Кафедра Инфокоммуникационных технологий (ИКТ)

**КУРСОВАЯ РАБОТА**

по дисциплине «Технологии программирования»

на тему «Генератор «Судоку» на языке программирования c#»

Исполнители:

|  |  |  |
| --- | --- | --- |
| Раздел | Фамилия И.О. | Группа |
| Формулирование требований | Домашина Е. П. | БПИ-20-8 |
| Анализ предметной области | Домашина Е.П. | БПИ-20-8 |
| Разработка алгоритмов | Калюжная Ж.Р | БПИ-20-7 |
| Диаграммы переходов состояний | Аплемах Е.А. | БПИ-20-6 |
| Кодирование модулей | Пинегина А.А | БПИ-20-7 |
| Структурное тестирование и отладка | Пинегина А.А | БПИ-20-7 |
| Функциональное тестирование | Пинегина А.А | БПИ-20-7 |

Проверил:   
ст. преп. каф. ИКТ

Карпишук А.В.

\_\_\_\_\_\_\_\_\_\_\_\_\_

Москва, 2021

План работы:

1. Формулирование требований
2. Анализ предметной области
3. Разработка алгоритмов
4. Диаграммы переходов состояний
5. Кодирование модулей
6. Структурное тестирование и отладка
7. Функциональное тестирование
8. **Формулирование требований**

Общее описание:

Судоку – это цифровая головоломка. В переводе с японского «су» означает «цифра», а «доку» – «стоящая отдельно». Не надо гадать или копаться в книгах – только – логика и внимательность.

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.

Сложность судоку 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 часы.

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

Цель игры:

Заполнить пустые клеточки игрового поля цифрами от 1 до 9. С сетками 4 × 4 вам нужно использовать и заполнять цифры от 1 до 4; с сетками 6 × 6 цифры 1-6 и с сетками 9 × 9 содержат цифры от 1 до 9 соответственно.  
В каждом столбце, строке и блоке вы можете использовать каждую цифру только один раз.

Пример:

Середина верхнего ряда блоков 3×3 и середина нижнего ряда блоков 3×3 почти полностью заполнены. В середине верхнего блока три нерешённых числа – 1, 4, и 9. Анализируя такую ситуацию, можно вписать число 4 в центр блока, число 1 в правый верхний угол, а число 9 – в левый верхний угол. Аналогично можно поступить с нижним центральным блоком 3×3: в нём отсутствуют числа 6, 8 и 9. Ячейки заполняются последовательно: число 6 ставим в центр, число 9 в нижний правый угол, а число 8 в нижний левый угол.

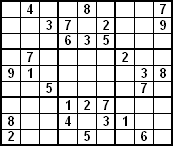


Рисунок 1 – пример игрового поля Судоку.

Перечень выполняемых функций:

1. После запуска программы перед пользователем появляется основная форма с двумя кнопками «проверить», «завершить» и сгенерированное поле игры .
2. Запрос и получение имени игрока;
3. Проверка каждого блока/строки/столбца;
4. Вывод на экран сообщения после проверки;
5. Ввод данных от игрока-человека желаемых цифр в желаемую область игрового поля;
6. Вывод на экран результата игры в виде сообщения о верности решения.

Требования к программе по каждой из функций, критерии качества выполнения:

1. Программа способна присвоить каждому игроку ПК и человеку игровое поле с сгенерированными рандомными цифрами от 1 до 9 в нём.
2. Программа предлагает проверить ответы на правильность и в случае ошибки выводит сообщение.
3. Программа способна в конце игры представить результат игры в виде сообщения о верности решения.

Краткое описание входных, выходных данных, результатов работы программы:

1. На вход программа получает имя игрока-человека (переменная типа «string»).
2. На вход программа получает цифры от 1 до 9, после чего генерирует их рандомным образом и распределяет по игровому полю, при этом оставляя пустые ячейки.
3. Программа получает на вход данные от игрока: какие цифры он хочет поставить в игровом поле, в какую ячейку.
4. Программа выводит на экран игровое поле с ячейками, некоторые из них заполнены сгенерированными рандомным образом цифрами от 1 до 9.
5. Программа выводит на экран сообщение об ошибке.
6. Программа по окончанию игры выводит на экран сообщение о результате игры.

Пользовательский интерфейс:

Примерный вид пользовательского интерфейса представлен на рисунке 2.

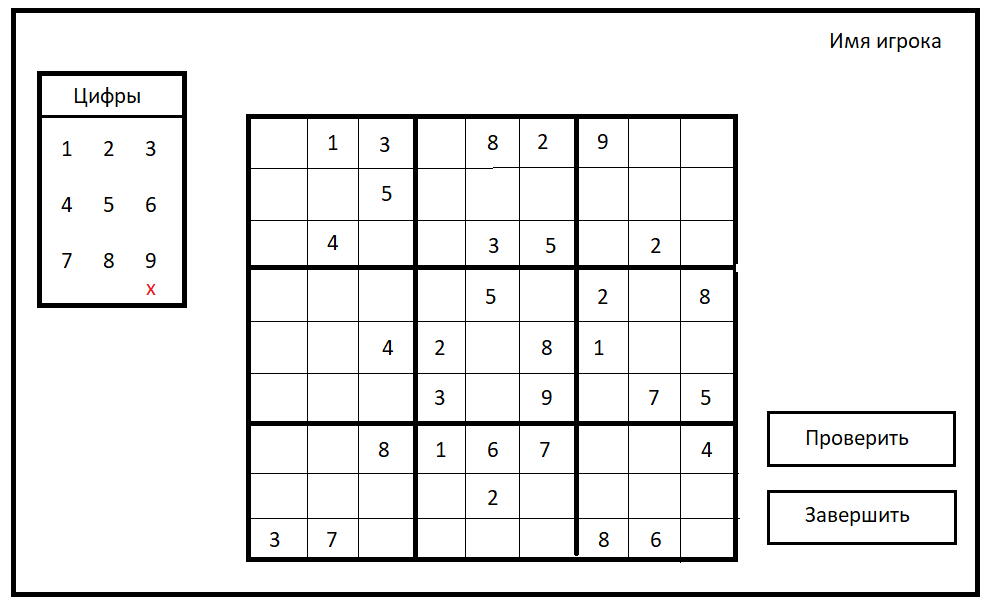


Рисунок 2 – Окно симуляции партии игры.

Список использованных источников

1. <https://ru.wikipedia.org/wiki/Судоку> – правила игры.
2. <https://www.sudokuweb.org/ru/kak-reshit-sudoku/> – цель игры.
3. **Анализ предметной области**

Опираясь на описанные в первом разделе функциональные возможности будущей программы, можно выделить основные задачи, требующие предварительного исследования:

* + выбор языка программирования;
  + выбор среды разработки;
  + генерация судоку.

