никаких предпосылок для изменения ситуации пока не произошло.

Произведенные действия применяются к каждому квадрату поля судоку. При достижении последнего квадрата и его проверке происходит повторный поиск теоретических линий, однако на этот раз вместе с поиском возможности записать действительное число. Также проверка теперь происходит не с помощью действительных чисел, а с помощью всех созданных до этого теоретических линий, что представлено на рисунке 12.

```
if(counterX==xCube[a*3+b-4])
    xLine[counterX]=0;
if(counterY==yCube[a*3+b-4])
    yLine[counterY]=0;
if(xLine[counterX]!=NoS and yLine[counterY]!=NoS)
    xSearch[i]=counterX;
    ySearch[i]=counterY;
    i++;
if(counterX==xCube[a*3+b-4])
    xLine[counterX]=NoS;
if(counterY==yCube[a*3+b-4])
    yLine[counterY]=NoS;
```

Рисунок 12. Проверка с помощью теоретических линий.

Представленная конструкция реализуется для каждой найденной пустой ячейки в квадрате. Как видно на рисунке 12, перед проверкой каждой ячейки проводится проверка на наличие записи о создании в текущем квадрате для текущих координат теоретической линии. Если данная линия действительно была создана именно ячейками этого квадрата и существует, тогда представленная теоретическая линия временно удаляется. После проверки ячейки на пересечение с теоретическими линиями, которые могут исключить её из претендентов на

рассматриваемое значение, все удаленные линии будут восстановлены. Такая конструкция необходима для того, чтобы избегать ситуации, когда теоретическая линия, созданная ячейками некоторого квадрата, исключает возможность появления в этих самых ячейках рассматриваемого возможного значения. Таким образом, в итоге можно получить ситуацию, что ни в одной ячейке квадрата не может быть рассматриваемого возможного значения, хотя в самом квадрате возможное значение в качестве действительного пока не присутствует.

Когда же все ячейки квадрата будут проверены, программа начинает поиск сочетания, при котором можно будет создать новые теоретические лини. Также теперь появляется возможность при получении только одной ячейки с возможным значением в квадрате записать данное значение в ячейку в качестве действительного.

Повтор цикла поиска теоретических линий и действительных чисел производится до тех пор, пока программа не перестанет получать новые линии или числа.

2.2.4. Логика применения пустых ячеек с одним возможным значением (функция fourthLVL())

Функция fourthLVL() в начале своей работы имеет очень много общего в сравнении с предыдущей рассмотренной функцией. В её работе также создаются и применяются теоретические линии, однако они необходимы лишь для того, чтобы сократить общее число ячеек с проверяемым возможным значением на поле судоку. Для пояснения разницы между функциями рассмотрим пример, проиллюстрированный на рисунке 13.

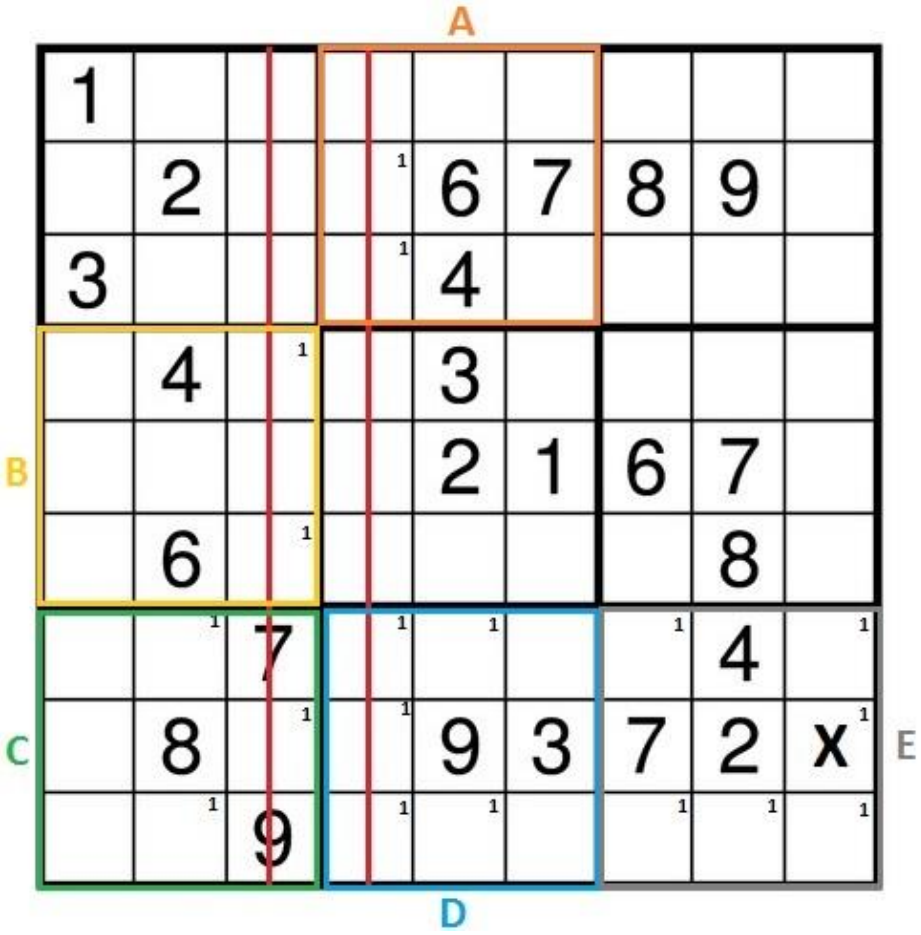

Рисунок 13. Пример использования пустых ячеек с одним возможным значением.

Исходя из ранее описанных правил в пункте 2.2.3., в первую очередь составим теоретические линии. Производить проверку будем по возможному значению 1(NoS=1). Используя лишь действительные числа, не сложно составить из образовавшихся на одной линии ячеек в квадрате A и на другой линии ячеек в квадрате B две теоретические линии. Благодаря этим самым линиям есть также возможность построить теоретические линии, исходящие из квадратов C и D, однако делать этого не станем, так как, в конечном счете, нам ничего это не даст, хотя в программе данные действия производятся для просмотра всех возможных вариантов.

Теперь рассмотрим подробнее квадрат E. В данном квадрате рассматриваемое возможное значение расположено в каждой пустой ячейке, при этом ни одной теоретической линии провести в сторону данного квадрата нельзя, из чего следует, что предыдущая функция тут не поможет.