Рассмотрим каждую задачу отдельно.

* 1. **Выбор языка программирования**

Если целью является эффективность, скорость выполнения и надежность, то C++ является одним из лидеров среди всех современных языков, и он идет впереди Си, потому что предлагает более широкие возможности из-за поддержки принципов ООП, которые отсутствуют в Си [7]. При разработке системного программного обеспечения, такого как драйверы и операционные системы, Си по-прежнему остается более предпочтительным выбором по сравнению с C++.

Python лучше всего использовать, если необходимо проектировать программное обеспечение, скорость которого не имеет особого значения [10]. Это также отличный выбор для новичков и для работы в таких областях, как анализ данных, машинное обучение и искусственный интеллект, в том числе из-за огромного количества свободно распространяемых Python-библиотек в этой области [8].

C# – это золотая середина среди современных высокоуровневых объектно-ориентированных языков. Если хочется отказаться от сложности C++, но также необходима скорость, которой не хватает Python, тогда C# – лучший выбор, потому что он также автоматически применяет оптимизацию, и это тот механизм, с которым начинающий программист может быть не столь хорошо знаком [4].

При сравнении различных высокоуровневых языков также следует отметить, что любая программа на языке C# является золотой серединой – при самом большом количестве строк кода по сравнению с программами на C++ и Python соответственно ее затраты памяти занимают промежуточное положение между ними, однако время выполнения является близким к минимальному значению [9].

Таким образом, в качестве языка программирования для разработки программного приложения был выбран C# с Windows.Forms в качестве основного средства реализации графического пользовательского интерфейса.

C# – это объектно-ориентированный язык программирования, который называют «Си-шарп». Он разработан Microsoft под руководством Андерса Хейлсберга и членов его команды в рамках инициативы .NET и был принят Европейской ассоциацией производителей компьютеров (ECMA) и Международной организацией по стандартизации (ISO). Язык был представлен в качестве конкурента Java в 2000 году на конференции профессиональных разработчиков, где основным докладчиком был основатель Microsoft Билл Гейтс. Символ # этого термина получил свое название от резкой музыкальной тональности, которая обозначает увеличение высоты тона на полтона.

Синтаксис C# исключительно выразителен, но при этом прост и легок в освоении. Разработчики, знакомые с любым из языков Си, C++, Java, обычно могут начать эффективно работать с C# за очень короткое время. Синтаксис C# упрощает многие трудности языков Си и C++ и предлагает уникальные функции, такие как обнуляемые, перечисляемые, делегируемые, лямбда-выражения и прямой доступ к памяти, которые недоступны в Java. C# поддерживает стандартные методы и типы, которые в некотором роде обеспечивают лучшую безопасность и производительность, а также итераторы, которые позволяют разработчикам легко использовать классы стандартной библиотеки. Выражения для реализации запросов, интегрированные в язык запросов LINQ, делают запросы первоклассной конструкцией языка.

Как объектно-ориентированный язык, C# поддерживает принципы инкапсуляции, наследования и полиморфизма. Все переменные и методы вместе с основным процессом, точкой входа в приложение, инкапсулируются в определение класса. Класс может наследоваться напрямую от базового класса, но также может реализовывать несколько интерфейсов. Для методов, которые переопределяют виртуальные методы в базовом (родительском) классе, требуется ключевое слово override для избежания незапланированного переопределения.

Если программисту необходимо взаимодействовать с другим различным программным обеспечением Windows, например с COM-объектами или собственными библиотеками Win32 DLL, он может сделать это на C# с помощью процедуры, которая называется Interoperation («взаимодействие»). Interoperation позволяет программам на C# делать примерно все, что может выполнять низкоуровневое приложение Си. C# даже обрабатывает указатели и концепцию «небезопасного» кода для тех частей, в которых прямой доступ к памяти имеет решающее значение.

Процесс создания программ на языке C# несложен по сравнению с Си и C++ и очень гибок по сравнению Java. Нет дополнительных заголовочных файлов, а также нет необходимости, чтобы методы и типы объявлялись в определенном формате или в определенном порядке. Файл с кодом на C# может описывать любое количество классов, структур, интерфейсов и событий.

Преимущества C# [1]:

1. C# объектно-ориентированный, в то время C++ представляет собой сочетание объектно-ориентированного и процедурно-ориентированного подхода;
2. C# типобезопасен (в программах, прошедших проверку согласования типов, исключена возможность возникновения ошибок согласования типов во время выполнения);
3. Программисту не нужно уделять много внимания таким проблемам, как потеря памяти, которая беспокоит программиста на C++;
4. Концепция сборки хорошо решает проблему контроля версий;
5. Простая в разработке, богатая библиотека классов позволяет легко реализовать многие функции;
6. Кроссплатформенность – приложение будет функционировать корректно в том случае, когда на компьютере установлена .NET Framework.
7. Автоматическая сборка мусора.

Недостатки C# [1]:

1. Программист не может выполнять низкоуровневые вещи, например, напрямую взаимодействовать с оборудованием через драйверы.
2. Многие языки программирования поставляются с независимым компилятором, который может напрямую интерпретировать верхние уровни для базовой аппаратной архитектуры чистого ассемблера. Вместо этого C# использует свой байт-код и JIT-компилятор, который в значительной степени встроен в структуру .NET и является основой структуры .NET в качестве посредника для машинного кода вместо прямого взаимодействия с оборудованием.
   1. **Выбор среды разработки**

На сегодняшний день для создания приложений на языке C# доступно несколько опций и инструментов. Некоторые из этих инструментов основаны на простых текстовых редакторах вместе с компилятором командной строки C#, а другие представляют собой графическую интегрированную среду разработки, которая предоставляет разработчику богатый набор функций [2]. Загрузив и установив бесплатный комплект для разработки программного обеспечения .NET Framework 4.0, разработчик может извлечь выгоду из множества утилит командной строки и компиляторов, а также предоставляемых им библиотек классов .NET. А написание приложения на языке C# или даже любого .NET-приложения может быть легко выполнено с помощью приложения Notepad вместе с компилятором командной строки C# под названием «csc.exe». Еще одно решение для создания приложений C# без использования IDE – это приложение Notepad++. Основные функции, предоставляемые инструментом Notepad++ в отношении разработки приложений C#, включают поддержку ключевых слов C# с подсветкой кода, автоматического завершения и сворачивания синтаксиса. В следующих абзацах кратко описаны решения IDE, доступные для использования при разработке приложений C#.