Главное отличие данной функции от предыдущей состоит в использовании не квадрата, где при наличии только одной ячейки с рассматриваемым возможным значением в данную ячейку число вписывается в качестве действительного, а прямых, на которых также в ситуации, когда ячейка с возможным значением встречается только однажды, возможное значение ячейки становится действительным. Таким образом, ещё раз обратив внимание на квадрат E, можно с уверенностью сказать, что на прямой, проходящей через квадраты C, D и E и проходящей через ячейку X возможное значение 1 встречается только один раз в самой ячейке X. Туда его и следует записать.

Теперь посмотрим содержание самой программы и разберем, как в ней описанная выше структура реализована. Для начала обратим внимание на рисунок 14.

```
if(xLine[counterX]!=NoS and yLine[counterY]!=NoS)
   SNumbers[counterX][counterY]=NoS;
   xSearch[i]=counterX;
   ySearch[i]=counterY;
   i++;
if(xLine[counterX]==NoS or yLine[counterY]==NoS)
   SNumbers[counterX][counterY]=0;
```

Рисунок 14. Сохранение возможных значений.

На рисунке представлена часть, в которой реализуется сохранение новых полученных данных о содержании ячейкой возможного значения. При этом в следующем условии представлено стирание данных о том, что ячейка содержит рассматриваемое возможное значение. Подобное удаление данных происходит в случае, если были образованы новые теоретические линии, которые исходят из другого квадрата и при этом исключает возникновение в проверяемой ячейке возможного значения. Данное сокращение числа ячеек с возможным значением уменьшает количество возможных ячеек в линиях и тем самым упрощает решение головоломки. Сама описанная конструкция является усовершенствованной частью конструкции, изображенной на рисунке 12, и также применяется при поиске возможных теоретических линий.

После нахождения всех возможных линий и в то же время нахождения минимального числа ячеек с возможным значением происходит проверка на уникальность ячейки на прямой. Данная проверка проходит в два этапа. Сначала

проверяются значения на прямых X для всех пустых ячеек всех квадратов судоку. После та же самая проверка проводится на прямых Y.

```
if(Number[counterX][counterY]==NoS)
    NCounter=9;
if(SNumbers[counterX][counterY]==NoS)
    for(square=0;square<9;square++)
        if(square<a*3-3 or square>a*3-1)
            if(SNumbers[square][counterY]==NoS)
                square=10;
    if(square==9)
        X=counterX;
        Y=counterY;
        NCounter++;
```

Рисунок 15. Определение уникальных ячеек.

Сам поиск подходящих ячеек в квадрате изображен на рисунке 15. Стандартно проверяется, присутствует ли уже в квадрате ячейки с проверяемым возможным значением. Далее происходит проверка, была ли записана данная ячейка в массив ячеек с рассматриваемым возможным значением (SNumbers[][]). При произошедшей ранее записи происходит проверка по одной из прямых (X или Y). При этом исключается из проверки квадрат, в котором находится ячейка с возможным значением во избежание того, чтобы не посчитать её во второй раз. В случае если на рассматриваемой прямой не было больше обнаружено ни одной ячейки с тем же возможным значением, координаты проверенной ячейки сохраняется для возможной последующей записи. В итоге, если при поиске пустых ячеек с возможным значением в рассматриваемом квадрате была найдена только одна, производится окончательная запись в данную уникальную ячейку подтвержденного числа.

2.2.5. Логика попарного использования пустых ячеек с несколькими возможными значениями (функция fifthLVL())

Функция fifthLVL() является завершающей функцией и применяется в случае, если ни одна из предыдущих четырех рассмотренных функций не дала результата. Чтобы понять работу функции fifthLVL(), для начала разберем пример на рисунке 16.

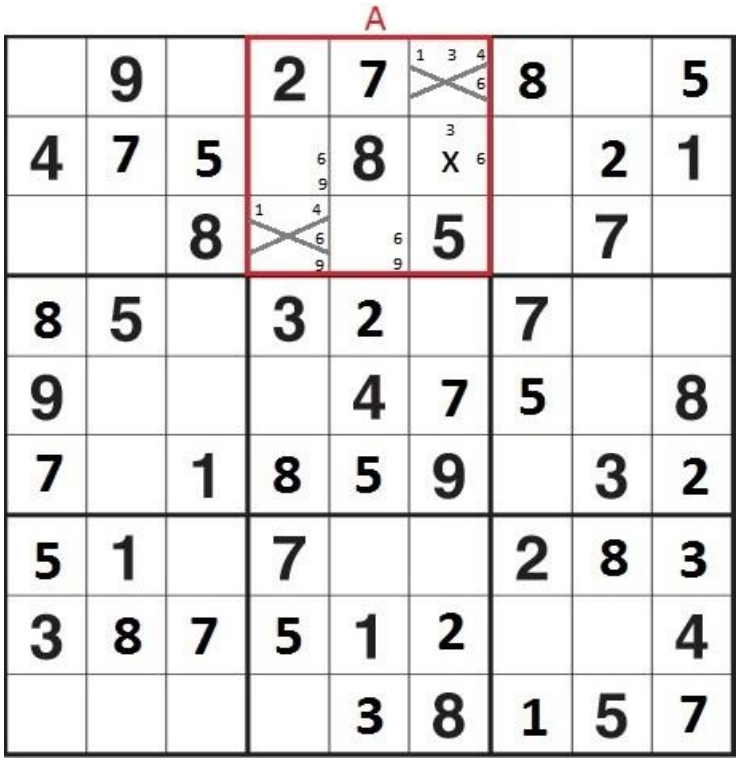

Рисунок 16. Описание использования функции fifthLVL().

Для начала найдем все возможные значения для квадрата A. Сделав это, можно увидеть, что возможных значений довольно много, каждое встречается не единожды, в каждой ячейке присутствует по несколько возможных значений. Создание теоретических линий ничего не даст, как и поиск уникальных ячеек с возможным значением по каждой линии. Таким образом, предыдущие рассмотренные правила в данном случае не применимы.

Однако обратим внимание на следующую особенность. Возможное значение 1, как и возможное значение 4, встречается дважды. При этом оба возможных значения находятся в одних и тех же ячейках. Предположим, что одна из ячеек получит действительное значение 1, в таком случае другая ячейка автоматически получит действительное значение 4, так как останется только одна ячейка с данным возможным значением. То же самое получится в противоположном случае. Если первая ячейка получит действительное значение 4, тогда вторая получит оставшееся возможное значение 1 в качестве действительного. Данная закономерность применима только в том случае, если количество рассматриваемых возможных ячеек совпадает с общим количеством ячеек, в которых данные возможные значения присутствуют. В нашем случае это две ячейки и два возможных значения. Под рассмотрение в теории может попасть до

8 ячеек с 8 возможными значениями, однако чем больше количество одновременно попадающих под рассмотрения ячеек с возможными значениями, тем меньше вероятность на удачный исход.