SharpDevelop – это бесплатная среда разработки с открытым исходным кодом, полностью написанная с использованием языка программирования C#. Он позволяет разработчикам разрабатывать проекты на C#, VB.NET, Python, IronRuby, F# и Boo. Среди множества функций, которые предоставляет эта IDE, важно упомянуть следующие: утилиты дизайнера форм, предназначенные для визуального проектирования форм Windows и баз данных, утилиты для просмотра объектов и написания кода, а также утилита редактирования кода, которая преобразует код из C# в VB и наоборот.

Microsoft Visual Studio [3] – это современная интегрированная среда, предназначенная для разработки программного обеспечения. Visual Studio включает в себя [редактор исходного кода](https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D0%B4%D0%B0%D0%BA%D1%82%D0%BE%D1%80_%D0%B8%D1%81%D1%85%D0%BE%D0%B4%D0%BD%D0%BE%D0%B3%D0%BE_%D0%BA%D0%BE%D0%B4%D0%B0) с поддержкой технологии [IntelliSense](https://ru.wikipedia.org/wiki/IntelliSense) и возможностью простейшего [рефакторинга кода](https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D1%84%D0%B0%D0%BA%D1%82%D0%BE%D1%80%D0%B8%D0%BD%D0%B3). Встроенный [отладчик](https://ru.wikipedia.org/wiki/Microsoft_Visual_Studio_Debugger) может работать как отладчик уровня исходного кода, так и отладчик машинного уровня. Присутствует удобный редактор форм для упрощения создания графического интерфейса приложения, веб-редактор, дизайнер [классов](https://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D0%B0%D1%81%D1%81_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) и дизайнер [схемы базы данных](https://ru.wikipedia.org/wiki/%D0%A1%D1%85%D0%B5%D0%BC%D0%B0_%D0%B1%D0%B0%D0%B7%D1%8B_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85). Visual Studio позволяет создавать и подключать сторонние дополнения ([плагины](https://ru.wikipedia.org/wiki/%D0%9F%D0%BB%D0%B0%D0%B3%D0%B8%D0%BD)) для расширения функциональности практически на каждом уровне, включая добавление поддержки систем [контроля версий исходного кода](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D1%83%D0%BF%D1%80%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D1%8F_%D0%B2%D0%B5%D1%80%D1%81%D0%B8%D1%8F%D0%BC%D0%B8).

Основными преимуществами Visual Studio являются:

* система проверки и дополнения ввода текста IntelliSense;
* быстрый рефакторинг кода;
* простая реализация общих задач и индивидуальный подход;
* быстрое создание высококачественного кода;
* возможность реализации идей и решений для широкого спектра платформ, включая Windows, Windows Server, веб-среду, облачную среду, Office и SharePoint;
* огромное количество плагинов;
* возможность интеграции проектов на разных языках;
* поддержка множества языков, заточенных под необходимую задачу.

Главными недостатками среды является ее ориентированность на платформу .NET и тот факт, что точно так же, как и в случае с Windows, лицензия VS для коммерческих продуктов является платной.

Eclipse – одна из самых популярных мультиязычных сред. Ориентирована преимущественно на разработку Java-приложений, но полезна и для кодов на C#.

Плюсы:

* Множество плагинов. У Eclipse едва ли не самое большое число надстроек «на все случаи жизни».
* Активное сообщество. Помогает быстрее освоить среду разработки, выпускает новые плагины.
* Отличные компилятор и отладчик. Первый работает на порядок быстрее, чем у конкурентов, второй – показывает потоки, пересечения, позволяет гибко управлять ходом отладки.
* Кастомизация. Благодаря плагинам и настройкам можно полностью персонализировать Eclipse.
* Бесплатность. Это open-source проект, абсолютно бесплатный.
* Высокая функциональность. Благодаря разработчикам-официалам и членам сообщества с помощью Eclipse можно провести любой C#-продукт по полному циклу разработки.

Минусы:

* Сложность. Как и любой функциональный продукт, Eclipse может показаться новичку слишком сложным.
* Нет гарантий надежности. Поскольку плагины создаются сообществом, за их качество отвечает только разработчик. Кроме того, сами создатели Eclipse с каждой новой версией не успевают исправить ошибки предыдущих версий среды.

MonoDevelop – это интегрированная среда разработки (IDE), которая является эквивалентом Microsoft Visual Studio IDE. Она поддерживает широкий спектр языков программирования (включая C#) и может читать и записывать файлы решений (.sln) и проектов (.csproj), используемые Visual Studio. В настоящее время поддерживаются все версии Visual Studio. MonoDevelop была запущена в 2003 году с целью перенести SharpDevelop на Mono и GTK#.

Многие из функций, к которым разработчики привыкли из других IDE, таких как Visual Studio или Eclipse, также доступны в MonoDevelop. Эти функции включают в себя, помимо прочего, автоматическое завершение, навигационную панель, браузер классов, интеграцию с контролем версий и отладчик. Диспетчер надстроек позволяет разработчикам расширять функциональность своей IDE с помощью надстроек сторонних разработчиков. Подобно базовой инфраструктуре Mono, MonoDevelop работает в основных операционных системах (Linux, MacOS, Windows). Он использует мультиплатформенный проект GTK+ для рисования своего графического интерфейса и виджетов.

Результаты сравнения сред разработки для языка программирования C# приведены в таблице 1.

Таблица 1 – Сравнение интегрированных сред разработки языка C#

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Критерий | Среда | Eclipse | VS 2019 | MonoDevelop | SharpDevelop |
| Лицензия | свободная | проприетарная | свободная | свободная |
| Последняя стабильная версия | 2017 | 2019 | 2016 | 2016 |
| Кроссплатформенная | да | нет | да | нет |
| Поддержка Unix-платформ | да | нет | да | нет |

По результатам сравнения сред разработки для дальнейшего использования была выбрана интегрированная среда Microsoft Visual Studio 2019 Community для Windows, которая получает регулярные обновления от Microsoft и является бесплатной средой для учебных проектов.

Для генерации, проверки и решения судоку было создано немало свободно распространяемых библиотек с открытым исходным кодом, в том числе для языка программирования C# с набором .NET-средств (интерфейс программирования приложений Windows.Forms) [5]. Как правило, генератор формирует доску для судоку на основе нескольких уровней сложности (по числу удаленных ячеек). Для решения судоку используется алгоритмом поиска с возвратом, способный решать головоломки любой сложности.

* 1. **Генерация судоку**

Алгоритм генерации может быть описан следующим образом [6]:

1. Формирование базовой сетки Судоку.

Сетка должна подчиняться правилам Судоку. В первой строке размещаются числа от 1 до 9, в строке ниже эти числа смещаются на три позиции влево, то есть в нее записываются значения от 4 до 9 и затем от 1 до 3. Следующая строка со смещением на три: 7..9 и 1..6.

Аналогично заполняются следующие три строки, в первой из которых записываются числа со смещением, равным единицы, то есть от 2 до 9 и заканчивая 1.

Результат такого заполнения приведен на рисунке 3.

1. Перетасовка сетки.

Есть несколько видов перестановок, после выполнения которых таблица Судоку останется в допустимом состоянии.

К ним относятся, например, транспонирование всей таблицы, при котором столбцы становятся строками и наоборот, обмен двух районов по горизонтали или по вертикали, обмен двух строк или столбцов в пределах одного района (области 3 х 3). Многократные перемешивания позволяют получить случайный вариант доски Судоку.

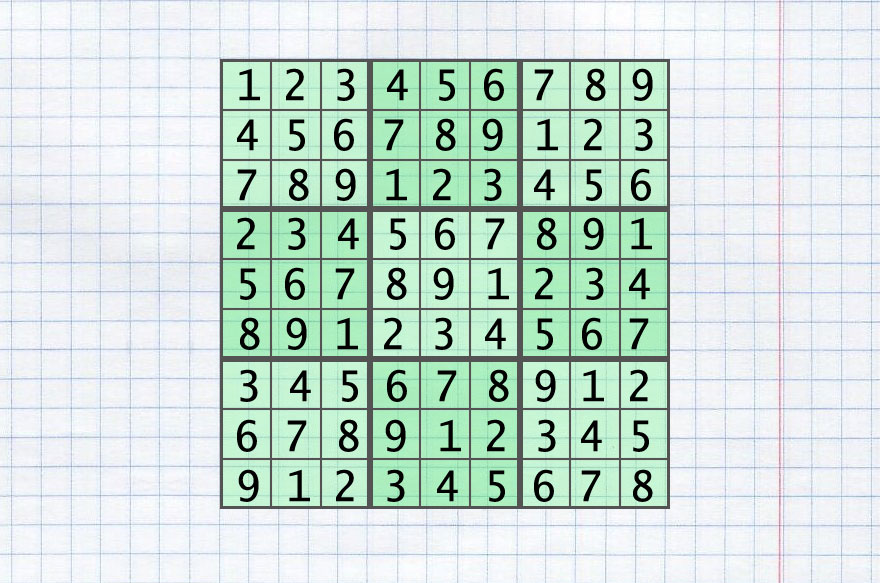


Рисунок 3 – Базовая сетка Судоку

1. Удаление клеток.

После полученного решения необходимо получить задачу (именно в такой последовательности можно гарантировать однозначность решения). И это самая сложная часть. Какое количество можно убрать, чтобы гарантировать однозначность решения? Это один из важных факторов, от которого зависит сложность Судоку. Всего в Судоку 81 клетка – обычно считают уровень легким, когда на поле есть 30-35 «подсказок», средним – 25-30, и сложным – 20-25. Это данные большого набора реальных примеров. Нет никаких законов для сложности. Можно сделать 30-клеточный неразрешимый вариант и 22 клеточный «лёгкий».

Удаление можно выполнить различными способами:

А) Случайный подход – можно попробовать выкинуть 50-60 клеток наугад, но где вероятность что Судоку можно будет решить? Например, если заполнены 3 строки (то есть 27 клеток).

Б) Случайный подход с простым ограничением – для примера можно взять некое число N в качестве предела, так что N строк и столбцов могут быть пустыми. Принимая N = 0 – для легкого уровня, N=1 – для среднего, N=2 – для сложного.

Алгоритм, используемый здесь, следующий:

А) Выбрать случайную ячейку N

Б) Отметить N просмотренной

В) Удалить N

Г) Посчитать решения. Если оно не единственное, то вернуть N обратно

На выходе получится самая сложная из возможных вариантов Судоку для данного перемешивания.

* 1. **Список использованных источников**
  2. Шилдт Г. Полный справочник по С#, 4-ое издание / Г. Шилдт пер. с англ. – М.: Издательский дом «Вильямс», 2015. – 704 с.
  3. Документация по семейству продуктов Visual Studio [Электронный ресурс]. – Режим доступа: https://docs.microsoft.com/ru-ru/visualstudio/?view=vs-2019 – (Дата обращения – 02.12.2021).
  4. C# docs [Электронный ресурс]. – Режим доступа: https://docs.microsoft.com/ru-ru/dotnet/csharp/ – (Дата обращения – 02.12.2021).
  5. Index | TIOBE [Электронный ресурс]. – Режим доступа: https://www.tiobe.com/tiobe-index/ – (Дата обращения – 02.12.2021).
  6. Sudoku Library [Электронный ресурс]. – Режим доступа: https://github.com/firateski/SudokuLibrary – (Дата обращения – 03.12.2021).
  7. Алгоритм генерации Судоку [Электронный ресурс]. – Режим доступа: https://habr.com/ru/post/192102/ – (Дата обращения – 04.11.2021).
  8. Aldrawiesh K., Al-Ajlan A., Al-Saawy Y., Bajahzar A. «A comparative study between computer programming languages for developing distributed systems in web environment», November, 2009
  9. Alomari Z., Halimi O., Sivaprasad K., Pandit C. «Comparative studies of six programming languages», April, 2015
  10. Pandit C. «How many programming languages are there in the world», November, 2017

Garcia R., Jarvi J., Lumsdaine A., Siek J., Willcock J. «An extended comparative study of language support for generic programming

1. **Разработка алгоритмов**

Среди функций программы, перечисленных в предыдущих пунктов, а также сопутствующих задач, большей алгоритмической проработки требует функция, отвечающая за генерацию игры (создания сетки с «удаленными» из нее цифрами»), поскольку само решение игры исходит от пользователя, следовательно, требуется специальный алгоритм, который сможет создать сетку.

По итогу - функции/алгоритмы программы требующие проработки:

* Заполнение сетки перед игрой
* Вывод цифр доступных для использования
* Проверка сетки на правильность
  1. – **Заполнение сетки перед игрой**

Сетка должна подчинятся правилам Судоку. Создаем базовую сетку. Размещаем в первую строку 1 2… 8 9, в строках ниже смещаем на 3 позиции влево, т.е. 4 5… 2 3 и 7 8… 5 6.

Далее переходя в следующий блок по вертикали смещаем на 1 позицию влево предыдущий блок.