Однако вернемся к рассмотрению примера. Поскольку рассмотренные 2 ячейки с возможными двумя значениями являются парными (при этом исключается любые другие возможные значения в данных ячейках, кроме найденных), тогда данные ячейки можно вычеркнуть из списка пустых на время проверки рассматриваемого квадрата. Удаление использованных возможных значений также дозволено. После этого можно продолжить рассмотрение оставшихся в квадрате пустых ячеек.

Не трудно заметить, что только в ячейке, отмеченной X, осталось возможное значение 3. Без промедления устанавливаем его в качестве подтвержденного. Подобным образом и работает попарное использование пустых ячеек с несколькими возможными значениями. Далее перейдем к рассмотрению того, как описанная выше структура реализована в программе.

```
for(NoS=1;NoS<=N;NoS++)
    NCounter=1;
    for(X=x*3-3;X<x*3;X++)
        for(Y=y*3-3;Y<y*3;Y++)
            if(Number[X][Y]==NoS)
                NCounter=0;
    for(square=0;square<N;square++)
        if(Number[counterX][square]==NoS)
            NCounter=0;
        if(Number[square][counterY]==NoS)
            NCounter=0;
    if(NCounter==1)
        SNumbers[i][NoS-1]=NoS;
i++;
```

Рисунок 17. Запись возможных значений пустой ячейки.

Изображенная на рисунке 17 структура производится для каждой найденной пустой ячейки в рассматриваемом квадрате. Обнаружив пустую ячейку, программа производит проверку по всем возможным значениям. Если некоторое возможное значение подходит для рассматриваемой ячейки, тогда данное значение вписывается в массив возможных значений (SNumbers[i][NoS-1]=NoS). Переменная i в данном массиве представляет порядковый номер пустой ячейки в

массиве пустых ячеек. Сама конструкция SNumbers[i][NoS-1]=NoS является удобной, так как с её помощью без проблем можно определить, было ли записано то или иное возможное значение в любой пустой ячейке в любое время, не прибегая при этом к помощи многих других переменных и их запоминанию.

```
b=2;
while(b<i)
    for(a=0;a<b;a++)
        ZNumbers[a]=a+1;
    finish=0;
    while(finish!=1)
```

Рисунок 18. Предустановка проверяемых возможных значений.

Далее для сравнения ячеек с возможными значениями между собой создается образец (ZNumbers[a]). Данный образец создается независимо от содержавшихся возможных значений какой-либо пустой ячейки. Проверяются все возможные комбинации чисел. При этом сначала проверяются все комбинации для возможных значений из двух чисел. После того, как все комбинации из двух чисел были проверены, начинается проверка комбинаций из трех чисел. Далее возможно из четырех, пяти и так до комбинации из i-1 чисел, где i – это количество найденных пустых ячеек в текущем квадрате. После создания первоначального образца из b чисел (при первом подходе b=2) происходит переход в цикл, который повторяется с изменением образца до тех пор, пока не будут проверены все возможные комбинации из b чисел.

В самом цикле изначально происходит проверка на совпадение для каждой пустой ячейки по её возможным значениям с текущим образцом (рисунок 19). Если было найдено более 1 совпадения, ячейка попадает в список парных ячеек. Если не было ни одного совпадения, ячейка отбрасывается, как непарная. При обнаружении ячейки с одним совпадением с образцом дальнейшая проверка отменяется для данного образца, так как одно непарное возможное значение автоматически исключает вероятность составить действующую пару из возможных значений. (В такой паре не может произойти взаимоисключения, что означает, что при условии использовании предыдущих функций такая пара не может существовать.) Также на всякий случай происходит проверка на наличие возможных значений, имеющихся в образце, в качестве действительных в ячейках квадрата. Если данные действительные значения оказываются обнаружены,

дальнейшая проверка также отменяется. При отмене или проведении дальнейшей проверки после нее происходит переход к следующему образцу, содержащему новую пару возможных значений. В случае если пара из пустых ячеек была составлена успешно и при этом дальнейшая проверка ранее не была отменена, переходим к поиску действительных чисел в пустых ячейках, при этом исключая из поиска парные ячейки.

```
for(counterZ=0;counterZ<i;counterZ++)
    Ccounter=0;
    for(a=0;a<b;a++)
        if(SNumbers[counterZ][ZNumbers[a]-
1]==ZNumbers[a])
            Ccounter++;
    if(Ccounter==1)
        denied=1;
    if(Ccounter>=2)
        CNumbers++;
        NVariants[counterZ]=1;
    else
        NVariants[counterZ]=0;
for(a=0;ZNumbers[a]!=0;a++)
    for(X=x*3-3;X<x*3;X++)
        for(Y=y*3-3;Y<y*3;Y++)
            if(Number[X][Y]==ZNumbers[a])
                denied=1;
```

Рисунок 19. Поиск совпадений с образцом.

В первую очередь стоит сказать, что описанная далее конструкция происходит при переборе всех возможных значений. Если одно из этих возможных значений совпадает с уже использованным значением в парных ячейках (рисунок 20), оно пропускается и происходит переход к следующему. Используя оставшиеся неиспользованные возможные значения, программа производит поиск чисел в текущем квадрате, при этом также исключая задействованные в парах пустые ячейки (NVariants[a]!=1). При обнаружении подобной ячейки происходит стандартная проверка на наличие в её области пересечения ячеек с действительным значением, равным проверяемому возможному значению. Если таких ячеек обнаружено в области пересечения не было, тогда временно сохраняются координаты прошедшей проверку ячейки. В конце концов, если по

некому возможному значению была обнаружена только одна подходящая ячейка, возможное значение становится действительным.

```
for(compare=0;compare<b;compare++)
    if(ZNumbers[compare]==NoS)
        OK=false;
if(OK)
    xSave=ySave=0;
    a=0;
    for(counterX=x*3-3;counterX<x*3;counterX++)
        for(counterY=y*3-3;counterY<y*3;counterY++)
            if(Number[counterX][counterY]==0)
                if(NVariants[a]!=1)
                .
                .
                .
if(Access==1)
    Number[xSave][ySave]=NoS;
```

Рисунок 20. Поиск непарных ячеек и незадействованных значений.