Поле может менять свои значения, не меняя базовой сетки, а лишь наклоняя или инвертируя сетку, также меняя блоки местами или меняя строки/столбцы в радиусе одной клетки. Но, также можно сетку создать используя в начале «заполнения» случайные цифры, которые по мере заполнения приведут поле к тому состоянию, что пк сам с собой сыграет, а потом предложит пользователю.

Псевдо код заполнения:

**static** **void** ChangeTwoCell(**ref** **int**[,] grid2, **int** findValue1, **int** findValue2)

        {

**int** xParm1, yParm1, xParm2, yParm2;

            xParm1 = yParm1 = xParm2 = yParm2 = 0;

**for** (**int** i = 0; i < 9; i += 3)

            {

**for** (**int** k = 0; k < 9; k += 3)

                {

**for** (**int** j = 0; j < 3; j++)

                    {

**for** (**int** z = 0; z < 3; z++)

                        {

**if** (grid2[i + j, k + z] == findValue1)

                            {

                                xParm1 = i + j;

                                yParm1 = k + z;

                            }

**if** (grid2[i + j, k + z] == findValue2)

                            {

                                xParm2 = i + j;

                                yParm2 = k + z;

                            }

                        }

                    }

                    grid2[xParm1, yParm1] = findValue2;

                    grid2[xParm2, yParm2] = findValue1;

                }

            }

* 1. – **Вывод доступных цифр и работа с ними**

Необходимо вывести небольшое окно, в котором будут представлены цифры от 1 до 9 в ячейках, которые, при нажатии на них, будут закрепляться за последующим нажатием в ячейку на игровом поле, а также сменяться на другую цифру при выборе иного числа, либо «открепляться» при нажатии вне контура сетки.

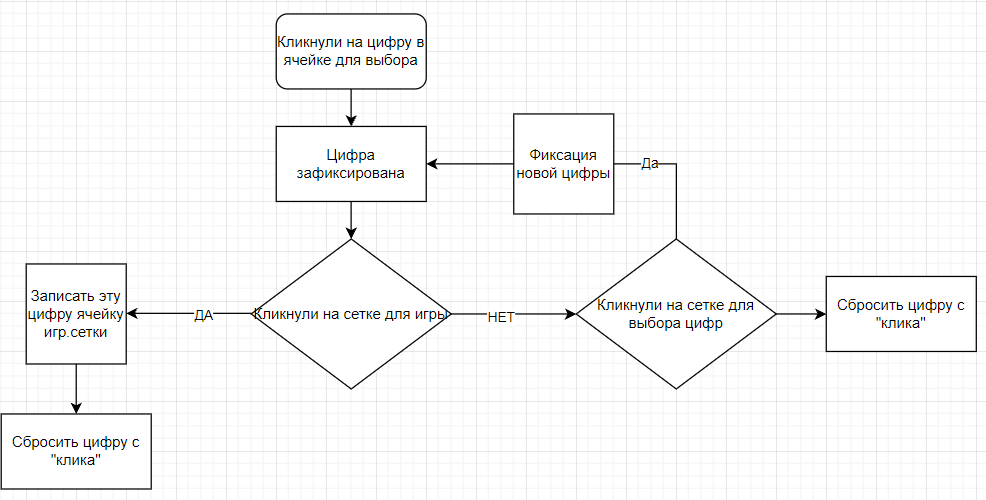


Рисунок 3.2 – алгоритм работы цифрами для использования в игре

* 1. – **Проверка сетки на правильность**

Необходимо, чтобы сетка Судоку была заполнена по всем правилам игры, в блоке нет повторяющихся цифр, а в трех блоках в строчке/столбце – есть все цифры от 1 до 9. Сделать это можно рекурсивным методом.

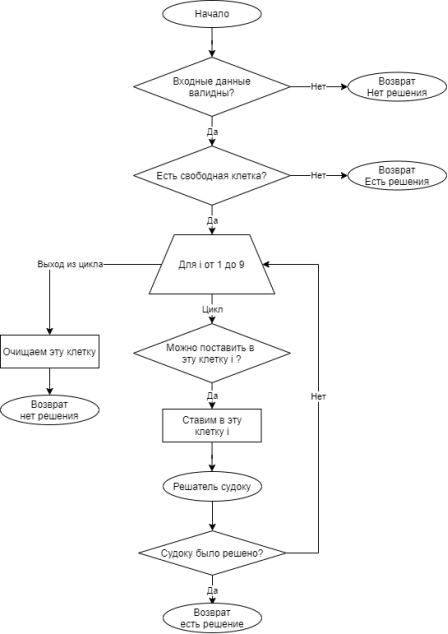


Рисунок 3.3 – Проверка рекурсивного решения Судоку

# Проектирование диаграммы переходов состояний

# 

Эта диаграмма описывает общую работу программы. Вначале у игрока 3 кнопки: выбор сложности, старт, выход. После нажатия на старт начинается игра, в конце игры игроку предлагается повторить игру или вернуться в начальный интерфейс. После нажатия на выбор сложности будут предложены 3 вида сложности, затем, после нажатия на один из них, происходит возврат в главное меню, в конце игры игроку предлагается повторить игру или вернуться в начальный интерфейс. После нажатия на выход программа закрывается.

**Кодирование модулей**

Для начала созданим функцию генерации карты GeneralMap. Здесь перебираются все клетки игры и значения сохраняются в массив. Также было создано пять функции, который расставляют цифры в соответствующем порядке, который описан в каждой функции: MatrixTransponation, SwapRowsInBlock, SwapColumsInBlock, SwapBlocksInRow, SwapBlocksInColumm. Способ сортировки выбираем рандомно с помощью функции SuffleMap. Также в программе имеется функция создания карты CreateMap. В ней создаем поле игры. Для этого следует создать кнопку, задать ей размер и координаты. Таким образом мы получим поле для игры в судоку. Далее нам необходимо скрыть некоторые ячейки и позволить пользователю подставить туда значения. Реализовано это в функциях: HideCells, OnCellPressed. В первой мы скрыли определенное количество ячеек и сделали уже сгенерированные неактивными. А вторая функция позволяет выбирать какую цифру поставить – нажатием на ячейку. Также существует функция проверки button1\_Click в ней проверяется правильность ввода пользователем и выводится соответствующее сообщение. После результата “Верно” поле игры генерируется снова. Есть еще функция завершения игры button2\_Click в ней игра сбрасывается и поле генерируется снова.