Далее, если не были проверены все возможные комбинации пар, состоящих из текущего количества возможных значений, происходит установка нового образца для сравнения с возможными значениями.

```
ZNumbers[b-1]++;
if(ZNumbers[b-1]==10)
    ZNumbers[b-1]=9;
    for(c=b-2;c>=0;c--)
        if(ZNumbers[c]!=ZNumbers[c+1]-1)
            ZNumbers[c]++;
            for(c=c+1;c<=b-1;c++)
                ZNumbers[c]=ZNumbers[c-1]+1;
            c=-2;
    if(c==-1)
        finish=1;
```

Рисунок 21. Создание нового образца для сравнения.

Логика перебора возможных значений для образца состоит в использовании неповторяющихся комбинаций для проверки пар возможных значений. При этом каждое последующее число образца больше предыдущего как минимум на 1. Например, сначала мы имеем в образце набор и 4 чисел 1, 2, 3, 4. После каждой проверки образца его последнее число увеличивается на 1 и, достигнув 9 (получив

набор 1,2,3,9), происходит следующее. Программа обнаруживает, что следующее число превысит предел (ZNumbers[b-1]=10). После этого находит, какое из чисел, стоящих перед превышающим предел, ещё не достигло предела. Это самое число и увеличивается на 1 (ZNumbers[c]++). В нашем случае это 3. После него все число приобретают значения предыдущего числа, увеличенного на 1 (т.е. новый вариант образца будет иметь набор 1,2,4,5).

Логику функции fifthLVL() также можно применить для проверки по прямым X и Y, однако использование одновременно всех проверок (в квадрате, по X и по Y) не дает ощутимой пользы. Сделан подобный вывод был исходя из сравнения проверок с использованием всех областей (в квадрате, по X и по Y) и лишь проверки в квадрате. В конечном счете, переменная Score, обозначающая общее использование всех основных функций, имела одно и то же значение для одних и тех же головоломок судоку. Поэтому, исходя из общей целесообразности, использована была только проверка в квадрате.

2.3. Метод подбора

Все описанные до этого момента методы строились на основании логики и применялись для разрешения некоторых конкретных ситуаций на поле судоку. Однако не всегда есть возможность полностью решить судоку, используя лишь описанные ранее правила. В конечном счете, возможна ситуация, при которой одно только применение этих самых методов не может привести к конечному решению головоломки. В таком случае не остается ничего иного, кроме как понадеяться на удачу и воспользоваться методом подбора. Однако стоит оговориться, что применяемый в данной программе метод подбора не перебирает слепо все возможные и невозможные значения для всех незаполненных и заполненных ячеек. Применение метода происходит для конкретной ячейки с последующей попыткой продолжить решение, применяя описанные методы в пунктах 2.2.1.-2.2.5.

2.3.1. Разбор примера работы метода перебора

Чтобы лучше понять логику метода подбора, применяемую в данной программе, для начала разберем пример (рисунок 22).

2	137				5	8	9	
	6			9		7	1	5
9	5		7					4
				6		9	5	8
		6			9	4		
5	9			3				
3			9		8			7
	4	9		2			8	
	8	5	1					9

Рисунок 22. Использование метода подбора.

Изображенные на рисунке значения были получены после использования всех пяти возможных методик для их получения. После того, как ни одна из функций не смогла найти какое-нибудь новое значение, программа перешла к использованию метода подбора. Основа данного метода, как уже было сказано прежде, лежит в использовании возможных значений каждой пустой ячейки и их применении в качестве действительных значений для данной ячейки с повторной попыткой продолжить поиск и дойти до конечного результата. Например, найдем возможные значения выделенной ячейки на рисунке 22. Как и указано, это возможные значения 1, 3 и 7. После выберем одно из значений и начнем решать согласно приведенным ранее методикам. В случае успеха мы получим решенную головоломку судоку. Если же решение не дойдет до конца или зайдет в тупик, можно взять другое возможное значение за действительное для рассматриваемой ячейки и попытаться снова. В конце концов, одно из значений должно быть верным и мы получим решенную головоломку (рисунок 21).

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

Рисунок 23. Решенная головоломка.

Однако далеко не всегда решение ограничивается одной ячейкой, как в приведенном здесь случае. Довольно часто решение просто не доходит до конца в одном или нескольких случаях для разных возможных значений одной ячейки. При этом ни одного случая правильного законченного решения так и не происходит. (Также при этом может присутствовать несколько ситуаций неправильного решения судоку.) В таком ситуации начинается перебор всех возможных значений следующей пустой ячейки. Перебор ячеек происходит до тех пор, пока не будет найдено решение головоломки.

Сам подобный перебор сначала проводится для одного значения в один промежуток времени. Но в случае, если решение после перебора по всем возможным значениям для всех пустых клеток так и не было найдено, количество одновременно используемых значений может быть увеличено. В данной работе я ограничился максимум двумя одновременно проверяемыми возможными значениями, поскольку все проверенные мною головоломки судоку получали итоговое решение максимум после применения двух возможных значений. При этом были ещё некоторые предпосылки для сокращения количества одновременно проверяемых возможных значений, однако подробнее об этом в главе 3.

2.3.2. Использование метода перебора в программе

После представленных разъяснений работы метода подбора перейдем к содержанию самой программы и рассмотрим описанную в теории реализацию (рисунок 24).

```
fail=checking(0);
if(flowLVL<16)
    fail=0;
if(fail==1)
    attempt=0;
    Op[0]=2;
    SaveScore=Score;
    for(x=0;x<9;x++)
        for(y=0;y<9;y++)
            SaveNumber[x][y]=Number[x][y];
    NumbOfOp=firstLVL(0,1);
```

Рисунок 24. Первоначальная проверка и сохранение результатов.

Удостоверившись, что ни одна функция больше не может найти новых действительных значений, программа переходит к представленному выше участку кода. В первой же строчке происходит запрос в функцию проверки, по какой причине использование стандартных функций для поиска ответа было завершено. Если было обнаружено, что головоломка решена или же то, что представленное судоку не имеет решения, на переменную fail данные события передадутся значениями 0 или 2 соответственно. При этом часть с проверкой по возможным значениям в пустых ячейках пропускается. Однако если переменная fail получает значение 1, обозначающее незаконченность решения, дальнейшая проверка состоится. Также проверяется количество действительных чисел, минимальное количество которых должно равняться 16. В случае если их количество меньше минимального, проверка также отменяется.

В качестве первоначальной установки происходит присваивание переменной Op[] значения 2, обозначающего порядковый номер проверяемого возможного значения. (Сам реальный порядковый номер можно вычислить арифметическим действием Op[]-1). Переменная attempt обозначает количество одновременно используемых возможных значений в качестве действительных (реальное количество является равным attempt+1). Далее также сохраняются полученное на данный момент значение переменной Score и все действительные значения ячеек. Напоследок вычисляется общее количество всех возможных значений всех ячеек

на поле судоку, обозначенное переменной NumbOfOp (при этом их реальное количество равно NumbOfOp-1). После всех приведенных действий предустановка заканчивается и происходит переход к основной части метода перебора.

Первую вещь, что мы встретим, перейдя к основной части перебора, это конструкция, необходимая для демонстрации работы программы, меняющая значения на линии прогресса работы программы. Не сказать о ней нельзя было, однако разбирать данную конструкцию нет нужды, так как она необходима только в эстетических целях и не несет на себе какой-либо нагрузки, влияющей на работу программы. Поэтому сразу пропустим её и перейдем к разбору следующей части, изображенной на рисунке 23.

```
Xray=checking(0);
if(Xray==2 and SaveOp!=0)
    NotAcceptable[SaveOp-2]=1;
if(Xray==1)
    b=0;
if(Xray==0)
    b=0;
    for(a=0;a<=z;a++)
        another=0;
        for(x=0;x<9;x++)
          for(y=0;y<9;y++)
            if(AnswerNumber[a][x][y]!=Number[x][y])
                another=1;
        if(another==0)
            if(Score<AnswerScore[a])
                AnswerScore[a]=Score;
            a=z+1;
    if(another==1)
        for(x=0;x<9;x++)
            for(y=0;y<9;y++)
                AnswerNumber[z][x][y]=Number[x][y];
        AnswerScore[z]=Score;
        z++;
```

Рисунок 25. Проверка результатов после использования возможных значений.

Посмотрев на первую строку кода можно вспомнить не раз уже встречавшуюся функцию проверки действительных значений. В данном случае ею также проверяются полученные в ходе решения с использованием возможного значения

действительные числа. Если полученная головоломка оказалась решена неверное, послужившее этому использованное возможное значение сохраняется как неприемлемое и впредь для поиска ответа данной головоломки использоваться не будет. Также переменной b обозначается возможность какого-либо решения. Если при использовании всех возможных значений какой-либо ячейки значение переменной b не будет равно 0, то это будет означать, что ни одно возможное значение ячейки не подошло и, следовательно, головоломка оказалась нерешаема.

В конечном счете, если функция checking() все-таки передала на переменную Xray значение 0, обозначающее правильность заполнения ячеек, происходит запись полученного ответа в массив ответов. Однако прежде, чем произойдет запись, ответ сверяется с другими полученными ответами. Тут стоит немного отвлечься и рассказать, что помимо одного правильного ответа, при решении судоку можно получить ещё несколько правильных ответов. Подобные несколько правильных ответов можно найти только методом перебора и при решении судоку, используя лишь описанные прежде основные методики, возникнуть не могут. Также подобное присутствие сразу нескольких правильных ответов означает, что автор головоломки не выполнил условия наличия одного единственного правильного решения составленной им головоломки судоку, что должно итак является неписаным правилом для каждого. Однако программа учитывает подобный исход и поэтому рассчитана на несколько возможных правильных решений.

Теперь вернемся к нашей программе. Проверка на уникальность решения производится по каждой ячейке с действительным значением. Если была найдена хотя бы одна ячейка (хотя их в таком случае должно быть как минимум 2), отличная от ячейки в записанном ответе, это означает, что весь ответ отличен от того, с которым происходит сравнение. Так сравнение происходит со всеми записанными ответами и, если не был найден записанный ответ, подобный только что полученному, полученный ответ записывается как ещё один вариант окончательного ответа. Если ответ уже был записан, значения переменных Score двух одинаковых ответов сравниваются между собой и сохраняется наименьшее.

```
if(Op[attempt]<=NumbOfOp)
    exit=0;
    while(exit==0)
        exit=1;
        for(x=0;x<9;x++)
          for(y=0;y<9;y++)
              Number[x][y]=SaveNumber[x][y];
        Score=SaveScore;
        for(c=0;c<=attempt;c++)
          if(attempt==0)
              SaveOp=Op[0];
          else
              SaveOp=0;
          answer[c]=firstLVL(0,Op[c]);
          if(SaveAnswer==0 or answer[0]>SaveAnswer)
              if(b==1)
                  compare=1;
              b=1;
              SaveAnswer=answer[0];
      |-> if(answer[c]<=1 and Op[attempt]>=2)
      |       denied=1;
      |       Op[attempt]=NumbOfOp;
      |       exit=1;
      |   else
      |       if(c>0 and answer[c]<=answer[c-1] and
answer[c]>1)
      |           Op[c]++;
      |---------> exit=0;
    Op[attempt]++;
    Done=1;
```

Рисунок 26. Применение переменной Op[] и назначение следующего возможного значения.

После того, как предыдущий ответ был проверен и возможно записан, происходит восстановление изначально подтвержденных действительных значений и переменной Score, которые были сохранены ещё на подготовительном этапе метода подбора. Такое восстановление происходит независимо от результата проверки в предыдущей рассмотренной части. Произведя восстановление, программа тут же записывает новые возможные значения в качестве действительных для новой попытки найти ответ судоку. Также если используется пока что только одно возможное значение за раз, тогда используемой значение сохраняется, чтобы впоследствии, если судоку окажется нерешаемой, записать данный вариант возможного значения, как неприемлемый. Сразу после записи

происходит проверка, был ли произведен переход на следующую ячейку. Если переход и вправду был произведен, но при этом все решения при использовании возможных значений прошедшей ячейки оказались неверными, происходит выход из цикла решения методом подбора.

Отдельно стоит упомянуть о проверке на соответствие использования возможных значений дозволенным пределам (данная проверка обозначена стрелочками). Данная проверка в основном нужна при использовании сразу нескольких возможных значений, поскольку исключает повторное использование уже проверенных пар возможных значений. Это производится путем проверки координат ячеек, возможные значения которых используются в данный момент. Данная проверка требует, чтобы каждое следующее значение координат ячейки используемого возможного значения было хотя бы на 1 больше предыдущего. Таким образом, убирается возможность использования тех комбинаций возможных значений, которые уже были проверены. Кроме того, выход за общее количество возможных значений также будет обнаружен (т.е. порядковый номер использованного возможного значения превысит общее количество этих возможных значений). Данная проверка производится для каждого используемого возможного значения отдельно.