Листинг программы

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace Sudoky

{

public partial class Form1 : Form

{

const int n = 3;

const int sizeButton = 50;

public int[,] map = new int[n \* n, n \* n]; //массив с размером карты для игры в судоку

public Button[,] buttons = new Button[n \* n, n \* n];

public Form1()

{

InitializeComponent();

textBox1.AcceptsReturn = textBox1.ReadOnly;

GenerateMap();

}

public void GenerateMap() //функция генерация карты, здесь генерируем поле игры

{

for (int i = 0; i < n \* n; i++)

{

for (int j = 0; j < n \* n; j++)

{

map[i, j] = (i \* n + i / n + j) % (n \* n) + i;

}

}

//рандомно выбираем способ расстановки цифр

Random r = new Random();

for(int i = 0; i < 10; i++)

{

SuffleMap(r.Next(0, 5));

}

CreateMap();

HideCells();

}

public void SuffleMap(int i)

{

/\*функция для выбора алгоритма по которому пудут расставляться значения

\* в поле игры. Их всего 5.

\*/

switch (i)

{

case 0:

MatrixTransponation();

break;

case 1:

SwapRowsInBlock();

break;

case 2:

SwapColumsInBlock();

break;

case 3:

SwapBlocksInRow();

break;

case 4:

SwapBlocksInColumm();

break;

default:

MatrixTransponation();

break;

}

}

public void HideCells()//функция для того, чтобы скрыть ячейки сгенерированные

{

int N = 40;

Random r = new Random();

while (N > 0)

{

for (int i = 0; i < n \* n; i++)

{

for (int j = 0; j < n \* n; j++)

{

if (!string.IsNullOrEmpty(buttons[i, j].Text))

{

int a = r.Next(0, 3);

buttons[i, j].Text = a == 0 ? "" : buttons[i, j].Text;

buttons[i, j].Enabled = a == 0 ? true : false;

if (a == 0)

N--;

if (N <= 0)

break;

}

if (N <= 0)

break;

}

}

}

}

public void SwapBlocksInColumm()

{

Random r = new Random();

var block1 = r.Next(0, n);

var block2 = r.Next(0, n);

while (block2 == block1)

{

block2 = r.Next(0, n);

}

block1 \*= n;

block2 \*= n;

for (int i = 0; i < n \* n; i++)

{

var k = block2;

for (int j = block1; j < block1 + n; j++)

{

var temp = map[i, j];

map[i, j] = map[i, k];

map[i, k] = temp;

k++;

}

}

}

public void SwapBlocksInRow()

{

Random r = new Random();

var block1 = r.Next(0, n);

var block2 = r.Next(0, n);

while (block1 == block2)

{

block2 = r.Next(0, n);

}

block1 \*= n;

block2 \*= n;

for (int i = 0; i < n \* n; i++)

{

var k = block2;

for(int j = block1; j < block1 +n; j++)

{

var temp = map[j, i];

map[j, i] = map[k, i];

map[k, i] = temp;

k++;

}

}

}

public void SwapRowsInBlock()

{

Random r = new Random();

var block = r.Next(0, n);

var row1 = r.Next(0, n);

var line1 = block \* n + row1;

var row2 = r.Next(0, n);

while (row2 == row1)

{

row2 = r.Next(0, n);

}

var line2 = block \* n + row2;

for(int i = 0; i < n\*n; i++)

{

var temp = map[line1, i];

map[line1, i] = map[line2, i];

map[line2, i] = temp;

}

}

public void SwapColumsInBlock()

{

Random r = new Random();

var block = r.Next(0, n);

var row1 = r.Next(0, n);

var line1 = block \* n + row1;

var row2 = r.Next(0, n);

while (row2 == row1)

{

row2 = r.Next(0, n);

}

var line2 = block \* n + row2;

for (int i = 0; i < n \* n; i++)

{

var temp = map[i, line1];

map[i,line1] = map[i, line2];

map[i,line2] = temp;

}

}

public void MatrixTransponation()

{

int[,] tMap = new int[n \* n, n \* n];

for(int i = 0; i < n\*n; i++)

{

for(int j = 0; j < n\*n; j++)

{

tMap[i, j] = map[j, i];

}

}

map = tMap;

}

public void CreateMap() //функция создания карты

/\*В этой функции создаем "кнопки" и заполняем их цифрами, которые лежат в массиве map

\* Таким образом у нас получится поле для игры в судоку

\*/

{

for (int i = 0; i < n \* n; i++)

{

for (int j = 0; j < n \* n; j++)

{

Button button = new Button();

buttons[i, j] = button;

button.Size = new Size(sizeButton, sizeButton);

button.Text = map[i, j].ToString();

button.Click += OnCellPressed;

button.Location = new Point(j \* sizeButton, i \* sizeButton);

this.Controls.Add(button);

}

}

MessageBox.Show("Введите пожалуйста имя игрока в поле в правом верхнем углу!");

}

public void OnCellPressed(object sender, EventArgs e)

{

/\*функция для генерации цифр

\*/

Button pressedButton = sender as Button;

string buttonText = pressedButton.Text;

if(string.IsNullOrEmpty(buttonText))

{

pressedButton.Text = "1";

}

else

{

int num = int.Parse(buttonText);

num++;

if(num == 10)

{

num = num - 1;

}

pressedButton.Text = num.ToString();

}

}

private void button1\_Click(object sender, EventArgs e)//функция проверки

/\*Необходимо проверить правильность ввода пользователем значений

\* Для этого сравним то что ввел пользователь с тем, что лежит в массиве map

\* Если пользователь ввел ошибочное значение, то выводи сообщение "Неверно!"

\* Если пользователь ввел все правильно, то сообщение "Верно!"

\* После победы завершаем игру и генерируем поле сначала

\*/

{

for(int i = 0; i < n\*n; i++)

{

for(int j = 0; j < n\*n; j++)

{

var btnText = buttons[i, j].Text;

if(btnText != map[i,j].ToString())

{

MessageBox.Show("Неверно!");

return;

}

}

}

MessageBox.Show("Верно!");

for(int i = 0; i < n\*n; i++)

{

for(int j = 0; j < n\*n; j++)

{

this.Controls.Remove(buttons[i, j]);

}

}

GenerateMap();

}

private void button2\_Click(object sender, EventArgs e)//функция для завершения игры

{

for (int i = 0; i < n \* n; i++)

{

for (int j = 0; j < n \* n; j++)

{

this.Controls.Remove(buttons[i, j]);

}

}

GenerateMap();

}

private void textBox1\_TextChanged(object sender, EventArgs e)

{

}

}

}

**Структурное тестирование и отладка.**

Структурное тестирование — это тип тестирования программного обеспечения, в котором для тестирования используется внутренний дизайн программного обеспечения, иными словами, тестирование программного обеспечения, выполняемое командой, которая знает фазу разработки программного обеспечения, называется структурным тестированием.