Если все проведенные проверки пройдены успешно вместе с установкой новых возможных значений в качестве действительных, программа назначит значения, которые будут проверяться следующими, и перейдет к следующей части. Если же при проверке обнаружилось неправильное использованное возможное значение, программа заново запишет все сохраненные правильные действительные значения вместе с переменной Score и попытается использовать следующие возможные значения. Так будет происходить до тех пор, пока проверка не будет пройдена успешно или не будет превышено общее количество возможных значений. В случае достижения значения общего количества возможных значений (NumbOfOp) некоторой порядковой переменной Op[] программа попадет в следующую часть, корректирующую эти самые порядковые переменные.

```
if(Op[attempt]>NumbOfOp)
    another=1;
    if(Op[0]>=NumbOfOp-attempt)
        SaveAnswer=0;
        attempt++;
    if(attempt==2)
        compare=1;
    if(attempt<2)
        if(Op[attempt]!=0)
            for(c=attempt-1;c>=0;c--)
            .
            .
        else
            c=0;
            Op[0]=2;
            while(c<attempt)
                Op[c+1]=Op[c]-20;
                if(Op[c+1]<2)
                    Op[c+1]=2;
                c++;
```

Рисунок 27. Выход за допустимое порядковое значение.

В изображенную на рисунке 27 конструкцию программа производит переход только в случае, если последнее порядковый номер возможного значения превысил значение общего количества возможных значений (Op[attempt]>NumbOfOp). В таком случае в первую очередь проверяется причина данного превышения (были ли проверены все возможные комбинации или просто один порядковый номер возможного значения достиг предела). В случае если все комбинации оказались проверены, происходит увеличение общего количество одновременно используемых возможных значений. Также если было достигнуто максимальное количество одновременно используемых возможных значений, происходит уведомление программы об этом с помощью присвоения переменной compare значения 1 (далее будет произведен выход из цикла метода подбора).

В случае если увеличение количества одновременно используемых возможных значений (увеличение переменной attempt) не произошло только что (Op[attempt]!=0), происходит стандартная установка новых значений для порядковых номеров возможных значений, достигших своего предела (похожая установка применялась в функции fifthLVL(), рисунок 21). Если же увеличение

переменной attempt произошло только что, происходит первоначальная установка порядковых номеров. Далее происходит не указанная на рисунке 27 проверка на использование порядкового номера возможного значения (проверка по списку неприемлемых порядковых номеров) и, если порядковый номер не является неприемлемым, переход к завершающей части цикла метода подбора. Иначе, если порядковый номер запрещен, программа переходит к следующему и делает это до тех пор, пока номер не окажется приемлемым.

```
if(z>0 and !ui->checkU->isChecked())
    compare=1;
if(compare==1)
    Done=0;
    NAnswers=z;
    attempt=2;
    if(z>0)
        Score=AnswerScore[0]+50;
        for(x=0;x<9;x++)
            for(y=0;y<9;y++)
                Number[x][y]=AnswerNumber[0][x][y];
```

Рисунок 28. Проверка законченности цикла и запись ответа.

В конечном счете, если был найден ответ и при этом не надо перебирать все возможные варианты для поиска общего количества ответов или проверки уникальности решения (подробнее в пункте 3), происходит запись ответа. Записывается общее количество найденных решений за время работы цикла, устанавливается, что с этого момента цикл завершен, а также записывается сам ответ и переменная Score. Всегда записывается первый сохраненный ответ независимо от общего количества ответов или времени, затраченного на его поиск. Он же и выводится пользователю на интерфейс.

На этом описание основы решателя “Sudoku Solver” заканчивается. Все функции, представляющие ценность в плане поиска ответа для головоломки судоку были описаны. Также присутствует довольно большое количество вспомогательных функций, не столь необходимых для поиска решения, но добавляющих несколько полезных возможностей и облегчающих работу с приложением “Sudoku Solver”. Вкратце опишем их в следующей главе.

3. Разбор интерфейса приложения

Дабы разобрать имеющиеся дополнительные возможности нашего приложения взглянем на интерфейс программы (рисунок 29).

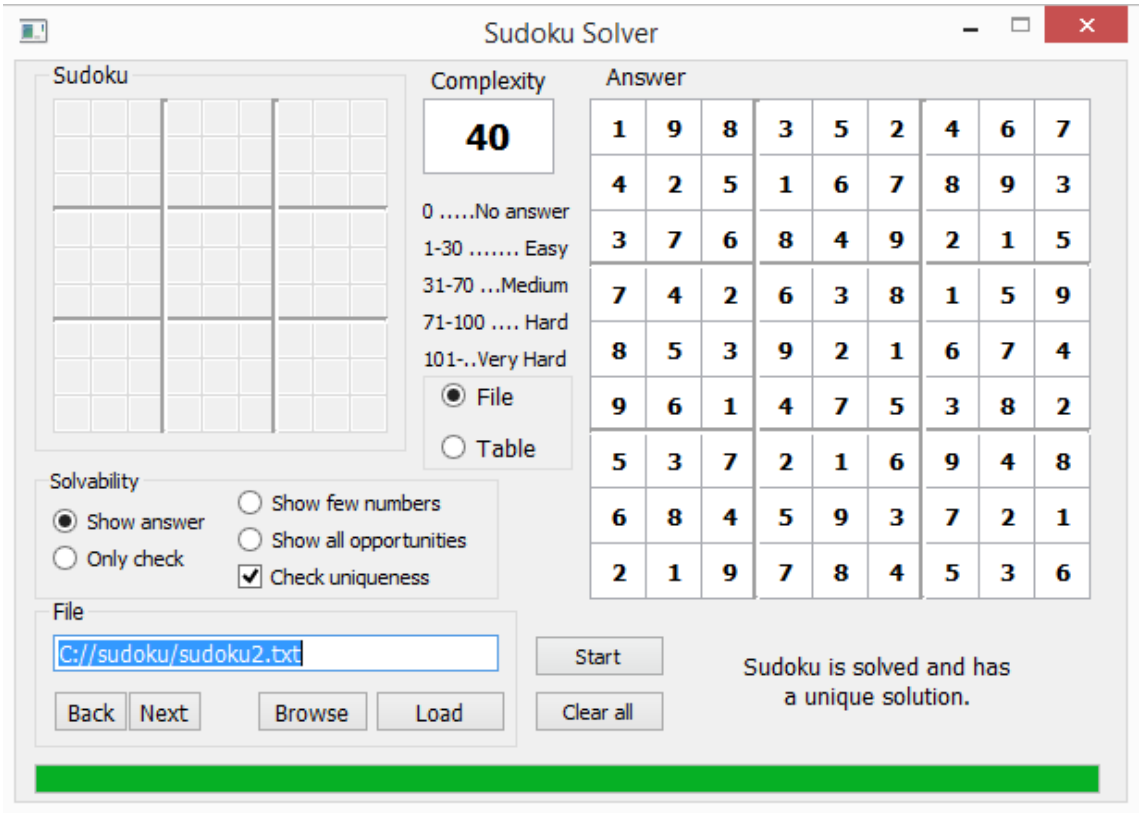

Рисунок 29. Рабочий интерфейс приложения

В первую очередь стоит обратить внимание на тот факт, что ввод чисел судоку для решения головоломки можно производить двумя способами: можно либо ввести все значения вручную, либо ввести имя файла, содержащего данные значения. В один момент времени можно выбрать или то, или иное. Переключение между двумя этими функциями осуществляется с помощью радиокнопок File и Table. В любом случае способ передачи информации в программу не влияет на работу самой программы, если все было сделано правильно. Однако стоит учесть, что данные желательно должны содержаться в файле .txt. Также они должны быть строго в определенном формате (пример рисунок 30).

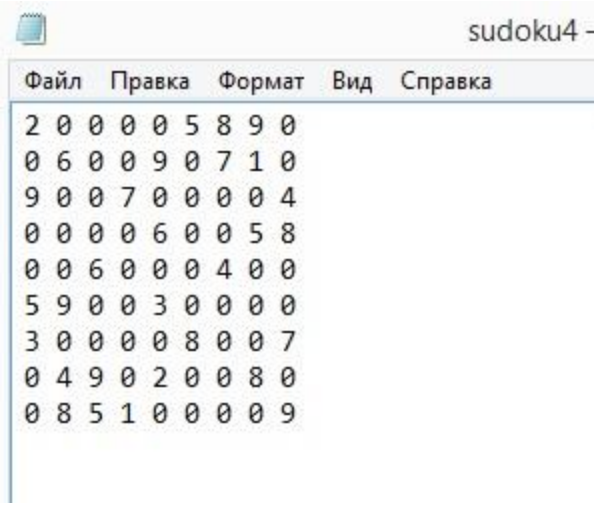

Рисунок 30. Пример данных, записанных в файл.

Как показано на рисунке, между числами должен стоять один пробел. Также каждая новая строка должна содержать данные. Пустых строчек или строчек с другой информацией между числами судоку быть не должно. Однако после записи всех девяти строчек с десятой строчки можно писать любые данные, так как они уже не считываются.

Также под строчкой, указывающей путь к файлу, присутствует четыре кнопки. Кнопка Load загружает данные, полученные из файла на интерфейс приложения. Сама загрузка данных не является обязательной, так как решение судоку можно производить, не загружая данные на интерфейс, а лишь используя их внутри программы. Однако загрузка из файла может пригодиться для демонстрации начальных значений, при этом без открытия каждого файла вручную. Также в случае, если начальные значения необходимо немного отредактировать и проверить, можно произвести загрузку из файла на интерфейс и после уже на самом интерфейсе изменить необходимые значения.

Кнопка Browse применяется для поиска текстового файла с заранее записанным вариантом головоломки в файловой системе компьютера. Варианты выбора для браузера ограничены только текстовыми файлами. Любые другие файлы просто не будут отображены.

Последние две кнопки Next и Back необходимы для переключения между головоломками судоку, предложенными автором приложения. Существует около 100 файлов, демонстрирующих работу приложения. Данные файлы содержать как

простые головоломки, так и сложные. Также они содержать нерешаемые головоломки, неправильно заполненные данные и всяческие примеры возможных ситуаций, которые могут возникнуть при использовании файлового способа передачи данных в программу. Данные файлы представлены исключительно для демонстрации работы и могут быть свободно изменены или удалены.

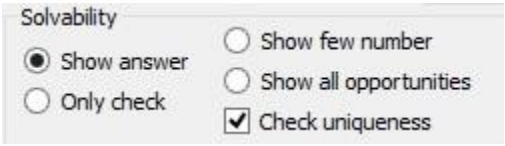

Рисунок 31. Опции решателя.

Далее рассмотрим варианты выбора решения, предлагаемые перед запуском поиска решения:

1) Вариант Show answer отображает полученный ответ полностью.

2) Опция Only check лишь выводит результат, полученный в ходе решения, в виде сообщения, при этом сам ответ не выводится. Данная опция полезна для тех, кто хочет лишь удостовериться в решаемости головоломки.

3) Выбрав опцию Show few numbers, программа произведет полное решение головоломки, однако выведет только значения в тех ячейках, которые были отмечены символом *. Конечно, также будут выведены значения, которые с самого начала были введены. Символ * может быть поставлен на место любой пустой ячейки и сработает как при использовании файла, так и интерфейса.

4) Используя опцию Show all opportunities, пользователь за место ответа получит все возможные значения для всех пустых ячеек.

5) Последняя опция Check uniqueness может использоваться или не использоваться по выбору пользователя. В отличии от предыдущих рассмотренных 4 опций, выбор между которыми точно должен быть совершен, пятая опция Check uniqueness работает независимо. Весь её смысл состоит в проверке уникальности решения для предоставленной пользователем головоломки. Проверка уникальности происходит при помощи рассмотренного в пункте 2.3.2. перебора всех возможных

комбинаций из возможных значений пустых ячеек и использовании данных возможных значений в виде подтвержденных.

Именно данной опцией контролируется количество проводимых проверок при поиске ответов. Если пользователь лишь хочет получить ответ, тогда проверка комбинаций возможных значений производится до момента нахождения первого ответа. После этого весь цикл метода подбора прерывается и выводится ответ. Если же опция Check uniqueness оказывается включенной, проверка производится до момента, когда будут проверены все возможные комбинации пар возможных значений. После этого на интерфейс будет выведен первый полученный ответ вместе с либо сообщением об уникальности решения, либо с сообщением о количестве полученных ответов.