Были выбраны следующие методы структурного тестирования:

1. Покрытие операторов. Этот критерий предполагает выбор такого тестового набора данных, который вызывает выполнение каждого операто­ра в программе хотя бы один раз.
2. Покрытие узлов ветвления (покрытия решений). Этот критерий предполагает разработку такого количества тестов, чтобы в каждом узле ветвления был обеспечен переход по веткам "истина" и "ложь" хотя бы один раз.
3. Покрытие условий. Если узел ветвления содержит более одного ус­ловия (сложное условие), тогда нужно разрабатывать число тестов, доста­точное для того, чтобы возможные результаты каждого условия в решении выполнялись по крайней мере 1 раз.
4. Комбинаторное покрытие условий. Этот критерий требует создания такого числа тестов, чтобы все возможные комбинации результатов условия в каждом решении выполнялись хотя бы 1 раз.

В нашем случае мы тестируем методы вручную, создавая такой набор входным данных (нажатий кнопок в игре Судоку), чтобы выполнялись условия выше.

6.1 Тестирование метода 1

public void OnCellPressed(object sender, EventArgs e)

{

/\*функция для генерации цифр

\*/

Button pressedButton = sender as Button;

string buttonText = pressedButton.Text;

if(string.IsNullOrEmpty(buttonText))

{

pressedButton.Text = "1";

}

else

{

int num = int.Parse(buttonText);

num++;

if(num == 10)

{

num = num - 1;

}

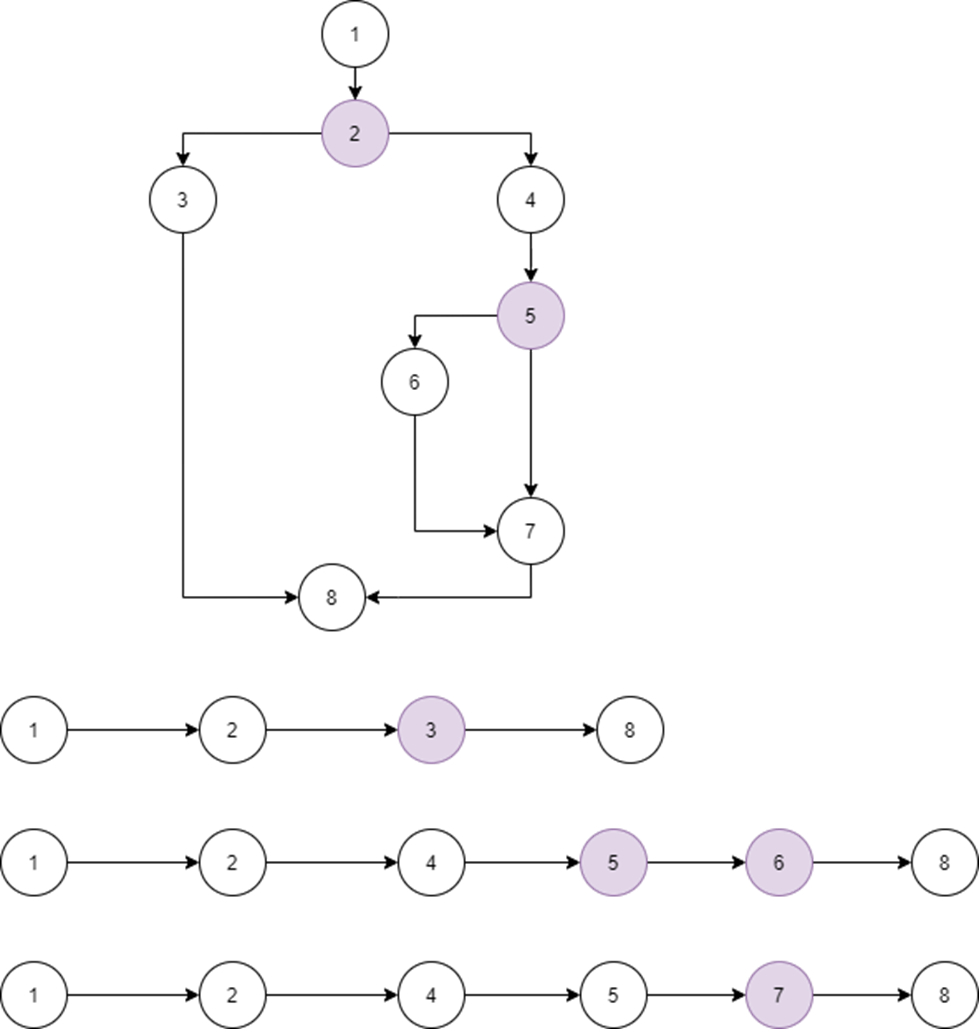
pressedButton.Text = num.ToString();

}

}

Метод отвечает за нажатие кнопки и смену значения в нем

Потоковый граф



Цикломатическая сложность =3 ( по формуле V (G) = P + 1

Необходимо 3 теста

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **№** | **Входные данные** | **Ожидаемый результат** | **Полученный результат** | **Вывод** |
| 1 | Нажатие 1 раз | 1 | 1 | Пройден |
| 2 | Нажатие от 2 до 9 раз | От 2 до 9 | 5 | Пройден |
| 3 | Нажатие 10 раз | 1 | 9 | Не пройден |

Последний тест не прошел. Необходимо исправить и доработать программу, чтобы при нажатии на кнопку 10 раз появлялась цифра 1, потому что у пользователя должна быть возможность поменять цифру если он ввел ошибочно.