В данном приложении я намеренно ограничил максимальное количество одновременно проверяемых значений двумя, так как каждое дополнительное значение значительно увеличивает время проверки на уникальность в случаях, если судоку оказывается очень сложной. Следствием подобного ограничения стал не полный набор решений после использования проверки на уникальность. Хотя обнаружение нескольких правильных ответов происходит, их число все-таки оказывается меньше реального количества ответов, содержащихся в проверяемом варианте судоку (хотя обнаружение головоломок с уникальным решением и без такового происходит безошибочно). Данная особенность, скорее всего, связана с тем, что многие решения просто не доходят до конца и на определенном этапе прерываются. Увеличение количества одновременно используемых в паре возможных значений могло бы помочь полностью устранить данную проблему, однако я считаю, что подобное увеличение нецелесообразным и по уже названным причинам не использую его. С тем же успехом можно было бы использовать перебор по всем возможным значениям для всей головоломки судоку с самого начала, без использования каких-либо логических конструкций и получить ответ намного быстрее.

Однако вернемся к рассмотрению интерфейса программы и пояснению его структуры.

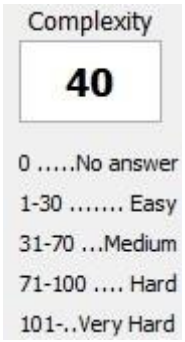

Рисунок 32. Сложность решения.

Небольшое окошко с цифрами с надписью над ним Complexity описывает общую сложность представленной программе и решенной ею головоломки судоку. В случае если решение не было получено, в окошке высветится значение 0. В ином случае некоторое числовое значение. Однако стоит учесть, что данная демонстрация довольно условна, так как реальное время и затраченные ресурсы на решение некоторой головоломки не могут быть полностью отображены лишь этим значением. Окошко Complexity может дать нам лишь общее представление о количестве использованных за время решения функций и подходов к ним.

Поля Sudoku и Answer являются полями ввода головоломки и вывода ответа соответственно. Кнопки же Start и Clear all являются кнопками команд начать решения или очистить все поля интерфейса тоже соответственно. Подробно описывать данные функции нет нужды, так как они говорят сами за себя.

На этом краткий обзор дополнительных функций приложения “Sudoku Solver” можно закончить. Все основные возможности программы были представлены и описаны. Таким образом, описание работы приложения “Sudoku Solver” можно завершить.

4. Заключение

В ходе проделанной работы была создана автономная кроссплатформенная рабочая модель приложения для решения головоломок судоку «Sudoku Solver». Все возникшие проблемы в ходе создания проекта были успешно разрешены. Общие логические модели, стандартно применяемые при решении головоломки судоку, были успешно интерпретированы в кодовом виде языка программирования C++. Также было включено несколько не столь необходимых, но в то же время придающих проекту завершенный вид, функций. Таким образом, поставленные в начале задачи для данной работы можно считать полностью выполненными.

Ссылки

[1] Статья в еженедельном международном научном журнале Nature под названием Mathematician claims breakthrough in Sudoku puzzle, опубликованная 6 января 2012 года. Также её можно найти, перейдя по ссылке http://www.nature.com/news/mathematician-claims-breakthrough-in-sudoku-puzzle-1.9751.

[2] Различные варианты судоку взяты с сайтов http://japonskie.ru, http://www.sudoku-club.ru, http://crosswords-world.net, http://golovolom.com.

[3] В качестве контрольного решателя судоку для проверки ответов использовался решатель на сайте http://www.sudoku-solutions.com.

[4] При создании приложения использовалась интегрированная среда разработки Qt Creator, которая была взята с официального сайта разработчиков данной IDE https://www.qt.io.

[5] Для работы с языком программирования C++, включающим также уникальные для Qt Creator конструкции, была использована книга «Программирование на языке C++ в среде Qt Creator» за авторством Е. Р. Алексеева, Г. Г. Злобина, Д. А. Костюка, О. В. Чесноковой, А. С. Чмыхало, опубликованная в Москве в 2015 году, которую также можно найти по ссылке http://www.electronics.lnu.edu.ua/fileadmin/add/flos/Book-qtC%2B%2B.pdf.

Lisa 1
rektori 27.02.2014 käskkirja nr 60 juurde

Lihtlitsents lõputöö üldsusele kättesaadavaks tegemiseks ja reprodutseerimiseks

Mina _____Ivan Bocharov_____ (*autori nimi*) (sünnikuupäev: ...21.06.1992...)

1. Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose
_____Sudoku_lahendaja_____
_____
_____,
(*lõputöö pealkiri*)

mille juhendaja on ____professor Peeter Ellervee_____,
(*juhendaja nimi*)

1.1. reprodutseerimiseks säilitamise ja elektroonilise avaldamise eesmärgil, sealhulgas TTÜ raamatukogu digikogusse lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;
1.2. üldsusele kättesaadavaks tegemiseks Tallinna Tehnikaülikooli veebikeskkonna kaudu, sealhulgas TTÜ raamatukogu digikogu kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.

2. Olen teadlik, et punktis 1 nimetatud õigused jäävad alles ka autorile.

3. Kinnitan, et lihtlitsentsi andmisega ei rikuta kolmandate isikute intellektuaalomandi ega isikuandmete kaitse seadusest ja teistest õigusaktidest tulenevaid õigusi.

_____ (*allkiri*)

__30.05.2015___ (*kuupäev*)

Lisa 2
rektori 27.02.2014 käskkirja nr 60 juurde

METAANDMED

Töö pealkiri (eesti keeles): Sudoku lahendaja

Töö pealkiri (inglise keeles): Sudoku solver

Autor: Ivan Bocharov

Juhendaja(d): prof. Peeter Ellervee

Kaitsmise kuupäev:

Töö keel: est / eng / rus: rus

Asutus (eesti keeles): TTÜ / TTÜ õppeasutus (nimi): Tallinna Tehnikaülikool

Asutus (inglise keeles): TTÜ / TTÜ õppeasutus (nimi): Tallinn University of Technology

Teaduskond (eesti keeles): Infotehnologia teaduskond

Teaduskond (inglise keeles): Faculty of Information Technology

Instituut (eesti keeles): Arvutitehnika instituut

Instituut (inglise keeles): Department of Computer Engineering

Õppetool (eesti keeles): Arvutisüsteemid

Õppetool (inglise keeles): Computer Systems

Märksõnad /kui on/ (eesti keeles):

Märksõnad /kui on/ (inglise keeles):

Õigused: juhul kui ligipääs on piiratud, siis sellekohane märkus