Так будет выглядеть метод после исправления ошибки:

public void OnCellPressed(object sender, EventArgs e)

{

/\*функция для генерации цифр

\*/

Button pressedButton = sender as Button;

string buttonText = pressedButton.Text;

if(string.IsNullOrEmpty(buttonText))

{

pressedButton.Text = "1";

}

else

{

int num = int.Parse(buttonText);

num++;

if(num == 10)

{

num = 1;

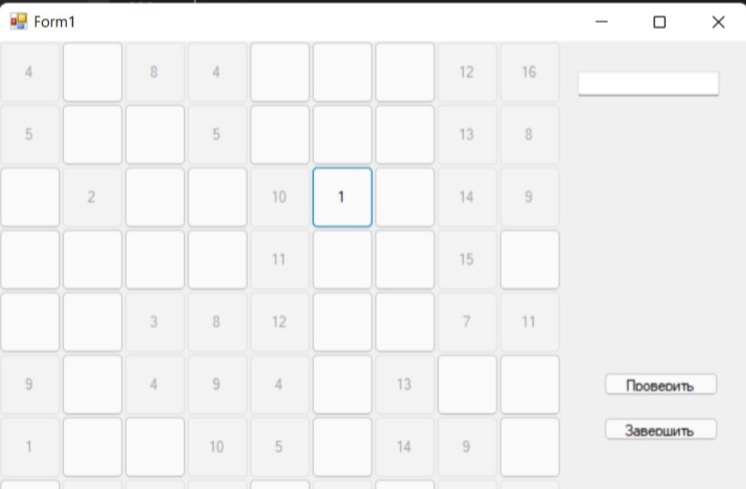
}

pressedButton.Text = num.ToString();

}

}

Тест пройден



6.2 Тестирование метода 2

private void button1\_Click(object sender, EventArgs e)//функция проверки

/\*Необходимо проверить правильность ввода пользователем значений

\* Для этого сравним то что ввел пользователь с тем, что лежит в массиве map

\* Если пользователь ввел ошибочное значение, то выводи сообщение "Неверно!"

\* Если пользователь ввел все правильно, то сообщение "Верно!"

\* После победы завершаем игру и генерируем поле сначала

\*/

{

for(int i = 0; i < n\*n; i++)

{

for(int j = 0; j < n\*n; j++)

{

var btnText = buttons[i, j].Text;

if(btnText != map[i,j].ToString())

{

MessageBox.Show("Неверно!");

return;

}

}

}

MessageBox.Show("Верно!");

for(int i = 0; i < n\*n; i++)

{

for(int j = 0; j < n\*n; j++)

{

this.Controls.Remove(buttons[i, j]);

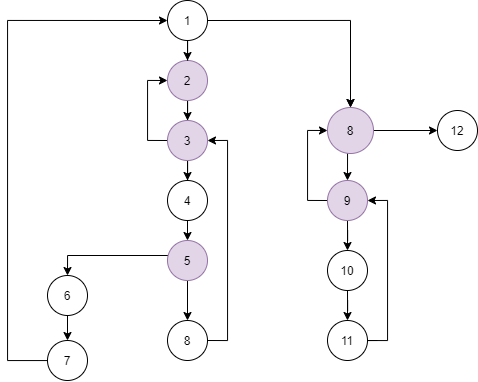
}

}

GenerateMap();

}

Потоковый граф:



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **№** | **Входные данные** | **Ожидаемый результат** | **Полученный результат** | **Вывод** |
| 1 | Правильный ввод судоку | Правильно | Правильно | Пройден |
| 2 | Неправильный ввод судоку | Неправильно | Неправильно | Пройден |

В данном методе есть два вложенных цикла, они не влияют на результат работы программы, от них зависит время работы, поэтому рассмотрим два теста:

* 1. Тестирование остальных методов

Метод завершения игры и обновления поля.

private void button2\_Click(object sender, EventArgs e)//функция для завершения игры

{

for (int i = 0; i < n \* n; i++)

{

for (int j = 0; j < n \* n; j++)

{

this.Controls.Remove(buttons[i, j]);

}

}

GenerateMap();

}

Требуется один тест, потому что цикломатическая сложность = 1.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Входные данные | Ожидаемый результат | Полученный результат | Вывод |
| 1 | Нажатие кнопки «завершение» | Завершение текущей игры и обновление поля |  | Пройден |

**Функциональное тестирование**

Одним из наиболее трудоемких этапов создания программного продукта является тестирование. Причем доля стоимости тестирования в общей стоимости разработки имеет тенденцию возрастать при увеличении сложности комплексов программ и повышении требований к качеству. В связи с этим большое внимание уделяется выбору стратегии и методов тестирования.

Тестированием называется процесс выполнения программы с целью обнаружения ошибки.

Исходными данными для этапа тестирования явились техническое задание и разработанные на предыдущих этапах структурная и функциональная схемы программного продукта.

Эксперименты показали, что с точки зрения нахождения ошибок , достаточно эффективными являются методы ручного контроля.Поэтому один или несколько из них должны использоваться в каждом программном проекте. Методы ручного контроля предназначены для периода разработки, когда программа закодирована. Доказано, что эти методы способствуют существенному увеличению производительности и повышению надежности программ и с их помощью можно находить от 30 до 70% ошибок логического проектирования и кодирования.

Основными методами ручного тестирования являются:

- инспекция исходного текста;

- сквозные просмотры;

- просмотры за столом;

- обзоры программ.

Также одним из способов проверки программ является стратегия тестирования, называемая стратегией «черного ящика» или тестированием с управлением по данным. В этом случае программа рассматривается как «черный ящик» и такое тестирование имеет целью выяснение обстоятельств, в которых поведение программы не соответствует спецификации.

При выборе стратегии тестирования был выбран метод «черного ящика». При тестировании данным методом, тестировщик имеет доступ к программному обеспечению только через интерфейс, что и заказчик или пользователь.

При тестировании данного продукта соблюдались следующие основные принципы:

-Предполагаемые результаты должны быть известны до тестирования.

-Следует избегать тестирования программы автором.

-Необходимо досконально изучать результаты каждого теста.

-Необходимо проверять действия программы на неверных данных.

-Необходимо проверять программу на неожиданные побочные эффекты.

-Удачным считается тест, который обнаруживает хотя бы одну еще не обнаруженную ошибку.

-Вероятность наличия ошибки в части программы пропорциональна количеству ошибок, уже обнаруженных в этой части.

Методом «черного ящика» производилась проверка правильности выполнения условий вызова той или иной функции и т.п. При дальнейшей реализации программы ошибки были устранены и применены меры по предотвращению подобных ошибок.

Ниже в таблице приведены тесты готового программного продукта по принципу «черного ящика»:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Номер теста | Назначение теста | Значение исходных данных | Ожидаемый результат | Реакция программы | Вывод |
| 1 | Проверить выводится ли сообщение о вводе имени в строку | Вывод сообщения | Вывод сообщения  «Введите пожалуйста имя игрока в поле в правом верхнем углу» |  | Правильно |
| 2 | Проверить выбирается ли число | Выбирается число | Выбирается число |  | Правильно |
| 3 | Проверить правильность работы кнопки «проверить» | Нажатие кнопки проверка | Выдать сообщение о ошибках |  | Правильно |
| 4 | Проверить правильность работы кнопки «завершить» | Нажатие кнопки завершить | Генерирует новые значения в поле и воводится сообщение о вводе имени |  | Правильно |